

UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería



**Control multiparamétrico en generación automática de
música: Extensión del sistema EMOPIA mediante información
de tonalidad y métodos de inferencia especializados**

Trabajo de graduación presentado por Samuel Alejandro Chamalé Rac
para optar al grado académico de Licenciado en Ingeniería en Ciencias
de la Computación y Sistemas de la Información

Guatemala,

2025

UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería



**Control multiparamétrico en generación automática de
música: Extensión del sistema EMOPIA mediante información
de tonalidad y métodos de inferencia especializados**

Trabajo de graduación presentado por Samuel Alejandro Chamalé Rac
para optar al grado académico de Licenciado en Ingeniería en Ciencias
de la Computación y Sistemas de la Información

Guatemala,

2025

Vo.Bo.:

(f) 
PhD. Alan Reyes

Tribunal Examinador:

(f) 
PhD. Alan Reyes

(f) _____
Ing. Bacilio Bolaños

Fecha de aprobación: Guatemala, 14 de noviembre de 2025.

Prefacio

La presente investigación surge del interés por explorar las capacidades de la música computacional mediante la generación de composiciones a través de métodos de aprendizaje profundo. El trabajo se centra en el estudio de los procesos de inferencia y el impacto de distintas variables en la generación musical, utilizando como punto de partida los antecedentes del proyecto EMOPIA, orientado a la creación de música de piano basada en emociones. A partir de ello, se amplió el conjunto de variables de inferencia, se implementaron métodos adicionales y se propuso una nueva variable fundamentada en conceptos de teoría musical, entrenando un modelo desde cero que incorporara dicha extensión.

La motivación principal detrás de este trabajo proviene de la relación personal y profunda que tengo con la música, en especial con el piano. Aunque los avances en este campo han sido notables, muchos se encuentran en el ámbito empresarial, lo que limita la disponibilidad de recursos abiertos para la comunidad investigadora. Este proyecto busca, en parte, contribuir al desarrollo de iniciativas accesibles y de código abierto dentro del ámbito de la música generada por inteligencia artificial.

Durante el proceso de elaboración, uno de los principales desafíos fue comprender a profundidad los aspectos teóricos y técnicos necesarios para transformar la idea inicial en un trabajo concreto. A esto se sumaron las limitaciones derivadas de los recursos computacionales, especialmente durante las etapas de entrenamiento, donde los tiempos de procesamiento y los requisitos de hardware variaban considerablemente entre fases. Asimismo, la disponibilidad de datos musicales clasificados emocionalmente, gracias al esfuerzo de etiquetado manual de los creadores de EMOPIA, fue fundamental para la realización de este estudio.

Deseo expresar mi sincero agradecimiento al asesor Alan Reyes por su guía en la investigación y el trabajo técnico, y al profesor Alexander Bolaños por su apoyo en la redacción y seguimiento del proyecto. Extiendo también mi gratitud a mi familia: a mi madre Cony Rac por su constante apoyo emocional, a mi padre Oscar Chamalé y a mi hermana Andrea Chamalé, cuyo acompañamiento fue esencial durante todo el proceso.

Índice

Prefacio	v
Lista de figuras	xiii
Lista de cuadros	xv
Resumen	xvii
Abstract	xix
1. Introducción	1
1.1. Finalidades del trabajo	1
1.2. Delimitación y definición del tema	2
1.2.1. Alcance técnico	2
1.2.2. Definiciones clave	2
1.3. Metodología y procedimientos	2
1.3.1. Fase 1: Revisión del estado del arte	3
1.3.2. Fase 2: Evaluación de replicabilidad	3
1.3.3. Fase 3: Extensión del dataset	3
1.3.4. Fase 4: Adaptación arquitectónica y entrenamiento	3
1.3.5. Fase 5: Desarrollo de métodos de inferencia	3
1.3.6. Fase 6: Evaluación sistemática	3
1.4. Conclusiones principales	3
1.5. Estructura del documento	4
2. Antecedentes	5
3. Justificación	7
4. Objetivos	9
5. Marco teórico	11
5.0.1. Introducción al Marco Teórico	11
5.0.2. Música y Emoción	12

5.0.3. Música Computacional	16
5.0.4. Fundamentos de Aprendizaje Profundo y Redes Neuronales	20
5.0.5. Generación Musical mediante Aprendizaje Profundo	22
5.0.6. El Dataset EMOPIA	29
5.0.7. Fundamentos Neurocientíficos y Psicológicos	31
5.0.8. Aplicaciones de la Generación Musical Emocional	32
5.0.9. Sistemas Alternativos de Generación Musical y Justificación de Decisiones Tecnológicas	35
5.0.10. Limitaciones de Investigación Existente y Gaps Identificados	38
5.0.11. Síntesis del Marco Teórico e Integración Conceptual	40
6. Metodología	45
6.1. Fase 1: Revisión bibliográfica y fundamentación teórica	46
6.1.1. Descripción de actividades realizadas	46
6.1.2. Materiales y recursos utilizados	47
6.1.3. Productos de la Fase 1	47
6.2. Fase 2: Evaluación de replicabilidad del sistema EMOPIA	48
6.2.1. Descripción de actividades realizadas	48
6.2.2. Materiales y recursos utilizados	56
6.2.3. Productos de la Fase 2	57
6.2.4. Conexión con objetivos específicos	58
6.3. Fase 3: Extensión del dataset con información de tonalidad	59
6.3.1. Descripción de actividades realizadas	59
6.3.2. Materiales y recursos utilizados	71
6.3.3. Productos de la Fase 3	72
6.3.4. Conexión con objetivos específicos	73
6.4. Fase 4: Adaptación de la arquitectura del Transformer y entrenamiento del modelo	74
6.4.1. Descripción de actividades realizadas	74
6.4.2. Materiales y recursos utilizados	86
6.4.3. Productos de la Fase 4	87
6.4.4. Conexión con objetivos específicos	88
6.5. Fase 5: Implementación de métodos de inferencia multiparamétrica	89
6.5.1. Descripción de actividades realizadas	89
6.5.2. Materiales y recursos utilizados	99
6.5.3. Productos de la Fase 5	100
6.5.4. Conexión con objetivos específicos	101
6.6. Fase 6: Evaluación y comparación del sistema extendido	102
6.6.1. Descripción de actividades realizadas	102
6.6.2. Resultados de la evaluación	113
6.6.3. Materiales y recursos utilizados	114
6.6.4. Productos de la Fase 6	115
6.6.5. Conexión con objetivos específicos	117
6.6.6. Desafíos encontrados y soluciones implementadas	118
6.6.7. Reflexiones sobre la fase	119

7. Resultados	121
7.1. Extensión del dataset EMOPIA con información de tonalidad	121
7.1.1. Características del dataset extendido	121
7.1.2. Vocabulario de tonalidad	122
7.1.3. Distribución de longitudes de secuencias	122
7.2. Modelo Transformer extendido y entrenamiento	123
7.2.1. Características arquitectónicas del modelo	123
7.2.2. Métricas de entrenamiento	124
7.2.3. Convergencia por dimensión	124
7.2.4. Comparación de velocidad de entrenamiento	125
7.3. Sistema de inferencia multiparamétrica	125
7.3.1. Modos de inferencia implementados	125
7.3.2. Parámetros de sampling por defecto	125
7.3.3. Capacidades del sistema de inferencia	125
7.4. Evaluación de adherencia al condicionamiento	126
7.4.1. Configuración de experimentos	126
7.4.2. Métricas de evaluación implementadas	127
7.4.3. Tiempos de ejecución del pipeline	127
7.5. Comparación cuantitativa: modelo extendido vs. modelo original	128
7.5.1. Adherencia por modo de inferencia	128
7.5.2. Métricas de distancia numérica y correlación tonal	129
7.5.3. Comparación de adherencia emocional	130
7.5.4. Adherencia tonal del modelo scratch	130
7.5.5. Patrones de adherencia a tokens de bajo nivel	130
7.6. Resumen cuantitativo de resultados	130
8. Discusión	133
8.1. Extensión exitosa del dataset con información de tonalidad	133
8.1.1. Alta tasa de procesamiento y robustez del pipeline	133
8.1.2. Integración natural de la dimensión de tonalidad	134
8.2. Convergencia excepcional de la dimensión de tonalidad durante entrenamiento	134
8.2.1. Factores que explican la alta convergencia de tonalidad	134
8.2.2. Implicaciones del patrón de convergencia diferenciado	135
8.3. Limitaciones de los métodos de inferencia para control multiparamétrico	136
8.3.1. Análisis del bajo rendimiento en adherencia categórica	136
8.3.2. Diferencias entre modos de inferencia	137
8.4. Validación del sistema extendido: Avances y limitaciones	137
8.4.1. Evidencia de aprendizaje efectivo de tonalidad	138
8.4.2. Comparación con modelo pretrained: Trade-offs inesperados	138
8.5. Factores que influyen en la adherencia al condicionamiento	139
8.5.1. Jerarquía de controlabilidad	139
8.5.2. Influencia del tamaño del vocabulario	139
8.5.3. Rol del contexto musical	139
8.6. Implicaciones para investigación futura	139
8.6.1. Métodos de condicionamiento alternativos	140
8.6.2. Arquitecturas especializadas para control	140
8.6.3. Evaluación más allá de adherencia categórica	140
8.7. Limitaciones del estudio	141

8.7.1. Limitaciones del dataset	141
8.7.2. Limitaciones de evaluación	141
8.7.3. Limitaciones de alcance	141
8.8. Contribuciones y valor del trabajo	141
8.8.1. Primera integración de tonalidad en EMOPIA	141
8.8.2. Análisis sistemático de métodos de condicionamiento	141
8.8.3. Marco de evaluación reproducible	142
8.8.4. Identificación de desafíos fundamentales	142
8.9. Conclusión de la discusión	142
9. Conclusiones	143
10. Recomendaciones	147
10.1. Recomendaciones sobre el dataset y preprocesamiento	147
10.2. Recomendaciones sobre arquitectura y entrenamiento	148
10.3. Recomendaciones sobre métodos de inferencia	149
10.4. Recomendaciones sobre evaluación	150
10.5. Recomendaciones metodológicas	150
10.6. Direcciones prioritarias para investigación inmediata	151
11. Bibliografía	153

Lista de figuras

1.	Modelo Circumplejo de Russell aplicado a la emoción musical. Los cuadrantes representan las cuatro categorías emocionales utilizadas en el dataset EMOPIA.	13
2.	Representación esquemática del perfil de tonalidad mayor según Krumhansl-Schmuckler. Las alturas indican el peso de cada clase de pitch en la tonalidad.	17
3.	Flujo del algoritmo de Krumhansl-Schmuckler para detección automática de tonalidad. El proceso transforma un clip MIDI en un vector de distribución de pitch que se compara con perfiles teóricos de tonalidades.	19
4.	Arquitectura básica del Transformer mostrando las capas principales del encoder y decoder. Las conexiones residuales y normalizaciones de capa permiten entrenar modelos muy profundos.	21
5.	Modelo conceptual integrado del marco teórico. El sistema extendido EMOPIA integra conceptos de cuatro dominios teóricos principales.	40
6.	Flujo metodológico del trabajo de graduación. Las fases se diseñaron para abordar secuencialmente los objetivos específicos, desde la fundamentación teórica hasta la evaluación comparativa del sistema extendido.	45
7.	Configuración del entorno de desarrollo local, mostrando versiones de CUDA 12.9, Python 3.13.2, y PyTorch 2.8.0+cu129 instaladas en Windows 11 con GPU RTX 2060.	52
8.	Tiempo de entrenamiento de una época completa en el entorno local (RTX 2060): 1 minuto 27 segundos para procesar 206 batches con un modelo de 39.2 millones de parámetros.	53
9.	Especificaciones del pod de RunPod utilizado para entrenamiento, mostrando GPU RTX 5090, 16 vCPU, 141GB RAM, y pricing de \$0.9011/hora.	55
10.	Configuración del entorno de desarrollo en RunPod, mostrando CUDA 12.8, Python 3.12.3, y PyTorch 2.8.0+cu128 con dependencias NVIDIA completas.	56
11.	Tiempo de entrenamiento de una época completa en RunPod (RTX 5090): 13.74 segundos, representando una aceleración de $6.35\times$ respecto al entorno local. La consistencia del loss inicial (1.5370) valida la correcta configuración del entorno.	56
12.	Comandos de configuración del entorno virtual Python 3.9 para procesamiento de sincronización MIDI con madmom	60

13.	Verificación de instalación de Python 3.9 y dependencias críticas (madmom, miditoolkit) en el entorno virtual	61
14.	Configuración de FFmpeg para madmom en Windows 11	61
15.	Modificación del listado de archivos en synchronizer.py para filtrar por MP3 disponibles	62
16.	Adición de manejo de excepciones en analyzer.py para archivos MIDI con tiempo cero	62
17.	Comandos de ejecución del script synchronizer.py	63
18.	Comando de ejecución del script analyzer.py	63
19.	Estructura de directorios generados durante el pipeline de preprocesamiento con conteo de archivos en cada etapa	63
20.	Definición de perfiles tonales de Krumhansl-Kessler en midi2corpus.py	65
21.	Función de cálculo de histograma de clases de pitch en midi2corpus.py	65
22.	Función de detección de tonalidad mediante algoritmo de Krumhansl-Schmuckler	65
23.	Invocación del algoritmo de detección de tonalidad y almacenamiento en metadata	66
24.	Ejecución del script midi2corpus.py mostrando el progreso y conteo de archivos procesados	66
25.	Definición del evento compuesto con dimensión de tonalidad	67
26.	Inserción del evento de tonalidad al inicio de la secuencia	67
27.	Ejecución del script corpus2events.py procesando archivos del corpus	67
28.	Distribución de longitudes de secuencias de eventos generadas (media: 442.77 tokens, desviación estándar: 203.68 tokens)	68
29.	Tamaños de vocabulario por dimensión, incluyendo tonalidad con 25 valores únicos	68
30.	Ejecución del script event2words.py mostrando los tamaños de vocabulario por dimensión	69
31.	Visualización de los tamaños de vocabulario por dimensión, destacando la dimensión de tonalidad (Key) con 25 valores únicos	69
32.	Configuración de longitud máxima y formato de compilación	70
33.	Salida de compile.py mostrando las dimensiones finales del dataset con 9 características	70
34.	Ejecución del script compile.py mostrando la compilación y dimensiones finales del dataset	70
35.	Configuración dinámica de tamaños de embeddings según número de tokens	75
36.	Creación de capa de embedding para la variable de tonalidad	75
37.	Capa de proyección para predicción de tonalidad	75
38.	Concatenación del embedding de tonalidad en forward_hidden	76
39.	Proyección de tonalidad en forward_output	76
40.	Cálculo de loss para la dimensión de tonalidad	76
41.	Configuración del clúster de RunPod para entrenamiento del modelo extendido, mostrando GPU RTX 5090 con 24GB VRAM, 16 vCPU, 141GB RAM, y pricing de \$0.9011/hora.	77
42.	Interfaz de Jupyter Lab conectado a RunPod con múltiples terminales abiertas	78
43.	Derivación dinámica de la columna de emoción	79
44.	Cálculo dinámico de loss promedio	79
45.	Evolución del loss total durante el entrenamiento de 875 épocas. Se observa una reducción consistente del 95.59 % desde 1.8109 hasta 0.0798.	82

46.	Evolución de losses individuales por dimensión durante el entrenamiento. La dimensión de tonalidad (key) muestra convergencia rápida y estable, alcanzando 98.46 % de reducción.	83
47.	Estadísticas comparativas de convergencia por dimensión. Se incluyen valores finales, reducción absoluta y reducción relativa para cada componente del modelo.	84
48.	Implementación del modo inference-normal en main_cp.py	91
49.	Configuración de parámetros para modo determinístico	92
50.	Función de parsing de condiciones en main_cp.py	93
51.	Función de aplicación de condiciones en la primera fila de generación	94
52.	Función de parsing de tokens forzados en main_cp.py	95
53.	Enforcement de tokens forzados después del sampling	96
54.	Activación del modo forced en main_cp.py	96
55.	Estructura de metadata guardada con cada composición generada	99
56.	Implementación de muestreo estratificado de valores para tokens de condicionamiento	103
57.	Extracción de valores numéricos de tokens para cálculo de métricas de distancia	105
58.	Extracción de histograma de clases de pitch ponderado por duración	106
59.	Implementación del algoritmo de Krumhansl-Schmuckler para detección de tonalidad	106
60.	Cálculo de correlación con tonalidad objetivo como métrica de distancia continua	107
61.	Invocación del clasificador EMOPIA para predicción de cuadrante emocional .	108
62.	Configuración de modelos, modos de inferencia y variables de condicionamiento	111

Lista de cuadros

1.	Comparación de paradigmas de inferencia	27
2.	Distribución de clips por cuadrante emocional en EMOPIA	31
3.	Comparación de sistemas de generación musical	36
4.	Mapeo entre marco teórico y objetivos específicos	42
5.	Comparación de servicios de GPU en la nube	54
6.	Parámetros de sampling por defecto para cada token. Índica ausencia de restricción (comportamiento estocástico puro o determinado por temperatura).	91
7.	Argumentos de línea de comandos para inferencia multiparamétrica	98
8.	Comparación de adherencia por modo de inferencia	113
9.	Métricas de distancia numérica por modo de inferencia	114
10.	Tamaños de vocabulario del dataset extendido	122
11.	Estadísticas de longitudes de secuencias	123
12.	Especificaciones arquitectónicas del modelo extendido	123
13.	Métricas globales de entrenamiento del modelo extendido	124
14.	Convergencia de loss por dimensión durante entrenamiento	124
15.	Comparación de velocidad de entrenamiento entre entornos	125
16.	Modos de inferencia implementados	126
17.	Parámetros de sampling por defecto por token	126
18.	Capacidades del sistema de inferencia multiparamétrica	127
19.	Diseño experimental del pipeline de evaluación	127
20.	Métricas de evaluación de adherencia implementadas	128
21.	Tiempos de ejecución del pipeline de evaluación	128
22.	Adherencia categórica por modo de inferencia y modelo	129
23.	Métricas de distancia numérica y correlación tonal	129
24.	Comparación de adherencia emocional entre modelos	130
25.	Adherencia tonal del modelo scratch por modo de inferencia	130
26.	Patrones de adherencia a tokens de bajo nivel	131
27.	Resumen cuantitativo de resultados por objetivo	131

Resumen

La generación automática de música mediante inteligencia artificial enfrenta el desafío fundamental del control limitado sobre las características de las composiciones generadas. Este trabajo extiende el sistema EMOPIA de generación de música de piano condicionada por emociones mediante la incorporación de información de tonalidad musical como variable de control adicional e implementación de métodos de inferencia multiparamétrica.

El dataset EMOPIA original fue extendido exitosamente con información de tonalidad detectada algorítmicamente, resultando en 824 secuencias tokenizadas (77 % del dataset original de 1,071 clips) con 9 dimensiones incluyendo un vocabulario de 25 tokens de tonalidad. La arquitectura Transformer fue adaptada para procesar esta dimensión adicional, entrenando un modelo de 39.2 millones de parámetros durante 875 épocas con convergencia del 98.46 % en la dimensión de tonalidad.

Se implementaron cuatro modos de inferencia (normal, determinístico, primed, forced) que proporcionan diferentes niveles de control sobre la generación. La evaluación sistemática mediante 192 experimentos reveló tasas de adherencia categórica moderadas, con máximo 50 % para emoción y 41.67 % para tonalidad, aunque las métricas de correlación tonal alcanzaron valores prometedores de hasta 0.76.

Los resultados demuestran que aunque el modelo aprendió representaciones significativas de tonalidad, el control preciso sobre generación musical autoregresiva permanece como desafío fundamental debido a la naturaleza probabilística de los modelos y conflictos entre restricciones múltiples. Se recomienda desarrollar métodos de condicionamiento jerárquico, ampliar significativamente el dataset, y diseñar arquitecturas especializadas que incorporen conocimiento musical explícito para avanzar hacia sistemas verdaderamente controlables.

Abstract

Automatic music generation using artificial intelligence faces the fundamental challenge of limited control over the characteristics of generated compositions. This work extends the EMOPIA emotion-conditioned piano music generation system by incorporating musical key information as an additional control variable and implementing multiparametric inference methods.

The original EMOPIA dataset was successfully extended with algorithmically detected key information, resulting in 824 tokenized sequences (77 % of the original 1,071 clips) with 9 dimensions including a vocabulary of 25 key tokens. The Transformer architecture was adapted to process this additional dimension, training a 39.2 million parameter model for 875 epochs with 98.46 % convergence on the key dimension.

Four inference modes were implemented (normal, deterministic, primed, forced) providing different levels of control over generation. Systematic evaluation through 192 experiments revealed moderate categorical adherence rates, with maximum 50 % for emotion and 41.67 % for key, although tonal correlation metrics reached promising values up to 0.76.

Results demonstrate that while the model learned meaningful key representations, precise control over autoregressive music generation remains a fundamental challenge due to the probabilistic nature of models and conflicts between multiple constraints. Recommendations include developing hierarchical conditioning methods, significantly expanding the dataset, and designing specialized architectures that incorporate explicit musical knowledge to advance toward truly controllable systems.

CAPÍTULO 1

Introducción

La generación automática de música mediante inteligencia artificial ha experimentado avances significativos en los últimos años, especialmente con el desarrollo de modelos basados en aprendizaje profundo. Sin embargo, estos sistemas enfrentan un desafío fundamental: el control limitado sobre las características específicas de las composiciones generadas. Mientras que los modelos actuales pueden producir música coherente y estilísticamente apropiada, los usuarios tienen capacidad restringida para dirigir aspectos específicos como la emoción, tonalidad, o características estructurales de la música resultante.

1.1. Finalidades del trabajo

El presente trabajo de graduación tiene como finalidad principal extender las capacidades de control del sistema EMOPIA (Emotion-conditioned Piano music generation) mediante la incorporación de información de tonalidad musical como variable de condicionamiento adicional. Este objetivo responde a la necesidad práctica de sistemas de generación musical que ofrezcan mayor controlabilidad para aplicaciones en composición asistida, educación musical, y producción multimedia.

Las finalidades específicas incluyen:

1. Demostrar la viabilidad técnica de integrar conceptos de teoría musical computacional (específicamente tonalidad) en modelos de aprendizaje profundo para generación musical.
2. Desarrollar métodos de inferencia que permitan control multiparamétrico simultáneo sobre diferentes aspectos de la generación musical.
3. Establecer un marco de evaluación sistemático para cuantificar la adherencia de composiciones generadas a condiciones especificadas.

4. Contribuir al avance del campo proporcionando un dataset extendido, arquitecturas adaptadas, y metodologías reproducibles para investigación futura.

1.2. Delimitación y definición del tema

Este trabajo se enfoca específicamente en la extensión del sistema EMOPIA, un modelo Transformer autoregresivo para generación de música de piano condicionada por emociones. La investigación se delimita a:

1.2.1. Alcance técnico

- **Dominio musical:** Música de piano en formato MIDI, específicamente el género pop representado en el dataset EMOPIA.
- **Variable de extensión:** Tonalidad musical detectada mediante el algoritmo de Krumhansl-Schmuckler, limitada a las 24 tonalidades del sistema temperado occidental (12 mayores y 12 menores).
- **Arquitectura base:** Modelo Transformer con representación Compound Word de 8 dimensiones, extendido a 9 dimensiones.
- **Condicionamiento:** Cuatro categorías emocionales del modelo de Russell (cuadrantes de arousal-valencia) más tonalidad y parámetros de bajo nivel.

1.2.2. Definiciones clave

Generación musical condicionada se refiere a la capacidad de un modelo de producir música que adhiera a características específicas proporcionadas como entrada. En este contexto, el condicionamiento puede ser emocional (qué emoción debe transmitir la música), tonal (en qué tonalidad debe estar), o estructural (características de tempo, dinámica, duración).

Tonalidad musical en este trabajo se define como el centro tonal alrededor del cual se organiza una composición, caracterizado por una tónica (nota fundamental) y un modo (mayor o menor). La detección algorítmica se basa en perfiles de distribución de clases de pitch establecidos por la investigación en cognición musical.

Inferencia multiparamétrica denota la capacidad de especificar y controlar múltiples aspectos de la generación simultáneamente, en contraste con sistemas que solo permiten condicionamiento unidimensional.

1.3. Metodología y procedimientos

El desarrollo del trabajo siguió una metodología estructurada en seis fases principales:

1.3.1. Fase 1: Revisión del estado del arte

Análisis exhaustivo de literatura en generación musical con IA, identificando el sistema EMOPIA como punto de partida óptimo por su arquitectura moderna, dataset anotado, y código disponible públicamente.

1.3.2. Fase 2: Evaluación de replicabilidad

Validación de la capacidad de replicar el sistema original, estableciendo entornos de desarrollo local y en la nube. Esta fase identificó que solo el 77 % del dataset original pudo ser recuperado debido a videos no disponibles en YouTube.

1.3.3. Fase 3: Extensión del dataset

Implementación del algoritmo de Krumhansl-Schmuckler para detección automática de tonalidad e integración en el pipeline de preprocesamiento. Resultó en 824 secuencias tokenizadas con 9 dimensiones.

1.3.4. Fase 4: Adaptación arquitectónica y entrenamiento

Modificación del modelo Transformer para procesar la dimensión adicional de tonalidad. Entrenamiento del modelo extendido durante 875 épocas utilizando infraestructura de GPU en la nube (4× RTX 5090).

1.3.5. Fase 5: Desarrollo de métodos de inferencia

Implementación de cuatro modos de inferencia (normal, determinístico, primed, forced) con sistema de configuración flexible para control granular sobre la generación.

1.3.6. Fase 6: Evaluación sistemática

Desarrollo de pipeline automatizado que ejecutó 192 experimentos para evaluar adherencia al condicionamiento mediante métricas categóricas, numéricas, y de correlación tonal.

1.4. Conclusiones principales

La investigación demostró exitosamente la viabilidad técnica de integrar información de tonalidad en sistemas de generación musical basados en aprendizaje profundo. El modelo extendido alcanzó una convergencia del 98.46 % en la dimensión de tonalidad durante entrenamiento, superando dimensiones estructurales como tempo y pitch.

Sin embargo, la evaluación reveló que el control preciso sobre la generación permanece como desafío fundamental. Las tasas de adherencia categórica fueron moderadas (máximo 50 % para emoción, 41.67 % para tonalidad), aunque las métricas de correlación tonal mostraron valores más prometedores (hasta 0.76). Estos resultados sugieren que los modelos aprenden representaciones significativas pero las expresan de forma probabilística y contextual en lugar de determinística.

El trabajo establece contribuciones concretas: un dataset público extendido con información tonal, métodos de inferencia multiparamétrica implementados, y un marco de evaluación reproducible. Las limitaciones identificadas proporcionan dirección clara para investigación futura, particularmente en el desarrollo de métodos de condicionamiento jerárquico y arquitecturas especializadas para música.

1.5. Estructura del documento

El presente documento se organiza en los siguientes capítulos:

- **Capítulo 2 - Antecedentes:** Presenta el contexto histórico y técnico de la generación musical con IA.
- **Capítulo 3 - Justificación:** Establece la motivación y relevancia del trabajo.
- **Capítulo 4 - Objetivos:** Define el objetivo general y los cinco objetivos específicos.
- **Capítulo 5 - Alcances y Límites:** Delimita el ámbito de la investigación.
- **Capítulo 6 - Marco Teórico:** Fundamenta los conceptos técnicos y musicales utilizados.
- **Capítulo 7 - Metodología:** Detalla las seis fases del desarrollo del proyecto.
- **Capítulo 8 - Resultados:** Presenta los hallazgos cuantitativos de cada objetivo.
- **Capítulo 9 - Discusión:** Interpreta y analiza los resultados obtenidos.
- **Capítulo 10 - Conclusiones:** Sintetiza los logros y aprendizajes del trabajo.
- **Capítulo 11 - Recomendaciones:** Propone direcciones para investigación futura.

Este trabajo representa un paso hacia sistemas de generación musical más controlables, demostrando que la integración de conocimiento de dominio musical con técnicas modernas de aprendizaje profundo es un enfoque prometedor que merece investigación continua.

CAPÍTULO 2

Antecedentes

El presente trabajo se fundamenta en dos proyectos previos que proporcionaron tanto el dataset como el código base como las herramientas de preprocesamiento necesarias para el desarrollo del sistema de generación musical propuesto.

En primer lugar, el proyecto EMOPIA [1] constituye el antecedente directo de esta investigación. EMOPIA proporciona un dataset multimodal de música de piano clasificado emocionalmente [2], originalmente compuesto por 1,087 clips de audio etiquetados manualmente según cuatro cuadrantes emocionales (Q1-Q4) basados en las dimensiones de arousal y valencia. El sistema incluye además una arquitectura de transformer para la generación de música condicionada por emociones, así como un clasificador preentrenado capaz de reconocer las categorías emocionales en composiciones generadas. Para la presente investigación, se utilizó el código base del transformer de EMOPIA, adaptándolo para incorporar variables adicionales de condicionamiento. Del dataset original, fue posible recuperar y procesar 854 clips utilizables, dado que algunos de los videos de YouTube empleados como fuente ya no se encontraban disponibles al momento de la recopilación. El clasificador preentrenado de EMOPIA se empleó posteriormente en la fase de evaluación para analizar la adherencia emocional de las composiciones generadas por el modelo extendido.

En segundo lugar, se emplearon los scripts de preprocesamiento desarrollados en el proyecto Compound Word Transformer [3], específicamente las herramientas para la sincronización temporal y el análisis simbólico de archivos MIDI. Estos scripts implementan técnicas de beat y downbeat tracking mediante la librería madmom, realizan la interpolación de 480 ticks entre pulsos adyacentes para mapear el tiempo absoluto en unidades de tick, e infieren los cambios de tempo a partir de los intervalos temporales entre pulsos consecutivos. Adicionalmente, el proyecto incluye algoritmos basados en reglas para la extracción de melodías y el reconocimiento de acordes a partir de representaciones simbólicas. Estas herramientas fueron esenciales para transformar los archivos MIDI del dataset EMOPIA en representaciones estructuradas que pudieran ser procesadas por el modelo de generación, permitiendo mantener información temporal precisa sin cuantización excesiva y preservando así matices expresivos para su posterior uso en la generación musical.

CAPÍTULO 3

Justificación

La generación automática de música emocional constituye un campo de investigación con creciente relevancia tanto teórica como aplicada. En particular, el estudio de la generación de música para piano con control explícito de las emociones permite profundizar en los vínculos entre inteligencia artificial, percepción emocional y creatividad musical, abordando un problema que combina dimensiones técnicas, expresivas y cognitivas.

Aunque en los últimos años se han logrado avances notables en la generación musical mediante inteligencia artificial, gran parte de los desarrollos con mayor impacto permanecen en el ámbito privado. Empresas como Suno [4] o Eleven Music [5] han presentado sistemas comerciales capaces de generar composiciones completas a partir de descripciones textuales, pero estos modelos no son de acceso abierto y sus parámetros internos, datos de entrenamiento y mecanismos de control emocional son propietarios. Esta situación limita la transparencia, la replicabilidad científica y la posibilidad de extender dichos sistemas desde la investigación académica.

En contraste, los proyectos de código abierto, como DiffRhythm [6], demuestran que es posible desarrollar arquitecturas de generación musical completas que combinan eficiencia con apertura y posibilidad de personalización. No obstante, la mayoría de estos esfuerzos se centran en aspectos estructurales o estilísticos de la música, dejando un espacio importante para la investigación enfocada en la representación y el control emocional explícito de la música instrumental, particularmente del piano como instrumento expresivo.

El enfoque basado en la parametrización para la inferencia —esto es, el ajuste explícito de variables como emoción, tempo o dinámica durante el proceso generativo— aporta un nivel superior de control que permite orientar la creación automática hacia objetivos emocionales específicos. Este tipo de modelado contribuye a la comprensión de la relación entre características musicales estructurales y su percepción emocional, habilitando además aplicaciones concretas en producción audiovisual, educación musical, videojuegos y terapias basadas en sonido.

Por otra parte, el impacto de la inteligencia artificial en la industria musical es cada vez

más significativo. Estudios recientes sugieren que, sin intervención adecuada, los trabajadores del sector musical podrían perder cerca de un cuarto de sus ingresos debido a la adopción de sistemas de IA en los próximos cuatro años, y que roles creativos se encuentran entre los más susceptibles a la automatización [7]. Este contexto refuerza la relevancia de desarrollar soluciones abiertas, transparentes y controlables que permitan a creadores humanos mantener agencia sobre el proceso creativo y contribuyan al desarrollo de una industria musical centrada en lo humano.

CAPÍTULO 4

Objetivos

Objetivo General:

Extender un sistema de generación de música de piano basado en aprendizaje profundo mediante la incorporación de una variable de control adicional fundamentada en teoría musical y la implementación de métodos de inferencia multiparamétrica, utilizando como base el sistema y dataset EMOPIA.

Objetivos Específicos:

1. Extender el dataset EMOPIA mediante la incorporación de información de tonalidad musical como variable de control, aplicando el algoritmo de detección de tonalidad de Krumhansl-Schmuckler al conjunto de datos procesado.
2. Adaptar la arquitectura del transformer de EMOPIA para incorporar la nueva variable de tonalidad como parámetro de condicionamiento adicional durante el entrenamiento y la inferencia del modelo generativo.
3. Implementar distintos métodos de inferencia que permitan la generación condicionada por múltiples parámetros simultáneos, incluyendo emoción, tonalidad, duración y velocidad, con control explícito sobre cada variable.
4. Desarrollar un pipeline de evaluación que permita cuantificar la adherencia de las composiciones generadas a las variables de condicionamiento especificadas durante la inferencia.
5. Evaluar y comparar el desempeño del modelo extendido con respecto al modelo original de EMOPIA mediante el análisis cuantitativo de la adherencia a las variables de condicionamiento en las composiciones generadas.

CAPÍTULO 5

Marco teórico

5.0.1. Introducción al Marco Teórico

El presente marco teórico establece las bases conceptuales, metodológicas y tecnológicas necesarias para abordar el objetivo general de este trabajo de graduación: la extensión de un sistema de generación de música de piano mediante la incorporación de variables de control adicionales fundamentadas en teoría musical y la implementación de métodos de inferencia multiparamétrica.

Propósito y Alcance

Este marco teórico cumple con las siguientes funciones específicas en relación con los objetivos del proyecto:

1. **Fundamentación emocional:** Establece el modelo teórico de representación emocional (modelo circumplejo de Russell) que constituye la base del sistema EMOPIA y que guiará la generación musical condicionada emocionalmente.
2. **Fundamentación musical computacional:** Revisa los conceptos de teoría musical computacional, incluyendo la detección automática de tonalidad mediante el algoritmo de Krumhansl-Schmuckler, directamente relacionado con el Objetivo Específico 1.
3. **Fundamentación arquitectónica:** Examina las arquitecturas de redes neuronales profundas, específicamente los Transformers, que serán adaptados en el Objetivo Específico 2 para incorporar condicionamiento multiparamétrico.
4. **Fundamentación metodológica:** Revisa técnicas de generación condicional y métodos de inferencia que respaldan el Objetivo Específico 3 sobre generación con múltiples parámetros simultáneos.

- 5. Fundamentación evaluativa:** Identifica métricas y metodologías de evaluación relevantes para los Objetivos Específicos 4 y 5 sobre cuantificación de adherencia a variables de condicionamiento.

Estructura del Marco Teórico

El marco teórico se organiza en las siguientes secciones principales:

- **Música y Emoción:** Revisa los modelos teóricos de representación emocional en música y las correlaciones entre características musicales y percepciones emocionales.
- **Música Computacional:** Examina los fundamentos de la teoría musical computacional, incluyendo detección automática de tonalidad y análisis de estructuras musicales.
- **Generación Musical mediante Aprendizaje Profundo:** Analiza representaciones musicales, arquitecturas neuronales, y técnicas de generación condicional.
- **El Dataset EMOPIA:** Describe el recurso empírico fundamental sobre el cual se construye este trabajo.
- **Sistemas Alternativos y Justificación:** Compara sistemas de generación musical existentes y justifica las decisiones tecnológicas del proyecto.
- **Limitaciones y Gaps de Investigación:** Identifica las áreas donde este trabajo aporta conocimiento nuevo.
- **Síntesis del Marco Teórico:** Integra los conceptos presentados en un marco unificado que sustenta el proyecto.

5.0.2. Música y Emoción

Modelos de Representación Emocional en Música

La relación entre música y emoción ha sido objeto de estudio durante décadas, dando lugar a diversos modelos teóricos que intentan explicar cómo la música evoca y expresa emociones específicas.

Modelo Circumplejo de Russell El modelo más ampliamente utilizado en la investigación musical computacional es el modelo circumplejo de Russell [8], que organiza las emociones en un espacio bidimensional definido por dos ejes principales:

- **Valencia (Valence):** Representa la dimensión hedónica de la emoción, desde emociones positivas hasta negativas
- **Activación (Arousal):** Indica el nivel de excitación o energía asociado con la emoción

Este modelo genera cuatro cuadrantes principales:

1. **Alta valencia, alta activación:** Feliz-enérgico (euforia, alegría intensa)
2. **Alta valencia, baja activación:** Feliz-calmado (relajación, serenidad)
3. **Baja valencia, alta activación:** Triste-enérgico (angustia, tensión)
4. **Baja valencia, baja activación:** Triste-calmado (melancolía, tristeza suave)

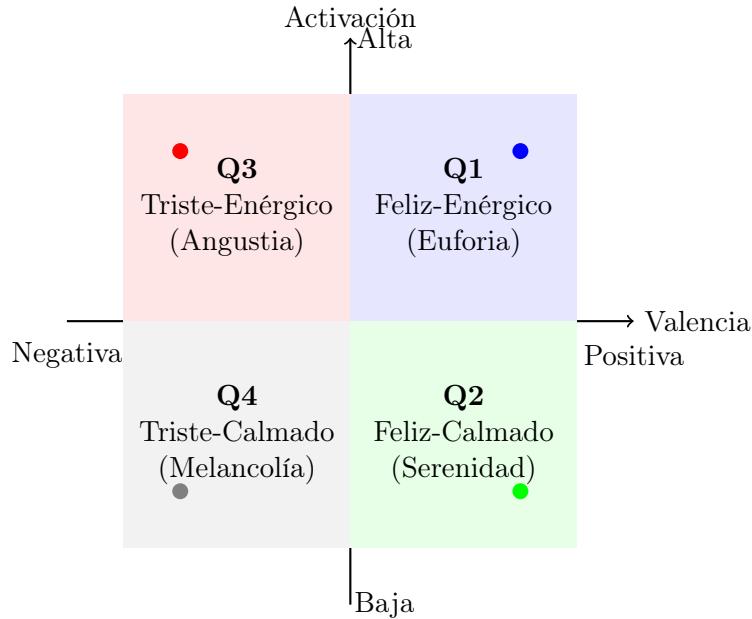


Figura 1: Modelo Circumplejo de Russell aplicado a la emoción musical. Los cuadrantes representan las cuatro categorías emocionales utilizadas en el dataset EMOPIA.

Investigaciones posteriores como la de Posner, Russell y Peterson [9] han validado la aplicabilidad de este modelo en el contexto de la neurociencia afectiva, mientras que Eerola y Vuoskoski [10] han comparado específicamente los modelos dimensionales y categóricos en el contexto musical.

Este modelo circumplejo constituye la base teórica fundamental para el presente trabajo de graduación, ya que define el espacio emocional bidimensional que guiará la generación musical. La adopción de este marco permite crear un sistema que no solo genera música, sino que la orienta específicamente hacia regiones emocionales precisas, facilitando aplicaciones terapéuticas y creativas donde se requiere un control emocional específico.

Modelo Categórico de Emociones Básicas Paralelamente, el modelo de emociones básicas propone la existencia de emociones universales fundamentales. Como indican Gu et al. [11], estas emociones básicas representan estados internos que pueden ser observados y medidos en comportamientos específicos. En el contexto musical, estas emociones se manifiestan a través de combinaciones específicas de elementos musicales.

Teorías de Percepción Emocional Musical Las teorías contemporáneas sobre percepción emocional musical incluyen:

- **Teoría de la Expresión:** La música expresa emociones mediante la imitación de patrones vocales y gestuales humanos
- **Teoría de la Inducción:** La música induce emociones a través de mecanismos psicológicos como la memoria episódica y el contagio emocional
- **Teoría Cognitivista:** Las emociones musicales surgen de la evaluación cognitiva de patrones musicales y su cumplimiento o violación de expectativas

Características Musicales y su Relación con Emociones

La investigación en psicología musical ha identificado correlaciones consistentes entre elementos musicales específicos y respuestas emocionales.

Características Temporales Liu et al. [12] demostraron que el tempo ejerce una influencia significativa en la percepción emocional tanto en músicos como en no músicos. Sus hallazgos indican que:

- **Tempo:** Velocidades rápidas (>120 BPM) se asocian con emociones de alta activación (alegría, excitación), mientras que tempos lentos (<60 BPM) evocan emociones de baja activación (tristeza, calma)
- **Ritmo:** Patrones rítmicos regulares y sincopados generan sensaciones de energía y movimiento
- **Métrica:** Métricas simples (4/4, 2/4) tienden a ser más estables emocionalmente que métricas complejas

Características Tonales y Tímbricas La investigación de Hailstone et al. [13] reveló que "no es lo que tocas, sino cómo lo tocas el timbre afecta significativamente la percepción de emoción en la música. Además:

- **Modo:** El modo mayor tradicionalmente se asocia con emociones positivas, mientras que el modo menor evoca emociones más melancólicas
- **Armonía:**
 - Consonancia: Acordes estables generan sensaciones de reposo y estabilidad
 - Disonancia: Tensiones armónicas crean expectativa y pueden evocar emociones de activación elevada
 - Progresiones armónicas: Secuencias como ii-V-I generan sensación de resolución y completitud

Expresión Musical y Comunicación Emocional Micallef Grimaud y Eerola [14] realizaron una comparación entre los enfoques de producción y percepción de la expresión emocional musical, encontrando que los músicos utilizan claves musicales específicas para comunicar emociones que son consistentemente percibidas por los oyentes.

La comprensión de estas correlaciones entre características musicales y respuestas emocionales es crucial para el desarrollo de sistemas generativos de este tipo, ya que permite establecer reglas de mapeo entre parámetros de condicionamiento emocional y elementos musicales específicos. Esto podría facilitar la implementación de algoritmos que traduzcan objetivos emocionales en decisiones compositivas concretas sobre tempo, armonía, ritmo y dinámica.

El Piano como Instrumento Expresivo

El piano posee características únicas que lo convierten en un instrumento ideal para la expresión emocional:

Capacidades Expresivas

- **Rango dinámico:** Desde pianissimo hasta fortissimo, permitiendo expresar un amplio espectro de intensidades emocionales
- **Rango tonal:** 88 teclas que abarcan desde frecuencias graves (27.5 Hz) hasta agudas (4186 Hz)
- **Polifonía:** Capacidad de ejecutar múltiples voces simultáneamente, creando texturas armónicas complejas
- **Control temporal:** Precisión en la articulación y timing de cada nota individual

Técnicas Interpretativas y Análisis Computacional Zhang et al. [15] presentaron ATEPP, un dataset de performance expresiva de piano transcrita automáticamente, que permite el análisis cuantitativo de las técnicas interpretativas emocionales:

- **Pedal de sostén:** Crea resonancias y conexiones armónicas que intensifican la expresión emocional
- **Rubato:** Flexibilidad temporal que permite expresar fraseo emocional
- **Toque:** Variaciones en el ataque y timbre de cada nota
- **Agógica:** Modificaciones expresivas del tempo

Portalés-Ricart et al. [16] identificaron el potencial de la investigación en performance expresiva computacional para aplicaciones en la producción musical popular, mientras que

Roth et al. [17] desarrollaron enfoques de aprendizaje automático para descubrir reglas de acciones de performance expresiva en música de jazz.

La elección del piano como instrumento objetivo en este trabajo de graduación se justifica por su extraordinaria capacidad expresiva y la disponibilidad del dataset EMOPIA específicamente diseñado para piano pop. Las características polifónicas del piano permiten expresar simultáneamente elementos melódicos, armónicos y rítmicos, facilitando la generación de composiciones completas que pueden transmitir eficazmente las emociones objetivo del sistema.

Conexión con el Trabajo de Graduación

La comprensión profunda de la relación entre música y emoción establecida en esta sección fundamenta directamente varios aspectos del presente trabajo:

- **Variable de condicionamiento emocional:** El modelo circumplejo de Russell constituye la base para la especificación de emociones objetivo en la generación musical (Objetivo Específico 3).
- **Evaluación de adherencia emocional:** Las correlaciones identificadas entre características musicales (tempo, modo, armonía) y percepciones emocionales proporcionan criterios para evaluar si las composiciones generadas corresponden a las emociones especificadas (Objetivos Específicos 4 y 5).
- **Control multiparamétrico:** La identificación de múltiples características musicales que contribuyen a la percepción emocional justifica la necesidad de un sistema con control sobre variables adicionales como tonalidad, duración y velocidad.

5.0.3. Música Computacional

La música computacional representa la intersección entre teoría musical, matemáticas y ciencias de la computación, permitiendo el análisis, representación y generación de música mediante algoritmos y sistemas formales.

Fundamentos de Teoría Musical Computacional

Boenn, Brain, De Vos et al. [18] definen la teoría musical computacional como la formalización de reglas musicales de manera que sus semánticas sean inteligibles para máquinas, permitiendo que computadoras razonen sobre y analicen estas reglas. Esta formalización constituye la base para sistemas de composición algorítmica que pueden generar música respetando principios teóricos musicales establecidos.

Los componentes fundamentales de la música computacional incluyen:

- **Representación formal de conceptos musicales:** Codificación de elementos como tonalidad, armonía, ritmo y estructura en formatos computacionalmente tratables

- **Algoritmos de análisis musical:** Métodos computacionales para extraer características musicales de representaciones simbólicas o de audio
- **Sistemas de razonamiento musical:** Implementaciones que pueden evaluar la validez o calidad de secuencias musicales según reglas teóricas
- **Generación algorítmica:** Síntesis de nuevas composiciones mediante procedimientos computacionales basados en teoría musical

Detección Automática de Tonalidad

La tonalidad constituye uno de los elementos estructurales más importantes de la música tonal occidental. La capacidad de detectar automáticamente la tonalidad de una pieza musical es fundamental para sistemas de análisis y generación musical.

Algoritmo de Krumhansl-Schmuckler Krumhansl y Kessler [19] desarrollaron un método basado en perfiles de clases de pitch para la detección de tonalidad. El algoritmo opera mediante los siguientes pasos:

1. **Extracción de perfil de pitch:** Se calcula la distribución de clases de pitch en la pieza musical, generando un vector de 12 dimensiones que representa la frecuencia de aparición de cada nota de la escala cromática.
2. **Correlación con perfiles teóricos:** Este vector se compara mediante correlación con 24 perfiles de referencia (12 tonalidades mayores y 12 menores) establecidos empíricamente.
3. **Selección de tonalidad:** La tonalidad con mayor correlación se identifica como la tonalidad de la pieza.

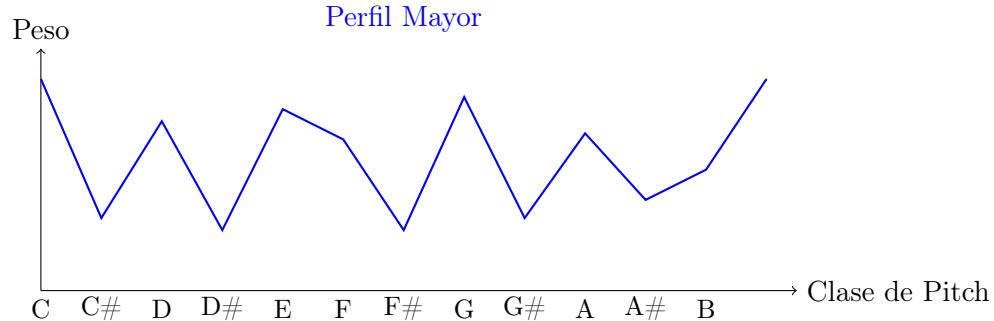


Figura 2: Representación esquemática del perfil de tonalidad mayor según Krumhansl-Schmuckler. Las alturas indican el peso de cada clase de pitch en la tonalidad.

Validación Empírica del Algoritmo Frankland y Cohen [20] realizaron una validación empírica del algoritmo de Krumhansl-Schmuckler, utilizándolo para cuantificar los efectos de

la tonalidad en tareas de comparación de pitch. Sus resultados demostraron que el algoritmo correlaciona significativamente con la percepción humana de tonalidad, especialmente en sujetos con entrenamiento musical. Este trabajo validó la efectividad del algoritmo para:

- **Determinar la tonalidad percibida:** El algoritmo predice con precisión la tonalidad que oyentes musicalmente entrenados perciben en secuencias cortas
- **Cuantificar efectos tonales:** Permite medir objetivamente cómo el contexto tonal afecta expectativas melódicas
- **Evaluar estabilidad tonal:** Identifica qué tan fuertemente una secuencia musical establece una tonalidad específica
- **Predecir expectativas:** El algoritmo puede usarse para predecir qué notas serán esperadas después de un contexto tonal dado

La validación empírica de Frankland y Cohen es particularmente relevante para este trabajo, ya que confirma que el algoritmo no solo detecta tonalidad algorítmicamente, sino que sus resultados corresponden a la percepción musical humana real. Esta correspondencia entre detección algorítmica y percepción humana es fundamental para justificar el uso del algoritmo en la evaluación de adherencia tonal de composiciones generadas.

Implementaciones Computacionales Temperley [21] extendieron este enfoque mediante métodos probabilísticos que consideran no solo la distribución de clases de pitch, sino también patrones de progresiones armónicas. La librería music21 [22] proporciona implementaciones de estos algoritmos que facilitan su aplicación práctica en proyectos de investigación.

Aplicaciones en Generación Musical El algoritmo de Krumhansl-Schmuckler tiene múltiples aplicaciones en sistemas de generación musical:

- **Análisis de dataset:** Permite caracterizar automáticamente la tonalidad de piezas existentes en datasets de entrenamiento
- **Evaluación de salida:** Facilita la verificación automática de que composiciones generadas respetan tonalidades especificadas
- **Variable de condicionamiento:** La tonalidad detectada puede usarse como parámetro adicional para condicionar la generación
- **Métrica de coherencia:** Puede medir la consistencia tonal de composiciones generadas, indicando calidad musical

Análisis de Estructuras Musicales

Más allá de la tonalidad, la música computacional permite el análisis de:

- **Estructura armónica:** Identificación de progresiones de acordes y funciones tonales
- **Estructura formal:** Detección de secciones (verso, coro, puente) y patrones de repetición
- **Características rítmicas:** Extracción de métricas, patrones rítmicos y micromotivos
- **Características melódicas:** Análisis de contornos, intervalos y motivos melódicos

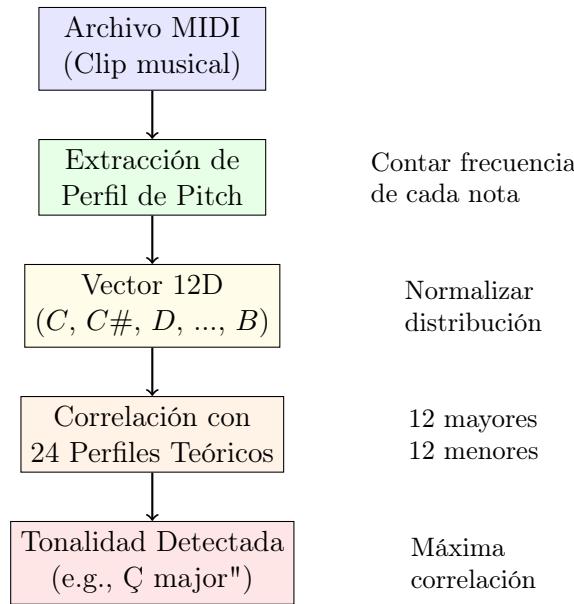


Figura 3: Flujo del algoritmo de Krumhansl-Schmuckler para detección automática de tonalidad. El proceso transforma un clip MIDI en un vector de distribución de pitch que se compara con perfiles teóricos de tonalidades.

Conexión con el Trabajo de Graduación

La música computacional fundamenta directamente el Objetivo Específico 1 del presente trabajo:

- **Extensión del dataset:** La aplicación del algoritmo de Krumhansl-Schmuckler a los clips del dataset EMOPIA permitirá incorporar información de tonalidad como variable de control adicional.
- **Variable de condicionamiento tonal:** La tonalidad detectada automáticamente servirá como parámetro adicional para condicionar la generación musical, complementando el condicionamiento emocional existente.
- **Coherencia musical:** La información de tonalidad facilitará la generación de composiciones con mayor coherencia armónica, ya que el modelo podrá aprender correlaciones entre tonalidad, emoción y patrones armónicos específicos.

- **Evaluación de adherencia tonal:** Los mismos algoritmos de detección de tonalidad podrán utilizarse para verificar que las composiciones generadas respetan la tonalidad especificada como variable de condicionamiento (Objetivos Específicos 4 y 5).

5.0.4. Fundamentos de Aprendizaje Profundo y Redes Neuronales

El aprendizaje profundo constituye el paradigma computacional fundamental sobre el cual se construyen los sistemas modernos de generación musical.

Arquitecturas de Redes Neuronales Profundas

Schmidhuber [23] proporciona una revisión histórica exhaustiva del aprendizaje profundo en redes neuronales, identificando que los aprendices profundos se distinguen de los superficiales por la profundidad de sus caminos de asignación de crédito, que son cadenas de enlaces causales entre acciones y efectos. Esta perspectiva histórica revela la evolución de técnicas que ahora permiten entrenar redes con millones de parámetros de manera eficiente.

Componentes Fundamentales de Redes Neuronales Las arquitecturas de aprendizaje profundo se componen de elementos básicos que operan en capas jerárquicas:

- **Neuronas artificiales:** Unidades computacionales que aplican funciones no lineales a combinaciones ponderadas de entradas
- **Capas ocultas:** Niveles intermedios de representación que extraen características de complejidad creciente
- **Funciones de activación:** Transformaciones no lineales (ReLU, sigmoid, tanh) que permiten modelar relaciones complejas
- **Propagación hacia adelante:** Proceso de cálculo desde entradas hasta salidas
- **Retropropagación:** Algoritmo para actualizar pesos mediante gradiente descendente

Arquitecturas Recurrentes y de Atención El desarrollo histórico del campo ha producido arquitecturas especializadas para datos secuenciales:

- **RNNs (Redes Neuronales Recurrentes):** Mantienen estado interno para procesar secuencias temporales
- **LSTMs (Long Short-Term Memory):** Abordan el problema del desvanecimiento de gradiente mediante células de memoria especializadas
- **GRUs (Gated Recurrent Units):** Versión simplificada de LSTMs con menos parámetros
- **Mecanismos de Atención:** Permiten que el modelo enfoque selectivamente partes relevantes de la entrada

Arquitectura Transformer

La arquitectura Transformer, introducida por Vaswani, Shazeer, Parmar et al. [24], revolucionó el procesamiento de secuencias al prescindir completamente de mecanismos recurrentes en favor de atención pura. Esta arquitectura constituye la base del sistema EMOPIA y, por extensión, del presente trabajo de graduación.

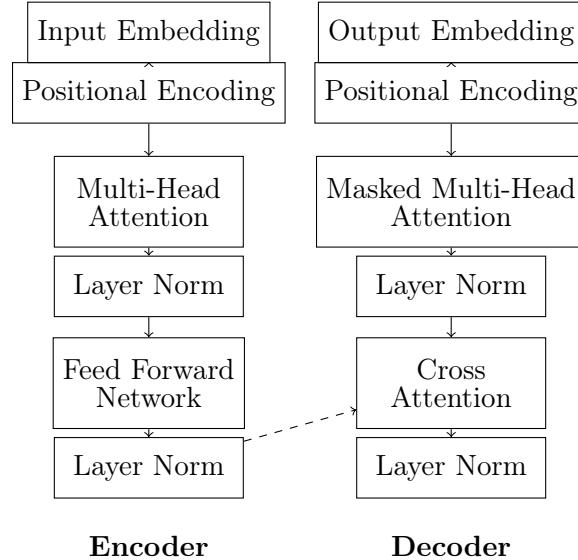


Figura 4: Arquitectura básica del Transformer mostrando las capas principales del encoder y decoder. Las conexiones residuales y normalizaciones de capa permiten entrenar modelos muy profundos.

Componentes Clave del Transformer Los elementos fundamentales incluyen:

- **Self-Attention Multi-Cabeza:** Permite que cada posición atienda a todas las demás posiciones simultáneamente, capturando dependencias de largo alcance
- **Positional Encoding:** Inyecta información sobre posiciones absolutas o relativas en la secuencia
- **Feed-Forward Networks:** Capas completamente conectadas aplicadas independientemente a cada posición
- **Conexiones Residuales:** Facilitan el flujo de gradientes en redes muy profundas
- **Layer Normalization:** Estabiliza el entrenamiento normalizando activaciones

Mecanismo de Atención El mecanismo de atención calcula pesos dinámicos para diferentes partes de la entrada:

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

Donde Q (queries), K (keys) y V (values) son proyecciones lineales de la entrada, y d_k es la dimensionalidad de las keys. Este mecanismo permite que el modelo aprenda qué elementos de la secuencia son relevantes para predecir cada token de salida.

Conexión con el Trabajo de Graduación

Los fundamentos de aprendizaje profundo y la arquitectura Transformer son directamente relevantes para múltiples objetivos específicos:

- **Objetivo Específico 2:** La comprensión detallada de la arquitectura Transformer es esencial para adaptar el modelo EMOPIA e incorporar la variable de tonalidad como parámetro de condicionamiento adicional. La modificación requerirá ajustes en las capas de embedding de entrada y potencialmente en los mecanismos de atención para procesar múltiples variables de control.
- **Objetivo Específico 3:** El conocimiento del mecanismo de atención multi-cabeza permite diseñar estrategias de inferencia multiparamétrica, donde diferentes cabezas de atención podrían especializarse en diferentes aspectos del condicionamiento (emoción, tonalidad, tempo).
- **Base arquitectónica:** El sistema EMOPIA utiliza una variante de Transformer adaptada para música, específicamente el Compound Word Transformer [3], que será la base arquitectónica de este trabajo.

5.0.5. Generación Musical mediante Aprendizaje Profundo

Representaciones Musicales para Aprendizaje Automático

La representación de información musical para su procesamiento por algoritmos de aprendizaje automático es un aspecto fundamental que determina la efectividad del sistema generativo.

Representaciones Simbólicas Ji, Luo y Yang [25] proporcionan una revisión exhaustiva de las representaciones musicales multinivel utilizadas en la generación musical profunda. Las representaciones principales incluyen:

- **Representación MIDI:** Formato estándar con información precisa de timing, velocidad y duración
- **Representación Piano Roll:** Matriz bidimensional donde el eje X representa tiempo y el eje Y representa pitch
- **Representación Event-based:** Secuencia de eventos musicales discretos que facilita el procesamiento secuencial

- **Representación ABC Notation:** Sistema de notación musical textual que permite codificar melodías y ritmos de manera compacta
- **MusicXML:** Formato estandarizado para intercambio de información musical que preserva elementos de notación tradicional
- **Representación por Tokens:** Tokenización de elementos musicales que permite aplicar técnicas de procesamiento de lenguaje natural

La música simbólica representa la información musical de manera discreta y estructurada, a diferencia del audio que es una señal continua. Esta representación es fundamental para este tipo de sistemas porque permite un control preciso sobre los elementos musicales específicos (notas, duraciones, dinámicas) y facilita la implementación de mecanismos de control emocional paramétrico. Además, el dataset EMOPIA incluye representaciones MIDI que podrían servir como base para el entrenamiento de modelos generativos.

Formatos de Audio Digital Complementariamente, los formatos de audio digital proporcionan representaciones continuas de la señal musical:

- **Waveform:** Representación temporal directa de la señal de audio
- **Espectrogramas:** Representación tiempo-frecuencia que revela el contenido espectral
- **Mel-espectrogramas:** Espectrogramas con escala perceptual que imita la audición humana
- **Constant-Q Transform (CQT):** Análisis frecuencial con resolución musical optimizada
- **Chromagrams:** Representación de clases de pitch que abstrae información tonal

Estos formatos de audio son relevantes para sistemas de este tipo porque el dataset EMOPIA incluye tanto representaciones MIDI como archivos de audio, permitiendo entrenar modelos que correlacionen características simbólicas con percepción auditiva real. Esta dualidad podría facilitar la validación perceptual de las composiciones generadas.

Embeddings Musicales Lisena, Meroño-Peña y Troncy [26] desarrollaron MIDI2vec, un enfoque para aprender embeddings de archivos MIDI que permite predicciones confiables de metadatos de música simbólica. Esta técnica representa archivos MIDI como vectores, facilitando su procesamiento por algoritmos de aprendizaje profundo.

Arquitecturas para Generación Musical

Modelos Transformer Los Transformers han revolucionado la generación musical mediante mecanismos de atención. Huang et al. [27] introdujeron el Music Transformer, que adapta la arquitectura Transformer para música mediante:

- **Relative attention:** Adaptación específica para capturar relaciones musicales
- **Memoria extendida:** Manejo de secuencias musicales largas
- **Generación con estructura:** Capacidad para mantener coherencia musical a largo plazo

Modelos Especializados en Estilo Hadjeres, Pachet y Nielsen [28] desarrollaron Deep-Bach, un modelo dirigible para la generación de corales de Bach, demostrando que es posible entrenar modelos especializados en estilos musicales específicos que mantienen las características estilísticas del compositor objetivo.

Modelos Adversariales Dong et al. [29] presentaron MuseGAN, que utiliza redes generativas adversariales para la generación de música simbólica multi-track y acompañamiento, mostrando la viabilidad de los enfoques adversariales en el dominio musical.

Generación Condicional de Música

La generación condicional representa un paradigma fundamental en la síntesis musical automática, donde el sistema genera contenido musical basado en parámetros específicos de entrada que controlan características deseadas del resultado.

Condicionamiento por Atributos Emocionales Ferreira y Whitehead [30] desarrollaron técnicas de generación musical condicional donde el sistema puede ser dirigido hacia emociones específicas mediante vectores de condicionamiento. Los enfoques principales incluyen:

- **Condicionamiento Directo:** Incorporación de etiquetas emocionales como entrada adicional al modelo
- **Condicionamiento por Embeddings:** Uso de representaciones vectoriales de emociones aprendidas automáticamente
- **Condicionamiento Jerárquico:** Control multinivel desde estructura global hasta detalles locales
- **Condicionamiento Adversarial:** Uso de discriminadores especializados en evaluar coherencia emocional

La generación condicional representa el mecanismo central en sistemas de este tipo, ya que permitiría generar composiciones de piano dirigidas hacia emociones específicas del modelo valencia-activación. Esta capacidad de control paramétrico distinguiría tales sistemas de generadores musicales genéricos y los orientaría hacia aplicaciones prácticas específicas.

Condicionamiento por Características Musicales Dai, Jin, Gomes et al. [31] presentaron enfoques para el control fino de características musicales específicas:

- **Control Temporal:** Manipulación de tempo, compás y estructura rítmica
- **Control Tonal:** Especificación de tonalidad, modo y progresiones armónicas
- **Control Dinámico:** Ajuste de intensidad y variaciones de velocidad MIDI
- **Control Estructural:** Definición de forma musical y secciones
- **Control Estilístico:** Orientación hacia géneros o estilos compositivos específicos

Estos mecanismos de control específico podrían complementar el condicionamiento emocional en sistemas de generación musical, permitiendo ajustes finos de las características musicales que contribuyen a la expresión emocional deseada.

Arquitecturas para Generación Condicional Las arquitecturas especializadas para generación condicional incluyen:

- **Conditional VAEs (CVAEs):** Variational Autoencoders que incorporan información de condicionamiento en el espacio latente
- **Conditional GANs (CGANs):** Redes adversariales donde tanto el generador como el discriminador reciben información de condicionamiento
- **Transformer Condicionales:** Modelos de atención que integran tokens de condicionamiento en la secuencia de entrada
- **Diffusion Models Condicionales:** Modelos de difusión que incorporan guidance hacia características específicas

La selección de la arquitectura apropiada para un sistema de este tipo requeriría la evaluación comparativa de cada enfoque, considerando factores como la coherencia emocional y la calidad musical de las composiciones generadas. No obstante, dado que el presente trabajo se fundamenta sobre el sistema EMOPIA, se utilizará la arquitectura Transformer como base, aprovechando tanto el modelo preentrenado como las implementaciones existentes para la generación condicional de música de piano.

Métodos de Inferencia en Generación Condicional

Una vez entrenado un modelo generativo condicional, existen múltiples estrategias para realizar inferencia (generación) que difieren en el grado de control y determinismo aplicado.

Muestreo Estocástico (Stochastic Sampling) El muestreo estocástico permite variabilidad natural en la generación mediante la introducción de aleatoriedad controlada:

- **Temperature Sampling:** Controla la entropía de la distribución de probabilidad mediante un parámetro de temperatura τ :

$$p_i = \frac{\exp(z_i/\tau)}{\sum_j \exp(z_j/\tau)}$$

donde z_i son los logits del modelo. Temperaturas altas ($\tau > 1$) incrementan diversidad, mientras que temperaturas bajas ($\tau < 1$) hacen el muestreo más conservador.

- **Nucleus Sampling (Top-p):** Muestra del subconjunto más pequeño de tokens cuya probabilidad acumulada supera un umbral p [32]:

$$V^{(p)} = \min \left\{ V' : \sum_{x \in V'} P(x) \geq p \right\}$$

Este método evita muestrear tokens de probabilidad muy baja, mejorando coherencia sin eliminar diversidad.

- **Top-k Sampling:** Restringe el muestreo a los k tokens más probables, re-normalizando sus probabilidades.

En el contexto de generación musical condicional, el muestreo estocástico permite que el modelo explore variaciones naturales mientras respeta las condiciones especificadas (emoción, tonalidad, etc.). Esto es deseable cuando se busca diversidad creativa dentro de restricciones emocionales o tonales.

Muestreo Determinístico (Deterministic Sampling) El muestreo determinístico elimina la aleatoriedad seleccionando siempre el token de mayor probabilidad:

- **Argmax Decoding:** Selecciona el token con máxima probabilidad en cada paso:

$$x_t = \arg \max_x P(x | x_{<t}, \mathbf{c})$$

donde \mathbf{c} representa las variables de condicionamiento.

- **Beam Search:** Mantiene los k caminos de mayor probabilidad acumulada, explorando múltiples secuencias candidatas simultáneamente.

El muestreo determinístico es útil cuando se requiere máxima adherencia a las distribuciones aprendidas del modelo, minimizando variabilidad. En generación musical, esto puede producir composiciones más predecibles pero potencialmente más coherentes con las variables de condicionamiento especificadas.

Generación con Tokens Forzados (Forced Token Generation) Un tercer paradigma implica fijar directamente ciertos tokens durante la generación:

- **Hard Constraints:** Tokens específicos (e.g., pitch, duración, velocidad) se establecen a valores fijos independientemente de las predicciones del modelo
- **Propósito:** Permite control directo sobre características musicales específicas
- **Trade-off:** Mayor control sobre tokens individuales a costa de potencial incoherencia en estructura de alto nivel (tonalidad global, coherencia emocional)

Este enfoque es conceptualmente diferente del condicionamiento: mientras que el condicionamiento influye en las distribuciones de probabilidad del modelo, el forzamiento de tokens anula las predicciones del modelo por completo para tokens específicos.

Característica	Estocástico	Determinístico	Forzado
Variabilidad	Alta (controlada por τ, p)	Mínima (siempre mismo output)	Ninguna para tokens forzados
Adherencia a condiciones	Suave (probabilística)	Estricta (máxima probabilidad)	Absoluta (por diseño)
Coherencia estructural	Alta	Alta	Potencialmente comprometida
Diversidad creativa	Alta	Baja	Media (forzada en aspectos específicos)
Aplicación típica	Exploración creativa condicionada	Reproducción consistente	Control detallado de características

Cuadro 1: Comparación de paradigmas de inferencia en generación musical condicional. Cada enfoque presenta trade-offs entre control, diversidad y coherencia.

Comparación de Paradigmas de Inferencia

Evaluación de Métodos de Inferencia La evaluación comparativa de métodos de inferencia requiere métricas que capturen diferentes aspectos:

- **Adherencia al condicionamiento:** ¿Qué tan bien respeta el output las variables especificadas (emoción, tonalidad)?
- **Coherencia musical:** ¿Es la composición generada musicalmente coherente según teoría musical?
- **Diversidad:** ¿Cuánta variación existe entre múltiples generaciones con las mismas condiciones?
- **Calidad perceptual:** ¿Son las composiciones agradables y expresivas según evaluadores humanos?

Un aspecto crítico es que la adherencia tautológica (como en tokens forzados, donde adherencia es 100 % por diseño) no es informativa. Lo relevante es evaluar si el forzamiento de tokens de bajo nivel (pitch, duración) compromete propiedades estructurales de alto nivel (emoción global, coherencia tonal).

Conexión con el Trabajo de Graduación

Los métodos de inferencia descritos fundamentan directamente el Objetivo Específico 3:

- **Implementación de métodos de inferencia multiparamétrica:** El presente trabajo implementará y comparará diferentes estrategias de inferencia (estocástica con condicionamiento, determinística con condicionamiento, y con tokens forzados) para evaluar sus efectos en la adherencia a múltiples variables simultáneas.
- **Framework de evaluación comparativa:** El Objetivo Específico 4 desarrollará un pipeline que cuantifique adherencia tanto a variables directamente controladas (tokens) como a propiedades emergentes (emoción global, tonalidad).
- **Análisis de trade-offs:** El Objetivo Específico 5 comparará cuantitativamente cómo diferentes métodos de inferencia equilibran adherencia a condiciones versus coherencia estructural.
- **Control explícito multiparamétrico:** La combinación de condicionamiento emocional/tonal con diferentes estrategias de muestreo permite control explícito sobre múltiples variables simultáneas, respondiendo al objetivo general del trabajo.

Técnicas de Evaluación para Música Generada

Metodologías de Evaluación Integral Xiong et al. [33] proporcionan una revisión exhaustiva de las metodologías de evaluación para música generada por IA, identificando los principales desafíos en la evaluación objetiva y subjetiva de sistemas generativos musicales.

Métricas Objetivas Yang y Lerch [34] propusieron un conjunto de métricas objetivas simples pero musicalmente relevantes para la evaluación de modelos generativos musicales, incluyendo:

- **Métricas basadas en Teoría Musical:** Coherencia tonal, estructura formal, complejidad rítmica
- **Métricas Estadísticas:** Perplexity, tasa de repetición de notas, entropía de clases de pitch
- **Métricas de Similitud:** Distancia de edición, similitud coseno en espacios de características

Evaluación de Adherencia al Condicionamiento Un aspecto crítico en sistemas de generación condicional es verificar que las composiciones generadas respetan las variables de condicionamiento especificadas:

- **Clasificadores de verificación:** Entrenar modelos separados que predicen variables de condicionamiento a partir de música generada
- **Análisis automático:** Aplicar algoritmos de análisis musical (e.g., detección de tonalidad) a composiciones generadas
- **Comparación estadística:** Contrastar distribuciones de características musicales entre clases de condicionamiento
- **Métricas de control:** Cuantificar qué tan bien el sistema responde a cambios en variables de condicionamiento

Conexión con el Trabajo de Graduación

Las técnicas de generación musical mediante aprendizaje profundo fundamentan múltiples aspectos de este trabajo:

- **Objetivo Específico 2:** Las representaciones musicales (especialmente MIDI/tokens) y arquitecturas Transformer proporcionan la base técnica para adaptar el modelo EMOPIA e incorporar la variable de tonalidad.
- **Objetivo Específico 3:** Las técnicas de generación condicional (condicionamiento directo, embeddings, control jerárquico) informan el diseño de métodos de inferencia multiparamétrica.
- **Objetivos Específicos 4 y 5:** Las metodologías de evaluación y métricas objetivas proporcionan el marco conceptual para desarrollar el pipeline de evaluación de adherencia y realizar comparaciones cuantitativas del sistema extendido versus el original.
- **Representación simbólica:** El uso de música simbólica (MIDI) facilita tanto el análisis automático de características (tonalidad, tempo) como la evaluación precisa de adherencia a variables de condicionamiento.

5.0.6. El Dataset EMOPIA

Estructura y Características

El dataset EMOPIA [1] representa un recurso fundamental para la investigación en generación musical emocional, específicamente diseñado para música de piano pop.

Composición del Dataset

- **Contenido Musical:** 1,087 clips musicales de piano pop
- **Duración:** 30 segundos por clip
- **Formato:** Audio (.wav) y representación MIDI
- **Género:** Piano pop contemporáneo

Diversidad Musical El dataset incluye múltiples artistas y compositores, proporcionando variedad de estilos dentro del piano pop, diferentes estructuras musicales (intro, verso, coro, puente), y un rango temporal de música contemporánea (2000-2020).

Metodología de Anotación Emocional

Modelo Teórico Adoptado EMOPIA utiliza el modelo bidimensional de valencia-activación de Russell [8], creando cuatro categorías emocionales:

1. **Q1 - Feliz/Enérgico** (High Valence, High Arousal): Euforia, alegría intensa, excitación
2. **Q2 - Feliz/Calmado** (High Valence, Low Arousal): Serenidad, contentamiento, paz
3. **Q3 - Triste/Enérgico** (Low Valence, High Arousal): Angustia, tensión, agitación melancólica
4. **Q4 - Triste/Calmado** (Low Valence, Low Arousal): Melancolía, tristeza suave, nostalgia

Proceso de Anotación El proceso incluyó la selección de músicos profesionales y semi-profesionales como anotadores, un protocolo de escucha de clips de 30 segundos con evaluación en escalas continuas de valencia y activación, y validación mediante múltiples anotadores por clip.

EMOPIA representa un recurso empírico fundamental para el desarrollo de sistemas de generación musical emocional, proporcionando tanto datos de entrenamiento como un marco de evaluación emocional. La metodología rigurosa de anotación emocional del dataset garantiza que modelos entrenados con este corpus puedan aprender correlaciones válidas entre características musicales y percepciones emocionales, mientras que la diversidad musical del corpus favorece la capacidad de generalización de sistemas generativos.

Análisis Preliminar de Patrones Emocionales-Musicales

Distribución de Categorías Emocionales La distribución de clips por cuadrante emocional en EMOPIA se detalla en el Cuadro 2:

Cuadrante	Número de clips	Duración promedio (seg)	Duración promedio (tokens)
Q1	250	31.9	1,065
Q2	265	35.6	1,368
Q3	253	40.6	771
Q4	310	38.2	729
Total	1,078	36.7	983

Cuadro 2: Distribución de clips por cuadrante emocional en el dataset EMOPIA. Q1: Feliz/Enérgico, Q2: Feliz/Calmado, Q3: Triste/Enérgico, Q4: Triste/Calmado. La duración se presenta tanto en segundos como en número de tokens de la representación simbólica.

Patrones de Anotación Berardinis et al. [35] propusieron el Patrón de Anotación Musical, que unifica diferentes sistemas de anotación y representa objetos musicales como acordes, patrones y estructuras, proporcionando un marco teórico para el manejo sistemático de anotaciones musicales complejas.

Conexión con el Trabajo de Graduación

El dataset EMOPIA constituye la base empírica fundamental de este trabajo de graduación y se relaciona directamente con los objetivos específicos:

- **Objetivo Específico 1:** El dataset EMOPIA será extendido mediante la incorporación de información de tonalidad detectada automáticamente usando el algoritmo de Krumhansl-Schmuckler. Esta extensión enriquece el recurso existente agregando una variable de control musical fundamentada en teoría.
- **Base de entrenamiento:** El dataset extendido servirá para entrenar el modelo adaptado del Objetivo Específico 2, permitiendo que el sistema aprenda correlaciones entre emoción, tonalidad y patrones musicales.
- **Base de evaluación:** Las anotaciones emocionales del dataset proporcionan ground truth para evaluar adherencia emocional en el Objetivo Específico 4. Las tonalidades detectadas servirán similarmente para evaluar adherencia tonal.
- **Distribución balanceada:** La distribución relativamente uniforme de clips entre los cuatro cuadrantes emocionales (250-310 clips por cuadrante) facilita el entrenamiento balanceado y la evaluación comparativa del Objetivo Específico 5.
- **Formato dual:** La disponibilidad de representaciones MIDI y audio permite tanto el entrenamiento de modelos simbólicos como la validación perceptual futura de las composiciones generadas.

5.0.7. Fundamentos Neurocientíficos y Psicológicos

Correlatos Neurales de la Emoción Musical

Koelsch [36] identificaron los correlatos cerebrales de las emociones evocadas por la música, demostrando que la amígdala superficial tiene un papel específico en el procesamiento

emocional musical y que estos hallazgos tienen implicaciones para terapias basadas en música.

Salimpoor et al. [37] descubrieron que existe liberación de dopamina anatómicamente distinta durante la anticipación y experiencia de emociones pico en respuesta a la música, proporcionando evidencia neurológica del valor emocional intrínseco de la música.

Aplicaciones Contemporáneas

Sistemas de Generación Emocional Kang, Lu, Yu et al. [38] desarrollaron EmoGen, un sistema que elimina el sesgo subjetivo en la generación de música emocional, mientras que Zheng, Yang, Zhang et al. [39] crearon EmotionBox, un sistema de generación musical emocional basado en elementos musicales y psicología musical.

Aplicaciones Terapéuticas Modran, Chamunorwa, Ursutiu et al. [40] demostraron el uso de aprendizaje profundo para reconocer efectos terapéuticos de la música basados en emociones, validando el potencial de estos sistemas en aplicaciones de salud mental y bienestar.

Conexión con el Trabajo de Graduación

Los fundamentos neurocientíficos y psicológicos validan la relevancia del presente trabajo:

- **Validación científica:** La evidencia neurológica sobre el procesamiento emocional musical sustenta la pertinencia del modelo circumplejo de Russell utilizado en EMO-PIA y en este trabajo.
- **Justificación de aplicaciones:** Los hallazgos sobre efectos terapéuticos de la música justifican el desarrollo de sistemas con control emocional preciso (Objetivos Específicos 3, 4, 5).
- **Orientación de variables:** La comprensión de cómo elementos musicales específicos (tempo, modo, timbre) afectan respuestas neuronales informa la selección de variables de control en el sistema.

5.0.8. Aplicaciones de la Generación Musical Emocional

Musicoterapia y Aplicaciones Terapéuticas

La generación automática de música emocional presenta aplicaciones significativas en el campo de la musicoterapia y el bienestar mental.

Fundamentos Terapéuticos Ramaswamy, Philip, Priya et al. [41] realizaron una revisión narrativa sobre el uso terapéutico de la música en trastornos neurológicos, demostrando que la música puede ser efectiva en intervenciones terapéuticas cuando está específicamente diseñada para objetivos emocionales. Los principios terapéuticos incluyen:

- **Regulación Emocional:** Uso de música para modular estados afectivos específicos
- **Estimulación Cognitiva:** Activación de redes neuronales mediante patrones musicales específicos
- **Relajación Dirigida:** Generación de composiciones para reducir ansiedad y estrés
- **Activación Motivacional:** Música energizante para combatir estados depresivos

Esta aplicación es directamente relevante para el desarrollo de sistemas de generación musical emocional, ya que la capacidad de generar música específicamente dirigida hacia emociones del modelo valencia-activación podría permitir crear herramientas terapéuticas personalizadas que se adapten a las necesidades emocionales específicas de cada paciente.

Interfaces de Biofeedback Musical Eaton, Barthet y Sandler [42] desarrollaron sistemas que combinan generación musical con medición de señales fisiológicas en tiempo real:

- **Monitoreo de Estados:** Sensores de ritmo cardíaco, conductancia de la piel y actividad cerebral
- **Adaptación Automática:** Modificación de parámetros musicales basada en respuestas fisiológicas
- **Bucles de Retroalimentación:** Sistemas que aprenden preferencias individuales a lo largo del tiempo

Producción Musical y Creatividad Asistida

Herramientas para Compositores Los sistemas de generación musical emocional pueden servir como herramientas de asistencia creativa para compositores y productores:

- **Exploración de Ideas:** Generación de motivos y progresiones para inspiración compositiva
- **Arreglos Emocionales:** Adaptación automática de melodías existentes hacia emociones específicas
- **Acompañamientos Inteligentes:** Generación de texturas armónicas complementarias
- **Variaciones Estilísticas:** Exploración de diferentes aproximaciones emocionales a una misma idea musical

Un sistema de este tipo podría proporcionar a compositores una herramienta que va más allá de la generación musical genérica, ofreciendo control específico sobre la dirección emocional de las composiciones generadas.

Aplicaciones en Medios Interactivos Nierhaus [43] identificó aplicaciones de la composición algorítmica en videojuegos, instalaciones artísticas y experiencias inmersivas:

- **Música Adaptativa para Videojuegos:** Generación de bandas sonoras que responden al estado emocional de la narrativa
- **Instalaciones Interactivas:** Espacios donde la música se adapta a la presencia y comportamiento de los visitantes
- **Realidad Virtual Emocional:** Ambientes inmersivos donde la música refuerza la experiencia emocional deseada

Educación y Democratización Musical

Herramientas Educativas Los sistemas de generación musical emocional pueden facilitar la educación musical:

- **Comprendión de Elementos Emocionales:** Demostración práctica de cómo diferentes elementos musicales afectan la percepción emocional
- **Composición Asistida:** Plataformas que permiten a estudiantes experimentar con composición sin conocimientos técnicos avanzados
- **Análisis Musical Interactivo:** Sistemas que explicitan las correlaciones entre estructura musical y respuesta emocional

Esta aplicación educativa representa una contribución social importante del presente trabajo, democratizando el acceso a herramientas de composición musical y facilitando la comprensión de los principios de expresión emocional musical.

Conexión con el Trabajo de Graduación

Las aplicaciones de la generación musical emocional contextualizan la relevancia práctica del presente trabajo:

- **Justificación del enfoque multiparamétrico:** Las aplicaciones terapéuticas y creativas requieren control preciso sobre múltiples características musicales simultáneamente, validando la necesidad de los Objetivos Específicos 2 y 3.
- **Criterios de evaluación:** Las aplicaciones prácticas establecen requisitos de calidad que informan el desarrollo del pipeline de evaluación del Objetivo Específico 4.

- **Impacto potencial:** La capacidad de generar música con control emocional específico tiene aplicaciones verificables en múltiples dominios, desde terapia hasta producción musical.

5.0.9. Sistemas Alternativos de Generación Musical y Justificación de Decisiones Tecnológicas

La elección del sistema EMOPIA como base para este trabajo de graduación se fundamenta en un análisis comparativo de sistemas alternativos de generación musical disponibles.

Panorama de Sistemas de Generación Musical

Sistemas Comerciales de Generación Musical El panorama actual incluye múltiples plataformas comerciales de generación musical mediante IA:

- **Suno [4]:** Plataforma de generación de música completa mediante IA generativa que permite crear canciones con letra y arreglos instrumentales a partir de descripciones textuales. Enfoque: música con letra, múltiples géneros.
- **ElevenLabs Music [5]:** Servicio de generación text-to-music que crea composiciones musicales de alta calidad basadas en prompts textuales. Enfoque: generación de audio directo, múltiples estilos.
- **Google Magenta:** Proyecto de investigación que explora el rol del machine learning en la creación de arte y música, incluyendo herramientas como Music Transformer y Performance RNN [44].

Sistemas Académicos Especializados La investigación académica ha producido sistemas especializados con diferentes enfoques:

- **Jukebox [45]:** Modelo generativo de OpenAI que genera música directamente como audio, incluyendo canto. Ventajas: alta calidad de audio. Limitaciones: control limitado sobre parámetros específicos, requerimientos computacionales elevados.
- **Music Transformer [27]:** Arquitectura Transformer adaptada para música simbólica con atención relativa. Ventajas: maneja dependencias de largo alcance. Limitaciones: no especializado en control emocional.
- **MuseGAN [29]:** Sistema basado en GANs para generación multi-track. Ventajas: genera múltiples instrumentos simultáneamente. Limitaciones: menos control sobre características específicas.
- **DeepBach [28]:** Modelo dirigible para corales estilo Bach. Ventajas: alta fidelidad estilística. Limitaciones: limitado a un estilo específico.

- **EmotionBox [39]:** Sistema de generación emocional basado en elementos musicales y psicología musical. Ventajas: enfoque teórico en elementos emocionales. Limitaciones: alcance limitado de control multiparamétrico.
- **EmoGen [38]:** Sistema que elimina sesgos subjetivos en generación emocional. Ventajas: menor sesgo en anotaciones. Limitaciones: enfoque principalmente en reconocimiento más que generación condicional compleja.

Sistemas de Difusión Recientes Ning, Chen, Jiang et al. [6] presentaron DiffRhythm, un sistema de difusión latente para generación de canciones completas de manera eficiente. Aunque prometedor, los modelos de difusión para música aún están en desarrollo temprano comparados con enfoques basados en Transformers.

Evaluación Comparativa de Alternativas

El Cuadro 3 presenta una evaluación comparativa de sistemas según criterios relevantes para los objetivos de este trabajo:

Sistema	Control Emocional	Piano Específico	Dataset Emocional	Código Abierto	Emoción Explícita
EMOPIA	✓	✓	✓	✓	✓
Suno	✗	✗	✗	✗	✗
ElevenLabs	✗	✗	✗	✗	✗
Jukebox	✗	✗	✗	✓	✗
Music Transformer	✗	✗	✗	✓	✗
MuseGAN	✗	✗	✗	✓	✗
DeepBach	✗	✗	✗	✓	✗
EmotionBox	✓	✗	Limitado	Parcial	✓
EmoGen	✓	✗	✓	✗	✓

Cuadro 3: Evaluación comparativa de sistemas de generación musical según criterios relevantes para este trabajo. EMOPIA destaca por ser el único sistema que combina todos los requisitos.

Justificación de la Elección de EMOPIA

La selección del sistema EMOPIA como base para este trabajo de graduación se fundamenta en múltiples razones que alinean con los objetivos específicos del proyecto:

Razones Técnicas

1. **Control emocional explícito:** EMOPIA es uno de los pocos sistemas con control emocional fundamentado en el modelo circunplejo de Russell, directamente relevante para los Objetivos Específicos 3, 4 y 5.
2. **Dataset anotado emocionalmente:** Incluye 1,087 clips con anotaciones emocionales rigurosas, esencial para el Objetivo Específico 1 de extensión del dataset.

3. **Especialización en piano:** Enfoque específico en piano pop, permitiendo especialización profunda en un instrumento expresivo.
4. **Arquitectura Transformer:** Base arquitectónica moderna que facilita modificaciones para incorporar variables adicionales (Objetivo Específico 2).
5. **Código abierto:** Disponibilidad de implementación completa que permite experimentación y extensión.
6. **Representación simbólica:** Uso de MIDI permite control preciso sobre elementos musicales específicos.

Razones Metodológicas

- **Reproducibilidad:** Sistema documentado con resultados publicados que pueden ser replicados
- **Evaluabilidad:** Métricas establecidas que facilitan comparación con el sistema extendido
- **Extensibilidad:** Arquitectura modular que permite incorporar nuevas variables de control
- **Base científica:** Fundamentado en investigación académica rigurosa validada por pares

Limitaciones de Alternativas Los sistemas comerciales (Suno, ElevenLabs) carecen de acceso a código y modelos, imposibilitando investigación académica. Los sistemas académicos alternativos carecen de enfoque emocional explícito o no se especializan en piano. EmotionBox y EmoGen, aunque orientados a emoción, no ofrecen la combinación de dataset especializado, código abierto y arquitectura modificable que proporciona EMOPIA.

Conexión con el Trabajo de Graduación

Esta comparación justifica decisiones tecnológicas clave:

- **Elección de sistema base:** EMOPIA cumple todos los requisitos para abordar los cinco objetivos específicos del trabajo
- **Enfoque de extensión:** Construir sobre EMOPIA permite comparación directa con sistema establecido (Objetivo Específico 5)
- **Viabilidad técnica:** Disponibilidad de código y dataset hace factible la implementación en el tiempo disponible

5.0.10. Limitaciones de Investigación Existente y Gaps Identificados

A pesar de los avances significativos en generación musical mediante IA, existen limitaciones fundamentales en el estado del arte que este trabajo busca abordar.

Limitaciones en Control de Variables Musicales

Control Uniparamétrico Predominante La mayoría de sistemas existentes permiten control sobre una única variable de alto nivel:

- **Limitación identificada:** Los sistemas como EMOPIA original solo permiten condicionamiento emocional, sin control sobre características musicales fundamentales como tonalidad.
- **Impacto:** Compositores y aplicaciones terapéuticas requieren control simultáneo sobre múltiples parámetros para lograr resultados específicos.
- **Gap de investigación:** Falta de métodos sistemáticos para incorporar múltiples variables de control de teoría musical en sistemas de generación emocional.

Desconexión entre Teoría Musical y Generación

- **Limitación:** Muchos sistemas generan música sin incorporar conocimiento explícito de teoría musical computacional.
- **Consecuencia:** Composiciones que pueden violar principios teóricos fundamentales como coherencia tonal.
- **Gap:** Escasa integración entre algoritmos de análisis musical (como Krumhansl-Schmuckler) y sistemas generativos basados en aprendizaje profundo.

Limitaciones en Evaluación de Adherencia

Métricas de Evaluación Incompletas Xiong et al. [33] identifican que la evaluación de música generada por IA enfrenta desafíos significativos:

- **Subjetividad:** Dificultad para cuantificar objetivamente calidad musical y expresión emocional
- **Métricas limitadas:** Falta de métricas estándar para evaluar adherencia a múltiples variables simultáneas
- **Gap metodológico:** No existen pipelines establecidos para cuantificar adherencia multiparamétrica en generación condicional

Sesgos en Datasets Emocionales

- **Sesgos culturales:** Las anotaciones emocionales reflejan perspectivas culturales específicas
- **Sesgos de género musical:** Datasets como EMOPIA se enfocan en un género específico (piano pop)
- **Limitación temporal:** Clips de duración limitada (30 segundos) pueden no capturar arcos emocionales completos

Limitaciones Arquitectónicas

Escalabilidad del Condicionamiento

- **Problema:** Agregar variables de condicionamiento puede causar:
 - Explosión combinatoria del espacio de parámetros
 - Degrado de calidad cuando múltiples condiciones entran en conflicto
 - Aumento de complejidad computacional
- **Gap técnico:** Falta de arquitecturas diseñadas específicamente para manejar condicionamiento multiparamétrico sin degradación

Interpretabilidad Limitada

- **Caja negra:** Las redes neuronales profundas ofrecen poca interpretabilidad sobre cómo incorporan condicionamiento
- **Dificultad de depuración:** Cuando el sistema no respeta una variable de condicionamiento, es difícil identificar la causa

Gaps que Este Trabajo Aborda

Este trabajo de graduación aporta conocimiento nuevo en las siguientes áreas:

1. **Integración de teoría musical:** Incorpora detección algorítmica de tonalidad (Krumhansl-Schmuckler) como variable de control en un sistema de generación emocional (Objetivos 1 y 2).
2. **Condicionamiento multiparamétrico:** Desarrolla métodos de inferencia que permiten control simultáneo sobre emoción, tonalidad, duración y velocidad (Objetivo 3).
3. **Evaluación de adherencia:** Crea un pipeline sistemático para cuantificar adherencia a variables múltiples (Objetivo 4).

4. **Comparación empírica:** Proporciona evaluación cuantitativa del impacto de condicionamiento multiparamétrico (Objetivo 5).

Limitaciones Reconocidas del Presente Trabajo

Es importante reconocer las limitaciones inherentes a este trabajo:

- **Alcance de instrumento:** Limitado a piano; generalización a otros instrumentos requiere investigación adicional
- **Alcance de género:** Enfocado en piano pop; otros géneros musicales pueden requerir diferentes enfoques
- **Variables de control:** Aunque se amplían las variables, aún quedan características musicales relevantes sin abordar (e.g., estructura formal, dinámicas expresivas)
- **Evaluación:** Evaluación principalmente objetiva; validación perceptual completa con oyentes humanos queda para trabajo futuro

5.0.11. Síntesis del Marco Teórico e Integración Conceptual

Esta sección integra los conceptos presentados en un marco unificado que sustenta el presente trabajo de graduación.

Modelo Conceptual Integrado

El presente trabajo se fundamenta en la integración de cuatro dominios teóricos principales:

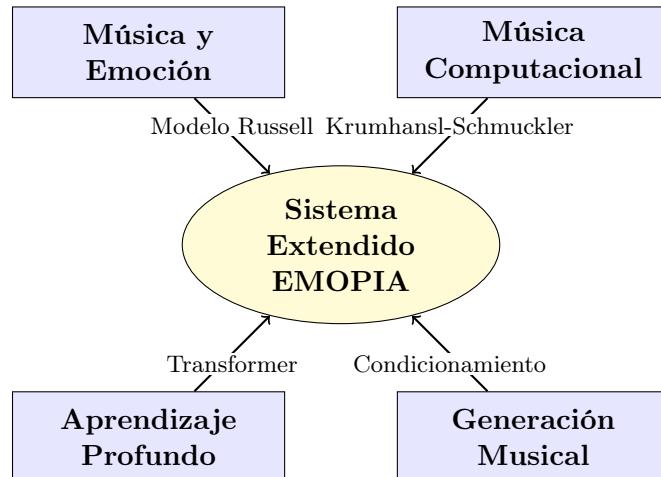


Figura 5: Modelo conceptual integrado del marco teórico. El sistema extendido EMOPIA integra conceptos de cuatro dominios teóricos principales.

Dominios Teóricos y sus Contribuciones

1. Música y Emoción:

- *Contribución:* Modelo circumplejo de Russell para representación emocional bidimensional
- *Aplicación:* Variable de condicionamiento emocional (valencia-activación)
- *Validación:* Correlaciones establecidas entre elementos musicales y respuestas emocionales

2. Música Computacional:

- *Contribución:* Algoritmo de Krumhansl-Schmuckler para detección de tonalidad
- *Aplicación:* Variable de condicionamiento tonal
- *Validación:* Fundamentos teóricos de teoría musical occidental

3. Aprendizaje Profundo:

- *Contribución:* Arquitectura Transformer con mecanismos de atención
- *Aplicación:* Base arquitectónica del modelo generativo
- *Validación:* Capacidad demostrada para modelar secuencias complejas

4. Generación Musical:

- *Contribución:* Técnicas de condicionamiento multiparamétrico
- *Aplicación:* Métodos de inferencia con múltiples variables
- *Validación:* Representaciones simbólicas (MIDI) para control preciso

Mapeo entre Marco Teórico y Objetivos Específicos

El Cuadro 4 presenta el mapeo explícito entre secciones del marco teórico y objetivos específicos del trabajo:

Contribución Teórica del Trabajo

El presente trabajo aporta conocimiento en la intersección de múltiples dominios:

- **Contribución primaria:** Metodología para integrar variables de teoría musical computacional (tonalidad) con condicionamiento emocional en sistemas generativos basados en Transformers.
- **Contribución metodológica:** Pipeline de evaluación para cuantificar adherencia multiparamétrica en generación musical condicional.
- **Contribución empírica:** Extensión del dataset EMOPIA con información de tonalidad, creando un recurso más rico para investigación futura.
- **Contribución práctica:** Sistema de generación musical con control explícito sobre múltiples variables relevantes para aplicaciones terapéuticas y creativas.

Sección del Marco Teórico	Objetivos Específicos Sustentados
Música y Emoción	Obj. 3 (condicionamiento emocional), Obj. 4 y 5 (evaluación adherencia emocional)
Música Computacional (incluye Krumhansl-Schmuckler)	Obj. 1 (detección y extensión de tonalidad), Obj. 4 y 5 (evaluación adherencia tonal con algoritmo validado)
Fundamentos de Aprendizaje Profundo	Obj. 2 (adaptación arquitectónica Transformer para múltiples variables)
Generación Musical (representaciones y arquitecturas)	Obj. 2 (incorporación de variables), Obj. 3 (base para métodos de inferencia)
Métodos de Inferencia Condicional	Obj. 3 (implementación de inferencia multiparamétrica: estocástica, determinística, forzada), Obj. 5 (comparación de paradigmas)
Dataset EMOPIA	Obj. 1 (extensión del dataset), base empírica de todos los objetivos
Técnicas de Evaluación	Obj. 4 (desarrollo de pipeline de adherencia), Obj. 5 (métricas cuantitativas)
Sistemas Alternativos	Justificación de decisiones tecnológicas y metodológicas
Limitaciones y Gaps	Justificación de la contribución y alcance del trabajo

Cuadro 4: Mapeo explícito entre secciones del marco teórico y objetivos específicos del trabajo de graduación. Se incluyen las nuevas secciones sobre Krumhansl-Schmuckler validado y métodos de inferencia condicional.

Coherencia del Marco Conceptual

El marco teórico presentado cumple con las funciones establecidas en la introducción:

- Organización de conocimientos:** Integra teorías de psicología musical, teoría musical computacional, y aprendizaje profundo en un sistema coordinado.
- Evitación de duplicación:** Identifica gaps específicos no abordados por investigación existente.
- Selección de variables:** Justifica la elección de emoción, tonalidad, duración y velocidad como variables de control.
- Prevención de sesgos:** Reconoce limitaciones de datasets y metodologías existentes.
- Orientación interpretativa:** Proporciona marco para interpretar resultados de evaluación.

Reflexión Final

Este marco teórico establece que la generación musical emocional efectiva con control multiparamétrico requiere la integración de:

- **Modelos psicológicos de representación emocional:** El modelo circunplejo de Russell proporciona un espacio bidimensional para especificar emociones objetivo
- **Algoritmos validados de análisis musical computacional:** El algoritmo de Krumhansl-Schmuckler, empíricamente validado por Frankland y Cohen, permite tanto la caracterización automática de tonalidad en datasets como la evaluación de adherencia tonal en composiciones generadas
- **Arquitecturas neuronales profundas:** Los Transformers con mecanismos de atención multi-cabeza constituyen la base arquitectónica para modelar dependencias de largo alcance en secuencias musicales
- **Técnicas de condicionamiento multiparamétrico:** La incorporación de múltiples variables de control (emoción, tonalidad, duración, velocidad) como embeddings en la entrada del modelo permite generación dirigida
- **Métodos diversos de inferencia:** La disponibilidad de estrategias de inferencia estocástica, determinística y con tokens forzados permite explorar trade-offs entre adherencia al condicionamiento y coherencia estructural
- **Métricas de evaluación integral:** La cuantificación de adherencia tanto a variables directamente controladas como a propiedades emergentes (emoción global, coherencia tonal) requiere pipelines de evaluación automática basados en los mismos algoritmos de análisis musical

El sistema extendido EMOPIA propuesto en este trabajo representa una síntesis de estos elementos, ofreciendo:

1. **Control explícito multiparamétrico:** Variables de emoción (4 cuadrantes) y tonalidad (24 opciones) como condicionamiento de alto nivel, complementadas con control opcional sobre duración y velocidad
2. **Flexibilidad metodológica:** Tres modos de inferencia distintos (estocástico-condicional, determinístico-condicional, forzado) que permiten investigar diferentes paradigmas de control
3. **Evaluabilidad rigurosa:** Pipeline automático que cuantifica adherencia usando algoritmos validados (Krumhansl-Schmuckler para tonalidad) y clasificadores para emoción
4. **Fundamentación teórica sólida:** Cada componente del sistema (desde la representación emocional hasta los métodos de inferencia) está respaldado por literatura académica establecida

La contribución del presente trabajo radica en la integración sistemática de estos componentes teóricos en un sistema funcional que extiende las capacidades de EMOPIA, incorporando control tonal basado en teoría musical computacional y métodos de inferencia que permiten estudiar empíricamente los trade-offs entre diferentes estrategias de condicionamiento multiparamétrico.

CAPÍTULO 6

Metodología

El desarrollo de este trabajo de graduación siguió una metodología estructurada en fases secuenciales, diseñada para abordar sistemáticamente cada uno de los objetivos específicos establecidos. La metodología integra fundamentos de teoría musical computacional, específicamente el algoritmo de detección de tonalidad de Krumhansl-Schmuckler, con técnicas de aprendizaje profundo basadas en arquitecturas Transformer para la extensión del sistema EMOPIA y la implementación de métodos de inferencia multiparamétrica.

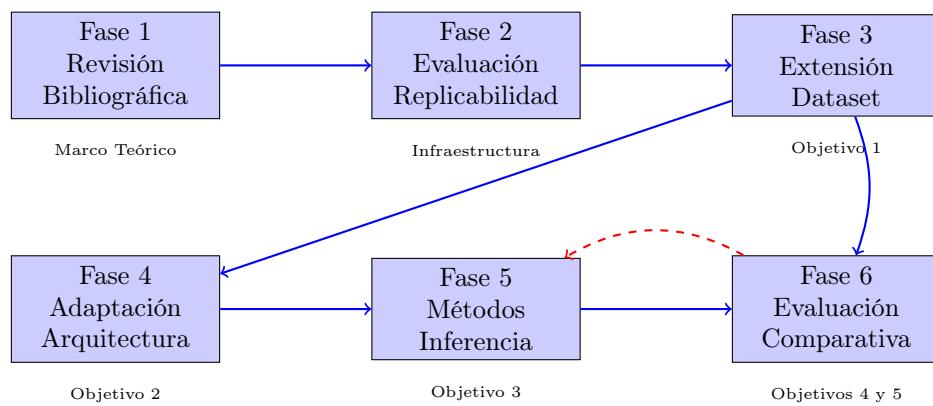


Figura 6: Flujo metodológico del trabajo de graduación. Las fases se diseñaron para abordar secuencialmente los objetivos específicos, desde la fundamentación teórica hasta la evaluación comparativa del sistema extendido.

6.1. Fase 1: Revisión bibliográfica y fundamentación teórica

Esta fase estableció las bases conceptuales, metodológicas y tecnológicas del trabajo, proporcionando el marco teórico documentado en el Capítulo 2 de este documento.

6.1.1. Descripción de actividades realizadas

Revisión sistemática de literatura

Se realizó una investigación bibliográfica exhaustiva sobre los fundamentos teóricos y tecnológicos relevantes para el proyecto, estructurada en cinco áreas temáticas principales:

1. **Modelos de representación emocional en música:** Se revisaron las teorías psicológicas de emoción musical, con énfasis en el modelo circumplejo de Russell [8] que constituye la base del sistema EMOPIA. Esta revisión incluyó estudios sobre la relación entre características musicales específicas (tempo, modo, armonía) y percepciones emocionales [12]-[14].
2. **Teoría musical computacional:** Se examinaron algoritmos de análisis musical automatizado, particularmente el algoritmo de Krumhansl-Schmuckler [19] para detección de tonalidad y su validación empírica [20]. Esta revisión fue fundamental para el Objetivo Específico 1 sobre extensión del dataset con información de tonalidad.
3. **Arquitecturas de aprendizaje profundo:** Se estudiaron los fundamentos de redes neuronales profundas, específicamente la arquitectura Transformer [24] y sus adaptaciones para generación musical [27]. Esta revisión sustentó el Objetivo Específico 2 sobre adaptación arquitectónica.
4. **Generación musical condicional:** Se analizaron técnicas de condicionamiento multíparamétrico [30], [31] y métodos de inferencia (muestreo estocástico, determinístico, tokens forzados), directamente relacionados con el Objetivo Específico 3.
5. **Evaluación de música generada:** Se revisaron metodologías de evaluación [33], [34] y métricas objetivas para cuantificar adherencia al condicionamiento, fundamentando los Objetivos Específicos 4 y 5.

Análisis comparativo de sistemas existentes

Se realizó una evaluación sistemática de sistemas alternativos de generación musical (Suno, ElevenLabs, Jukebox, Music Transformer, MuseGAN, DeepBach, EmotionBox, EmoGen) según criterios de: control emocional explícito, especialización en piano, disponibilidad de dataset anotado, código abierto, y representación simbólica. Este análisis justificó la selección de EMOPIA como sistema base para el proyecto.

Caracterización del dataset EMOPIA

Se documentaron las características técnicas del dataset EMOPIA [1]:

- Composición: 1,087 clips musicales de piano pop de 30 segundos
- Formato: Archivos MIDI (representación simbólica) y audio (YouTube IDs)
- Anotación emocional: Cuatro cuadrantes del modelo valencia-activación
- Distribución: Q1 (250 clips), Q2 (265 clips), Q3 (253 clips), Q4 (310 clips)

Esta caracterización no incluyó análisis estadístico de patrones musicales, ya que el alcance del proyecto se enfocó en la extensión mediante variables de teoría musical computacional más que en análisis exploratorio de datos.

6.1.2. Materiales y recursos utilizados

- **Bases de datos académicas:** IEEE Xplore, ACM Digital Library, arXiv, Google Scholar
- **Gestión bibliográfica:** BibTeX para la organización y manejo de referencias
- **Período de ejecución:** Julio - Agosto 2025
- **Responsable:** Samuel Alejandro Chamalé Rac

6.1.3. Productos de la Fase 1

1. **Marco teórico documentado:** Capítulo completo que integra música y emoción, teoría musical computacional, aprendizaje profundo, generación musical condicional, el dataset EMOPIA, sistemas alternativos, limitaciones de investigación existente, y síntesis conceptual (aproximadamente 30 referencias bibliográficas).
2. **Justificación de decisiones tecnológicas:** Análisis comparativo fundamentado para la selección de EMOPIA como sistema base (Cuadro 3 en el marco teórico).
3. **Mapeo teoría-objetivos:** Tabla explícita que relaciona cada sección del marco teórico con objetivos específicos del trabajo (Cuadro 4).
4. **Identificación de gaps de investigación:** Documentación de limitaciones en control multiparamétrico, evaluación de adherencia, y escalabilidad de condicionamiento que este trabajo aborda.

6.2. Fase 2: Evaluación de replicabilidad del sistema EMOPIA

Esta fase evaluó la viabilidad técnica de replicar el sistema EMOPIA original, identificando requisitos de infraestructura computacional, dependencias de software, y procedimientos operativos necesarios para el desarrollo del proyecto. Esta evaluación fue fundamental para establecer la línea base sobre la cual se construirían las extensiones propuestas.

6.2.1. Descripción de actividades realizadas

Análisis de requisitos del sistema original

Se realizó un análisis exhaustivo del repositorio oficial de EMOPIA [1] disponible en GitHub¹ para identificar todos los componentes necesarios para replicar el sistema:

1. **Revisión de documentación:** Se examinó el README principal, documentación de dataset, y scripts de ejemplo para comprender la arquitectura del sistema y sus dependencias.
2. **Identificación de dependencias de software:** Se catalogaron todas las bibliotecas y frameworks requeridos:
 - PyTorch 1.7.0 (framework de aprendizaje profundo)
 - pytorch-fast-transformers (implementación eficiente de Transformers)
 - miditoolkit 0.1.14 (manipulación de archivos MIDI)
 - scikit-learn 0.24.1 (métricas de evaluación)
 - numpy 1.19.5, pandas 1.1.5 (procesamiento de datos)
 - Bibliotecas auxiliares: seaborn, matplotlib, scipy, tqdm, ipdb
3. **Análisis de compatibilidad de versiones:** Se identificaron restricciones de versiones específicas, particularmente PyTorch 1.7.0 que requiere compatibilidad con versiones específicas de CUDA según la GPU disponible.

Evaluación de requisitos de datos

Se identificaron tres componentes de datos necesarios:

1. **Dataset EMOPIA procesado:** Archivos npz con representación tokenizada de clips musicales
 - **Opción 1 (recomendada):** Dataset preprocessado disponible mediante Google Drive (ID: 17dKUF33ZsDbHC5Z6rkQclge3ppDTVCMP)
 - **Opción 2 (replicabilidad completa):** Procesamiento desde cero:

¹<https://github.com/annahung31/EMOPIA>

- Dataset EMOPIA v2.2 desde Zenodo [2] (1,071 clips MIDI)
- Descarga de audios de YouTube usando yt-dlp [46] y FFmpeg [47]
- Pipeline de preprocesamiento de Compound Word Transformer [48]
- Requiere Python 3.9 por dependencias específicas (numpy==1.19.5, madmom)

2. **Checkpoints preentrenados:** Tres modelos disponibles para inferencia:

- Baseline (LSTM-based)
- Transformer sin preentrenamiento
- Transformer con preentrenamiento en AILabs17k

3. **Dataset AILabs17k (opcional):** Necesario solo para reentrenar el modelo con pre-entrenamiento desde cero.

Verificación de capacidad de inferencia

Se evaluó la viabilidad de ejecutar el proceso de generación musical:

1. **Análisis de opciones de inferencia:** Se documentaron los parámetros configurables del sistema original:

- `task_type`: Tipo de condicionamiento ('4-cls', 'Arousal', 'Valence', 'ignore')
- `emo_tag`: Categoría emocional objetivo (1-4 para 4-cls)
- `num_songs`: Cantidad de composiciones a generar
- `out_dir`: Directorio de salida para archivos MIDI generados

2. **Identificación de proceso de generación:** Se examinó el script `main_cp.py` y el notebook `generate.ipynb` para comprender el flujo de inferencia.

3. **Análisis de formato de salida:** Se determinó que el sistema genera archivos MIDI directamente, los cuales pueden reproducirse mediante sintetizadores de piano o convertirse a audio.

Evaluación de capacidad de entrenamiento

Se analizó la viabilidad de reentrenar o fine-tunear el modelo:

1. **Opciones de entrenamiento identificadas:**

- Entrenamiento desde cero en EMOPIA (no-pretrained transformer)
- Pre-entrenamiento en AILabs17k seguido de fine-tuning en EMOPIA

2. **Requisitos computacionales:** Se identificó la necesidad de GPU con soporte CUDA para entrenamiento eficiente, con opción de paralelización de datos (`data_parallel=1`).

3. Proceso de preparación de datos: Se documentó y evaluó la replicabilidad del pipeline completo desde cero siguiendo la metodología de Compound Word Transformer [48]:

- **Obtención del dataset original:** Descarga de EMOPIA v2.2 desde Zenodo [2], conteniendo 1,071 clips MIDI de 387 videos únicos de YouTube
- **Descarga de audios complementarios:** Desarrollo de script personalizado (`download_timestamp_clips.py`) utilizando yt-dlp [46] y FFmpeg [47] para descargar audios de YouTube según timestamps, resultando en 854 clips de audio exitosamente descargados (320 videos únicos, 67 videos no disponibles)
- **Pipeline de preprocesamiento:**
 - a) Paso 1-3: Procesamiento según Compound Word Transformer (requiere Python 3.9 por dependencias específicas de numpy==1.19.5 y madmom)
 - b) `synchronizer.py`: Sincronización de audio y MIDI
 - c) `analyzer.py`: Análisis de características musicales (modificado para omitir archivos con tempo 0)
 - d) `midi2corpus.py`: Cuantización y extracción de etiquetas emocionales desde nombres de archivo
 - e) `corpus2events.py`: Conversión a eventos de Compound Phrase (CP)
 - f) `event2words.py`: Tokenización y construcción de diccionario
 - g) `compile.py`: Compilación a formato npz para entrenamiento

Evaluación de replicabilidad del dataset desde scratch

Para garantizar la reproducibilidad completa del proyecto, se evaluó la viabilidad de preparar el dataset EMOPIA desde cero, siguiendo la documentación oficial del repositorio [48]. Este proceso complementó el uso del dataset preprocesado y demostró que es posible replicar completamente el pipeline de datos.

Obtención del dataset original Se descargó el dataset EMOPIA v2.2 desde Zenodo [2], que contiene:

- 1,071 clips MIDI anotados emocionalmente
- 387 videos únicos de YouTube como fuente original
- Archivo `timesteps.json` con marcas temporales de inicio y fin de cada clip
- Nomenclatura de archivos: `Q{label}_{youtube_id}_{index}.mid`

Descarga automatizada de audios Dado que los archivos de audio no están incluidos en el repositorio de Zenodo (solo los MIDI), se desarrolló un script personalizado `download_timestamp_clips.py` para descargar automáticamente los audios correspondientes desde YouTube:

- **Herramientas utilizadas:**

- yt-dlp [46]: Descarga de audio de YouTube en formato MP3
- FFmpeg [47]: Recorte de clips según timestamps y conversión de formatos

- **Proceso de descarga:**

1. Descarga del audio completo de cada video de YouTube
2. Segmentación en clips individuales según `timesteps.json`
3. Conversión a MP3 a 44.1kHz para compatibilidad con el pipeline

- **Resultados de descarga:**

- Videos descargados exitosamente: 320 de 387 (82.7 %)
- Videos no disponibles: 67 (17.3 %, removidos o privados)
- Clips de audio generados: 854 de 1,071 (79.7 %)
- Clips omitidos por falta de audio fuente: 217 (20.3 %)
- Tiempo total de descarga: 6,218 segundos (1.7 horas)

- **Manejo de errores:** El script implementa reintentos automáticos con limpieza de caché para mitigar errores HTTP 416 (Range Not Satisfiable), comunes en descargas de YouTube.

Implicaciones para el proyecto La evaluación de replicabilidad del dataset demostró:

- **Viabilidad:** Es posible preparar el dataset completo desde cero sin acceso al preprocesado
- **Limitaciones:** La disponibilidad de videos de YouTube disminuye con el tiempo (17.3 % no disponibles en enero 2025 vs. publicación original en 2021)
- **Recomendación práctica:** Para iniciar rápidamente, usar el dataset preprocesado; para replicabilidad completa o extensiones del dataset, seguir el proceso desde scratch
- **Documentación:** El script `download_timestamp_clips.py` desarrollado está disponible en el repositorio del proyecto para futuros usuarios

Configuración del entorno de desarrollo local

Se estableció inicialmente un entorno computacional local para evaluación preliminar del sistema:

- **Hardware:** NVIDIA GeForce RTX 2060 (6GB VRAM, laptop)
- **Sistema operativo:** Windows 11
- **CUDA:** versión 12.9 (Build `cuda_12.9.r12.9/compiler.35813241_0`)

- **Python:** versión 3.13.2
- **PyTorch:** versión 2.8.0+cu129 con soporte CUDA
- **Bibliotecas complementarias:** pytorch-fast-transformers, miditoolkit, numpy, pandas, según `requirements.txt`
- **Control de versiones:** Git para gestión de código

La Figura 7 muestra las especificaciones exactas del entorno local configurado.

```
S:\UVG\LOCKIN\tesis git:(main) 9 files changed, 196 insertions(+), 202 deletions(-) (0.141s)
nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2025 NVIDIA Corporation
Built on Wed_Apr_9_19:29:17_Pacific_Daylight_Time_2025
Cuda compilation tools, release 12.9, V12.9.41
Build cuda_12.9.r12.9/compiler.35813241_0

S:\UVG\LOCKIN\tesis git:(main) 9 files changed, 196 insertions(+), 202 deletions(-) (0.067s)
python --version
Python 3.13.2

S:\UVG\LOCKIN\tesis git:(main) 9 files changed, 196 insertions(+), 202 deletions(-) (3.601s)
pip show torch
Name: torch
Version: 2.8.0+cu129
Summary: Tensors and Dynamic neural networks in Python with strong GPU acceleration
Home-page: https://pytorch.org/
Author: PyTorch Team
Author-email: packages@pytorch.org
License: BSD-3-Clause
Location: C:\Users\chama\AppData\Local\Programs\Python\Python313\Lib\site-packages
Requires: filelock, fsspec, jinja2, networkx, setuptools, sympy, typing-extensions
Required-by: pytorch-fast-transformers, torchaudio, torchvision
```

Figura 7: Configuración del entorno de desarrollo local, mostrando versiones de CUDA 12.9, Python 3.13.2, y PyTorch 2.8.0+cu129 instaladas en Windows 11 con GPU RTX 2060.

Evaluación de rendimiento en entorno local

Se ejecutó una prueba de entrenamiento del modelo EMOPIA original utilizando el script `main_cp.py` con los siguientes parámetros:

```
python transformer/main_cp.py --path_train_data train
    --exp_name checkpoints --load_ckt none
    --load_dict train_dictionary.pkl
```

Los resultados de esta prueba inicial revelaron características de rendimiento importantes:

- **Arquitectura del modelo:** 39,168,153 parámetros

- **Tamaño del batch de entrenamiento:** torch.Size([824, 1024, 9])
- **Número de batches por época:** 206
- **Tiempo de ejecución por época:** 1 minuto 27 segundos (87.22 segundos)
- **Loss inicial:** 1.5370 (epoch 0)

La Figura 8 muestra la salida completa de la primera época de entrenamiento en el entorno local.

```
train_x: torch.Size([824, 1024, 9])
num of classes: [53, 127, 18, 5, 85, 18, 41, 5, 25]
>>>: [53, 127, 18, 5, 85, 18, 41, 5, 25]
n_parameters: 39,168,153
    num_batch: 206
----- 1.344202, 1.286495, 0.722695, 2.509074, 1.693421, 1.922520, 0.031320
epoch: 0/4000 | Loss: 1.536981459381511 | time: 0:01:27.224800
    > 1.429199, 1.608161, 1.469720, 0.648407, 3.304919, 2.137946, 2.764186, 0.168971
[*] saving model to exp/checkpoints, name: loss_high
    num_batch: 206
```

Figura 8: Tiempo de entrenamiento de una época completa en el entorno local (RTX 2060): 1 minuto 27 segundos para procesar 206 batches con un modelo de 39.2 millones de parámetros.

Análisis de viabilidad y selección de infraestructura en la nube

Dado que el entrenamiento completo del modelo EMOPIA requiere 4,000 épocas según la documentación original, y considerando el tiempo de 87.22 segundos por época en el entorno local, se calculó que el entrenamiento completo requeriría aproximadamente 97 horas (4 días) de cómputo continuo. Además, las limitaciones de VRAM de la GPU RTX 2060 (6GB) podrían restringir la capacidad de experimentar con tamaños de batch más grandes o extensiones del modelo.

Por estas razones, se evaluó la necesidad de migrar a infraestructura de GPU en la nube para las fases posteriores del proyecto que involucrarían reentrenamiento del modelo con variables de condicionamiento adicionales.

Comparación de servicios de GPU en la nube Se realizó un análisis comparativo de los principales proveedores de infraestructura de GPU en la nube, evaluando criterios relevantes para el proyecto:

Selección de RunPod Se seleccionó RunPod [49] como plataforma de infraestructura en la nube por las siguientes razones:

1. **Costo-efectividad:** Precio competitivo de \$0.90/hora para GPU RTX 5090, significativamente inferior a proveedores tradicionales (AWS, Google Cloud, Azure)
2. **Facilidad de configuración:** Plantillas preconfiguradas con PyTorch y CUDA instalados (template `runpod-torch-v280`), reduciendo tiempo de setup

Proveedor	Precio/hora (RTX 4090)	Facilidad de uso	GPU disponibles	Flexibilidad de pago
RunPod [49]	\$0.90	Alta	30+ SKUs	Pay-per-use
AWS EC2 (p3.2xlarge) [50]	\$3.06	Media	V100, A100, H100	Por hora/reserva
Google Cloud (a2) [51]	\$2.93	Media	A100, L4	Por hora/reserva
Azure NC-series [52]	\$3.06	Media	V100, A100	Por hora/reserva
Vast.ai [53]	\$0.75-1.20	Media-Baja	Variable	Pay-per-use
Lambda Labs [54]	\$1.10	Alta	RTX 4090, A100	Por hora

Cuadro 5: Comparación de proveedores de infraestructura de GPU en la nube según criterios relevantes para el proyecto. Los precios son aproximados y corresponden a GPUs de gama alta disponibles en cada plataforma.

3. **Modelo de pago flexible:** Facturación por uso exacto sin compromisos de tiempo mínimo o contratos de reserva
4. **Disponibilidad de GPUs modernas:** Acceso a RTX 5090 con 24GB VRAM, permitiendo experimentación con configuraciones de modelo más grandes
5. **Escalabilidad:** Capacidad de escalar verticalmente (GPUs más potentes) u horizontalmente (múltiples instancias) según necesidades del proyecto

Configuración del entorno de desarrollo en la nube

Se configuró un pod de RunPod con las siguientes especificaciones:

- **GPU:** NVIDIA GeForce RTX 5090 (24GB VRAM)
- **vCPU:** 16 cores
- **Memoria RAM:** 141 GB
- **Almacenamiento:** 30 GB Container Disk + 50 GB Persistent Volume
- **Template:** runpod-torch-v280 (PyTorch 2.8.0 + CUDA 12.8 preinstalado)
- **Sistema operativo:** Ubuntu Linux (contenedor Docker)
- **CUDA:** versión 12.8 (Build cuda_12.8.r12.8/compiler.35583870_0)
- **Python:** versión 3.12.3
- **PyTorch:** versión 2.8.0+cu128 con soporte CUDA
- **Costo:** \$0.9011/hora (incluye almacenamiento)

La Figura 9 muestra las especificaciones del pod utilizado, y la Figura 10 detalla las versiones de software instaladas.

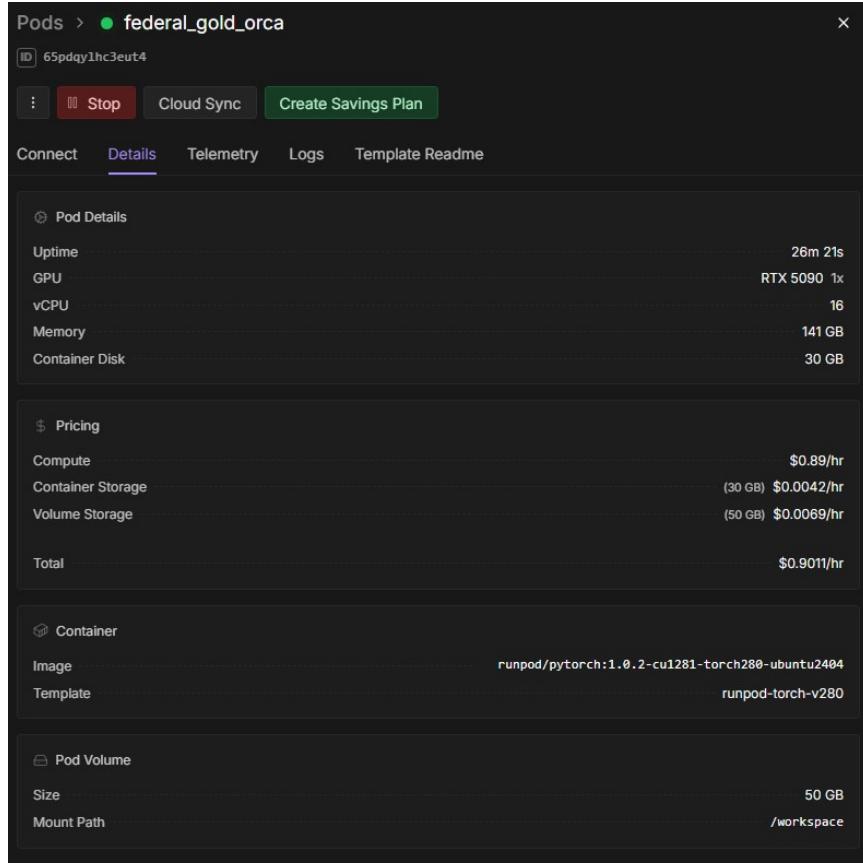


Figura 9: Especificaciones del pod de RunPod utilizado para entrenamiento, mostrando GPU RTX 5090, 16 vCPU, 141GB RAM, y pricing de \$0.9011/hora.

Validación de rendimiento en la nube

Se ejecutó la misma prueba de entrenamiento en el entorno de RunPod para validar la mejora de rendimiento:

```
python transformer/main_cp.py --path_train_data train
    --exp_name checkpoints --load_ckt none
    --load_dict train_dictionary.pkl
```

Los resultados mostraron una mejora significativa en el tiempo de ejecución:

- **Tiempo de ejecución por época:** 13.74 segundos
- **Aceleración:** $6.35 \times$ más rápido que el entorno local (87.22s vs 13.74s)
- **Loss inicial:** 1.5370 (idéntico al entorno local, validando correctitud)
- **Tiempo estimado para 4,000 épocas:** 15.3 horas (vs 97 horas en local)

La Figura 11 muestra la salida del entrenamiento en el entorno de RunPod.

```

root@45cc31672b94:/workspace/emotions# nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2025 NVIDIA Corporation
Built on Fri_Feb_21_20:23:50_PST_2025
Cuda compilation tools, release 12.8, V12.8.99
Build cuda_12.8.99/compiler.39583870.9
root@45cc31672b94:/workspace/emotions# python --version
Python 3.12.3
root@45cc31672b94:/workspace/emotions# pip show torch
Name: torch
Version: 2.8.0+cu128
Summary: Tensors and Dynamic neural networks in Python with strong GPU acceleration
Home-page: https://pytorch.org/
Author: PyTorch Team
Author-email: packages@pytorch.org
License: BSD-3-Clause
Location: /usr/local/lib/python3.12/dist-packages
Requires: filelock, fsspec, jinja2, networkx, nvidia-cublas-cu12, nvidia-cuda-cupti-cu12, nvidia-cuda-mvrtc-cu12, nvidia-cuda-runtime-cu12, nvidia-cudnn-cu12, nvidia-cufft-cu12, nvidia-cufile-cu12, nvidia-curand-cu12, nvidia-cusolver-cu12, nvidia-cusparse-cu12, nvidia-cusparselt-cu12, nvidia-ncc1-cu12, nvidia-nvjitlink-cu12, nvidia-nvtx-cu12, setuptools, sympy, triton, typing-extensions
Required-by: pytorch-fast-transformers, torchaudio, torchvision
root@45cc31672b94:/workspace/emotions#

```

Figura 10: Configuración del entorno de desarrollo en RunPod, mostrando CUDA 12.8, Python 3.12.3, y PyTorch 2.8.0+cu128 con dependencias NVIDIA completas.

```

train_x: torch.Size([824, 1024, 9])
num of classes: [53, 127, 18, 5, 85, 18, 41, 5, 25]
>>>: [53, 127, 18, 5, 85, 18, 41, 5, 25]
n_parameters: 39,168,153
    num_batch: 206
----- 1.343534, 1.283884, 0.715442, 2.508748, 1.690993, 1.923148, 0.031340
epoch: 0/4000 | Loss: 1.5369687369726237 | time: 0:00:13.744103
    > 1.428981, 1.607926, 1.470067, 0.648469, 3.305306, 2.137854, 2.764098, 0.168802
[*] saving model to exp/checkpoints, name: loss_high
    num_batch: 206

```

Figura 11: Tiempo de entrenamiento de una época completa en RunPod (RTX 5090): 13.74 segundos, representando una aceleración de $6.35 \times$ respecto al entorno local. La consistencia del loss inicial (1.5370) valida la correcta configuración del entorno.

Análisis de costo-beneficio El análisis comparativo de costo-beneficio entre el entorno local y la nube reveló:

- **Entorno local:** Tiempo de entrenamiento completo: 97 horas. Costo directo: \$0 (hardware ya disponible). Limitaciones: Uso exclusivo de hardware personal, restricciones de VRAM.
- **RunPod (RTX 5090):** Tiempo de entrenamiento completo: 15.3 horas. Costo estimado: \$13.78 por entrenamiento completo. Beneficios: Liberación de hardware personal, mayor VRAM (24GB vs 6GB), capacidad de paralelizar múltiples experimentos.

La decisión de utilizar RunPod para las fases posteriores se justificó por la reducción de 84 % en tiempo de entrenamiento y la capacidad de realizar múltiples experimentos en paralelo, acelerando significativamente el desarrollo del proyecto.

6.2.2. Materiales y recursos utilizados

- **Repositorio original:** <https://github.com/annahung31/EMOPIA>
- **Documentación de referencia:** README de EMOPIA, documentación de PyTorch 2.8.0, documentación de fast-transformers, documentación de RunPod
- **Herramientas de gestión de paquetes:** pip para instalación de dependencias Python
- **Infraestructura computacional:**
 - Entorno local: Laptop con GPU RTX 2060, Windows 11

- Entorno en la nube: RunPod con GPU RTX 5090, Ubuntu Linux
- **Período de ejecución:** Septiembre 2025
- **Ubicación:** Desarrollo remoto (entorno local y RunPod en la nube)

6.2.3. Productos de la Fase 2

1. **Documentación de requisitos técnicos:** Especificación completa de dependencias de software con versiones exactas (PyTorch 2.8.0, CUDA 12.8/12.9, fast-transformers), requisitos de hardware mínimos (GPU con 6GB VRAM) y recomendados (GPU con 24GB VRAM), y procedimientos de instalación validados en Windows 11 y Ubuntu Linux.
2. **Entornos de desarrollo funcionales:**
 - Entorno local: Windows 11 con RTX 2060, PyTorch 2.8.0+cu129, CUDA 12.9
 - Entorno en la nube: RunPod con RTX 5090, PyTorch 2.8.0+cu128, CUDA 12.8
 - Validación exitosa de compatibilidad de ambos entornos con el código EMOPIA
3. **Dataset EMOPIA procesado:** Se empleó un enfoque dual para asegurar tanto la viabilidad práctica como la replicabilidad:
 - **Dataset preprocesado:** Descarga del dataset procesado (npz) desde Google Drive para iniciar rápidamente la posibilidad de inferencias
 - **Preparación desde scratch:** Evaluación completa de replicabilidad procesando el dataset original EMOPIA v2.2 [2] desde Zenodo:
 - 1,071 clips MIDI originales de 387 videos únicos de YouTube
 - 854 clips de audio descargados exitosamente (79.7 % de éxito)
 - Pipeline completo de preprocesamiento ejecutado siguiendo metodología de Compound Word Transformer [48]
 - Script personalizado desarrollado para descarga automatizada de audios usando yt-dlp [46] y FFmpeg [47]
4. **Checkpoints preentrenados obtenidos:** Descarga exitosa de modelos baseline, transformer sin preentrenamiento, y transformer preentrenado de EMOPIA para uso como referencia y línea base de comparación en la Fase 6.
5. **Métricas de rendimiento de entrenamiento:**
 - Entorno local: 87.22 segundos por época (206 batches, 39.2M parámetros)
 - Entorno RunPod: 13.74 segundos por época (aceleración de 6.35×)
 - Estimaciones de tiempo para entrenamiento completo: 97 horas (local) vs 15.3 horas (nube)
 - Evidencia documental mediante capturas de pantalla (Figuras 8 y 11)
6. **Análisis comparativo de infraestructura en la nube:**

- Tabla comparativa de proveedores (RunPod, AWS, Google Cloud, Azure, Vast.ai, Lambda Labs)
- Justificación técnica y económica de la selección de RunPod
- Análisis de costo-beneficio: \$13.78 por entrenamiento completo en RunPod vs 97 horas de tiempo de desarrollo en local

7. Informe de replicabilidad:

Documento técnico que identifica:

- Componentes del sistema EMOPIA exitosamente replicados en ambos entornos
- Compatibilidad verificada entre PyTorch 2.8.0 y código original (diseñado para PyTorch 1.7.0)
- Diferencias de rendimiento cuantificadas entre entornos local y en la nube
- **Evaluación de replicabilidad del dataset:**
 - Pipeline completo de preparación desde scratch validado exitosamente
 - Script personalizado `download_timestamp_clips.py` desarrollado y documentado
 - 79.7 % de clips recuperables desde YouTube (854 de 1,071)
 - 20.3 % de pérdida por videos no disponibles o removidos
 - Tiempo de preparación: 1.7 horas de descarga + tiempo de preprocesamiento
- Recomendaciones para la extensión del sistema: utilizar RunPod para reentrenamiento con variables adicionales en Fase 4
- Documentación de limitaciones de replicabilidad a largo plazo debido a disponibilidad variable de videos de YouTube

8. Configuración de infraestructura en la nube:

- Pod de RunPod configurado y validado (RTX 5090, 16 vCPU, 141GB RAM)
- Persistent Volume de 50GB para almacenamiento de datasets y checkpoints
- Template `runpod-torch-v280` con PyTorch y CUDA preinstalados
- Scripts de sincronización de datos entre entorno local y RunPod

6.2.4. Conexión con objetivos específicos

Esta fase preparatoria fue esencial para todos los objetivos específicos posteriores:

- **Fundamento para Objetivo 1:** La comprensión del formato de datos de EMOPIA y el pipeline de procesamiento permitió posteriormente extender el dataset con información de tonalidad.
- **Fundamento para Objetivo 2:** El análisis de la arquitectura del sistema original identificó los puntos de extensión necesarios para incorporar la variable de tonalidad como parámetro de condicionamiento.
- **Fundamento para Objetivo 3:** La evaluación de los métodos de inferencia del sistema original estableció la línea base sobre la cual se implementarían los métodos de inferencia multiparamétrica.

- **Fundamento para Objetivos 4 y 5:** La capacidad de ejecutar inferencias con el sistema original fue prerequisito para generar composiciones de referencia contra las cuales comparar el sistema extendido.

6.3. Fase 3: Extensión del dataset con información de tonalidad

Esta fase abordó la extensión del dataset EMOPIA con información de tonalidad mediante la implementación del algoritmo de Krumhansl-Schmuckler para detección automática de tonalidad y su integración en el pipeline de procesamiento del dataset. La fase se dividió en dos etapas principales: primero, la preparación del dataset mediante el pipeline de Compound Word Transformer para obtener archivos MIDI sincronizados temporalmente con el audio, y segundo, la implementación del algoritmo de detección de tonalidad y la ejecución del pipeline de procesamiento de EMOPIA para generar el dataset tokenizado final con la novena dimensión de tonalidad integrada.

6.3.1. Descripción de actividades realizadas

La fase se dividió en dos etapas principales: (1) preparación del dataset mediante el pipeline de Compound Word Transformer, y (2) implementación del algoritmo de detección de tonalidad y ejecución del pipeline de procesamiento de EMOPIA para generar el dataset tokenizado con la variable de tonalidad integrada.

Etapa 1: Preparación del dataset mediante pipeline de Compound Word Transformer

Justificación de preparación del dataset desde scratch En la Fase 2 se obtuvo y utilizó el dataset EMOPIA preprocesado para evaluación de replicabilidad del sistema, la extensión con información de tonalidad requirió preparar el dataset completo desde cero. Esta decisión se fundamentó en las siguientes razones:

1. **Acceso a representación MIDI completa:** La detección de tonalidad mediante el algoritmo de Krumhansl-Schmuckler requiere analizar la distribución de clases de pitch (pitch classes) en los archivos MIDI, información que no está disponible en el dataset preprocesado (formato npz tokenizado).
2. **Sincronización audio-MIDI:** El pipeline de Compound Word Transformer genera archivos MIDI sincronizados con beat tracking de audio, lo cual mejora la precisión temporal de las características musicales extraídas.
3. **Extracción de características musicales:** Los scripts `synchronizer.py` y `analyzer.py` de Compound Word Transformer extraen metadatos musicales (tempo, acordes) que son prerequisitos para etapas posteriores del pipeline.

4. **Integración con pipeline existente:** La detección de tonalidad debía integrarse como paso adicional en el flujo de preprocesamiento antes de la tokenización, requiriendo acceso a los archivos MIDI intermedios generados por el pipeline.

Configuración del entorno de procesamiento especializado La ejecución de los scripts `synchronizer.py` y `analyzer.py` del repositorio Compound Word Transformer [48] requirió un entorno de desarrollo con dependencias específicas incompatibles con el entorno principal del proyecto (PyTorch 2.8.0, Python 3.13.2). Específicamente, la biblioteca `madmom` [55] para beat tracking y downbeat detection presentaba incompatibilidades con versiones modernas de NumPy.

Según la documentación oficial de Compound Word Transformer, `madmom` es esencial para el proceso de sincronización:

“In this step, we use `madmom` for beat/downbeat tracking. Next, We interpolate 480 ticks between two adjacent beats, and map the absolute time into its according tick. Lastly, we infer the tempo changes from the time interval between adjacent beats. We choose beat resolution=480 because it’s a common setting in modern DAW. Notice that we don’t quantize any timing in this step hence we can keep tiny offset for future purposes.” [48]

Por lo tanto, se configuró un entorno virtual aislado con Python 3.9 específicamente para esta etapa de preprocesamiento. El proceso de configuración en el entorno local (Windows 11) se muestra en la Figura 12.

```
# Verificación de versiones disponibles de Python
py --list-paths

# Instalación de Python 3.9 mediante winget
winget install Python.Python.3.9

# Verificación de instalación
py --list-paths

# Creación de entorno virtual con Python 3.9
py -3.9 -m venv venv_py39

# Activación del entorno virtual
venv_py39\Scripts\activate

# Actualización de herramientas base
pip install --upgrade setuptools wheel

# Instalación de dependencias numéricas (orden específico)
pip install numpy scipy cython

# Instalación de bibliotecas de procesamiento musical
pip install librosa madmom

# Instalación de herramientas MIDI
pip install miditoolkit
```

Figura 12: Comandos de configuración del entorno virtual Python 3.9 para procesamiento de sincronización MIDI con `madmom`

Este entorno permitió ejecutar exitosamente el pipeline de Compound Word Transformer sin interferir con el entorno principal del proyecto.

Modificación del script `synchronizer.py` El script `synchronizer.py` del repositorio original de Compound Word Transformer requirió dos adaptaciones críticas para funcionar

```

compound-word-transformer\dataset on \ main [!?] via v3.13.2
> py --list-paths
-V:3.13 * C:\Users\chama\AppData\Local\Programs\Python\Python313\python.exe
-V:3.11 C:\Users\chama\AppData\Local\Programs\Python\Python311\python.exe
-V:3.9 C:\Users\chama\AppData\Local\Programs\Python\Python39\python.exe
-V:3.8 C:\Users\chama\AppData\Local\Programs\Python\Python38\python.exe

compound-word-transformer\dataset on \ main [!?] via v3.13.2
> venv_py39\Scripts\activate

compound-word-transformer\dataset on \ main [!?] via v3.9.13 (venv_py39)
> pip show madmom
Name: madmom
Version: 0.16.1
Summary: Python audio signal processing library
Home-page: https://github.com/CPJKU/madmom
Author: Department of Computational Perception, Johannes Kepler University, Linz, Austria and Austrian Research Institute for Artificial Intelligence (OFAI), Vienna, Austria
Author-email: madmom-users@googlegroups.com
License: BSD, CC BY-NC-SA
Location: s:\uvg\lockin\compound-word-transformer\dataset\venv_py39\lib\site-packages
Requires: cython, mido, numpy, scipy
Required-by:

compound-word-transformer\dataset on \ main [!?] via v3.9.13 (venv_py39)
> pip show miditoolkit
Name: miditoolkit
Version: 1.8.1
Summary: A python package for working with MIDI data files.
Home-page:
Author:
Author-email: wayne391 <s101062219@gmail.com>
License:
Location: s:\uvg\lockin\compound-word-transformer\dataset\venv_py39\lib\site-packages
Requires: matplotlib, mido, numpy
Required-by: chorder

```

Figura 13: Verificación de instalación de Python 3.9 y dependencias críticas (madmom, miditoolkit) en el entorno virtual

en el entorno local de Windows 11:

Adaptación 1: Configuración de FFmpeg para madmom La biblioteca `madmom` utiliza FFmpeg internamente para decodificar archivos de audio MP3. En sistemas Windows, es necesario especificar explícitamente la ubicación del ejecutable FFmpeg. Se agregaron las siguientes líneas al inicio del script:

```

# Configure ffmpeg path for madmom
ffmpeg_path = r'S:\ffmpeg-master-latest-win64-gpl\
                ffmpeg-master-latest-win64-gpl\bin'
os.environ['FFMPEG_BINARY'] = os.path.join(ffmpeg_path,
                                            'ffmpeg.exe')
os.environ['PATH'] = ffmpeg_path + os.pathsep + \
    os.environ.get('PATH', '')

```

Figura 14: Configuración de FFmpeg para madmom en Windows 11

Esta modificación configuró las variables de entorno `FFMPEG_BINARY` y `PATH` para que `madmom` pudiera localizar el ejecutable FFmpeg necesario para el procesamiento de audio.

Adaptación 2: Filtrado por archivos MP3 disponibles El script original listaba todos los archivos MIDI del directorio de entrada (`./midi_transcribed`) y asumía que existía un archivo MP3 correspondiente para cada uno. Sin embargo, como se documentó en la Fase 2, solo el 79.7% de los clips de audio fueron descargados exitosamente desde YouTube (854 de 1,071 clips), resultando en la ausencia de archivos MP3 para 217 clips MIDI.

Para evitar errores durante el procesamiento paralelo, se modificó el script para listar archivos basándose en los MP3 disponibles en lugar de los MIDI:

Esta modificación garantizó que solo se procesaran pares MIDI+MP3 completos, evitando errores de archivo no encontrado durante la sincronización.

```

# Versión original (líneas 214-218)
# list files
midifiles = traverse_dir(
    path_indir,                      # './midi_transcribed',
    is_ext=False,
    is_pure=True,
    is_sort=True)

# Versión modificada (líneas 218-224)
# mp3 list files
midifiles = traverse_dir(
    path_audiadir,                   # './mp3',
    extension=('mp3',),             # Solo archivos MP3
    is_ext=False,
    is_pure=True,
    is_sort=True)

```

Figura 15: Modificación del listado de archivos en synchronizer.py para filtrar por MP3 disponibles

Modificación del script analyzer.py El script `analyzer.py` extraía características musicales globales (tempo, acordes) de los archivos MIDI sincronizados. Durante las pruebas preliminares, se identificó que algunos archivos MIDI contenían valores de tempo igual a cero, lo que causaba errores de división por cero (`ZeroDivisionError`) durante el cálculo de BPM mediano, interrumpiendo la ejecución completa del pipeline.

Para garantizar la robustez del procesamiento, se agregó manejo de excepciones en la función `proc_one`:

```

# Versión original (líneas 68-116)
def proc_one(path_infile, path_outfile):
    print('---')
    print('>', path_infile)
    print('>', path_outfile)

    # load
    midi_obj = miditoolkit.midi.parser.MidiFile(path_infile)
    # ... procesamiento ...
    midi_obj_out.dump(path_outfile)

# Versión modificada (líneas 68-122)
def proc_one(path_infile, path_outfile):
    print('---')
    print('>', path_infile)
    print('>', path_outfile)

    try:
        # load
        midi_obj = miditoolkit.midi.parser.MidiFile(path_infile)
        # ... procesamiento ...
        midi_obj_out.dump(path_outfile)

    except (ZeroDivisionError, ValueError, Exception) as e:
        print(f'> ERROR processing {path_infile}: {e}')
        print('> Skipping this file...')

    return

```

Figura 16: Adición de manejo de excepciones en analyzer.py para archivos MIDI con tempo cero

Esta modificación permitió que el script omitiera archivos problemáticos sin interrumpir el procesamiento del resto del dataset, registrando los errores para revisión posterior.

Ejecución del pipeline de preprocessamiento Con las adaptaciones realizadas, se ejecutó el pipeline de Compound Word Transformer en dos pasos secuenciales:

Paso 1: Sincronización audio-MIDI Se ejecutó `synchronizer.py` para sincronizar los archivos MIDI con el beat tracking de audio:

Este proceso generó archivos MIDI sincronizados en el directorio `./midi_synchronized`, aplicando las siguientes transformaciones:

```

# Activar entorno Python 3.9
venv_py39\Scripts\activate

# Ejecutar sincronización
python synchronizer.py

```

Figura 17: Comandos de ejecución del script synchronizer.py

- **Beat tracking:** Detección automática de beats y downbeats mediante `madmom`
- **Interpolación de ticks:** Mapeo de 480 ticks por beat para resolución temporal consistente
- **Inferencia de tempo:** Cálculo de cambios de tempo basado en intervalos entre beats adyacentes
- **Alineación temporal:** Ajuste de time signatures para alinear el primer downbeat detectado

Paso 2: Análisis de características musicales Se ejecutó `analyzer.py` para extraer características musicales globales:

```
python analyzer.py
```

Figura 18: Comando de ejecución del script analyzer.py

Este proceso generó archivos MIDI anotados en el directorio `./midi_analyzed`, agregando:

- **Tempo global:** BPM mediano calculado de los primeros 40 tempo changes, almacenado como marker `global_bpm_X` en tiempo 0
- **Progresión de acordes:** Secuencia de acordes detectados mediante la biblioteca `chorder`, almacenados como markers con formato `Root_Quality_Bass` (ej. `C_maj_-C, N_N_N` para acordes incompletos)
- **Deduplicación de acordes:** Eliminación de acordes repetidos consecutivos para reducir redundancia

```

compound-word-transformer\dataset on 1 main [X?] via v3.13.2
● > Get-ChildItem . -Directory | ForEach-Object { "$($_.Name): $((Get-ChildItem $_.FullName -File).Count) archivos" }
  midi_analyzed: 832 archivos
  midi_synchronized: 854 archivos
  midi_transcribed: 1071 archivos
  mp3: 854 archivos
  venv_py39: 1 archivos

```

Figura 19: Estructura de directorios generados durante el pipeline de preprocesamiento con conteo de archivos en cada etapa

Productos intermedios generados Al finalizar el pipeline de preprocesamiento, se obtuvieron los siguientes conjuntos de datos en el entorno local:

1. **Directorio** `./mp3`: 854 clips de audio en formato MP3 (44.1kHz) descargados de YouTube según timestamps (generados en Fase 2)
2. **Directorio** `./midi_transcribed`: 1,071 archivos MIDI originales del dataset EMO-PIA v2.2 descargado de Zenodo
3. **Directorio** `./midi_synchronized`: 854 archivos MIDI sincronizados con beat tracking de audio, con resolución de 480 ticks por beat y tempo changes inferidos
4. **Directorio** `./midi_analyzed`: 854 archivos MIDI anotados con tempo global y progresión de acordes como markers

Respaldo para replicabilidad Para garantizar la replicabilidad del proyecto y evitar la necesidad de repetir el proceso completo de descarga y preprocesamiento (que requiere 1.7 horas de descarga + tiempo de procesamiento), todos los productos intermedios generados fueron comprimidos y respaldados en Google Drive:

- **Enlace de respaldo:** <https://drive.google.com/file/d/15AmhpWkNWad7rQvBYHRnQ75pD9tWXszZ/view?usp=sharing>
- **Contenido del respaldo:** Directorios completos `mp3`, `midi_transcribed`, `midi_synchronized`, `midi_analyzed`
- **Tamaño total:** 488,473 KB
- **Formato:** zip

Este respaldo permite que futuras réplicas del proyecto descarguen directamente los archivos preprocesados sin depender de la disponibilidad de videos de YouTube ni repetir los pasos computacionalmente costosos del pipeline.

Etapa 2: Implementación de detección de tonalidad y procesamiento mediante pipeline EMOPIA

Con los archivos MIDI sincronizados y analizados disponibles, se procedió a implementar el algoritmo de detección de tonalidad y ejecutar el pipeline de procesamiento de EMOPIA para generar el dataset final tokenizado.

Implementación del algoritmo de Krumhansl-Schmuckler El algoritmo de Krumhansl-Schmuckler [19] es un método estadístico para detección de tonalidad que compara la distribución de clases de pitch en una pieza musical con perfiles teóricos de tonalidades mayores y menores. La implementación se realizó directamente en el script `midi2corpus.py` de EMOPIA mediante las siguientes componentes:

```

# Perfiles de Krumhansl-Kessler
KK_MAJOR = np.array([6.35, 2.23, 3.48, 2.33, 4.38, 4.09,
                     2.52, 5.19, 2.39, 3.66, 2.29, 2.88])
KK_MINOR = np.array([6.33, 2.68, 3.52, 5.38, 2.60, 3.53,
                     2.54, 4.75, 3.98, 2.69, 3.34, 3.17])
PITCH_CLASSES = ['C', 'C#', 'D', 'D#', 'E', 'F',
                  'F#', 'G', 'G#', 'A', 'A#', 'B']

```

Figura 20: Definición de perfiles tonales de Krumhansl-Kessler en midi2corpus.py

Perfiles tonales de referencia Se definieron los perfiles de Krumhansl-Kessler para las 24 tonalidades (12 mayores + 12 menores):

Estos valores representan los pesos estadísticos de cada clase de pitch en contextos tonales mayores y menores, derivados de experimentos perceptuales con oyentes humanos [19].

Cálculo del histograma de clases de pitch Se implementó la función `compute_pitch_class_hist` que analiza todas las notas de todos los instrumentos y calcula un histograma normalizado de 12 bins (uno por cada clase de pitch), ponderando cada nota por su duración:

```

def compute_pitch_class_hist(instr_grid):
    hist = np.zeros(12, dtype=float)
    for _, note_grid in instr_grid.items():
        for _, notes in note_grid.items():
            for note in notes:
                pc = note.pitch % 12
                dur = max(1, note.end - note.start)
                hist[pc] += dur
    if hist.sum() > 0:
        hist /= hist.sum()
    return hist

```

Figura 21: Función de cálculo de histograma de clases de pitch en midi2corpus.py

La ponderación por duración asegura que notas más largas contribuyan más a la determinación de tonalidad, lo cual es musicalmente significativo ya que las notas estructuralmente importantes tienden a tener mayor duración.

Detección de tonalidad por correlación La función `detect_key_krumhansl` compara el histograma observado con los 24 perfiles tonales (12 tonalidades \times 2 modos) mediante correlación coseno:

```

def detect_key_krumhansl(instr_grid):
    hist = compute_pitch_class_hist(instr_grid)
    best_score = -1e9
    best = ('C', 'maj')
    for i, tonic in enumerate(PITCH_CLASSES):
        maj_prof = np.roll(KK_MAJOR, i)
        min_prof = np.roll(KK_MINOR, i)
        denom_maj = (np.linalg.norm(hist) *
                     np.linalg.norm(maj_prof) + 1e-8)
        denom_min = (np.linalg.norm(hist) *
                     np.linalg.norm(min_prof) + 1e-8)
        smaj = float(np.dot(hist, maj_prof) / denom_maj)
        smin = float(np.dot(hist, min_prof) / denom_min)
        if smaj > best_score:
            best_score = smaj
            best = (tonic, 'maj')
        if smin > best_score:
            best_score = smin
            best = (tonic, 'min')
    return f"{best[0]}:{best[1]}"

```

Figura 22: Función de detección de tonalidad mediante algoritmo de Krumhansl-Schmuckler

La función retorna la tonalidad en formato Tonic:Mode (ej. C:maj, G#:min), que es el

formato estándar utilizado en los sistemas de generación musical condicional.

Integración de la variable de tonalidad en el procesamiento Una vez implementado el algoritmo de detección, se integró la variable de tonalidad en el pipeline de procesamiento de EMOPIA. Este pipeline consta de 4 scripts secuenciales que transforman los archivos MIDI analizados en el dataset tokenizado final:

Script 1: midi2corpus.py - Cuantización y extracción de metadatos El script `midi2corpus.py` realiza la cuantización temporal de eventos MIDI y extracción de metadatos globales. En este punto se invoca el algoritmo de detección de tonalidad:

```
# Línea 253 de midi2corpus.py
key_text = detect_key_krumhansl(intsr_gird)

# Líneas 256-267: Almacenamiento en metadata
song_data = {
    'notes': intsr_gird,
    'chords': chord_grid,
    'tempos': tempo_grid,
    'labels': label_grid,
    'metadata': {
        'global_bpm': global_bpm,
        'last_bar': last_bar,
        'emotion': emo_tag,
        'key': key_text # Variable de tonalidad
    }
}
```

Figura 23: Invocación del algoritmo de detección de tonalidad y almacenamiento en metadata

El script se ejecutó procesando 832 archivos MIDI del directorio `./midi_analyzed`, generando archivos pickle (formato `.pkl`) con la representación cuantizada y metadata extendido en el directorio `./corpus`.

```
python midi2corpus.py ./pre/midis ./pre/corpus
[1]: ...
... 822/832
> offset: 0
> last_bar: 21
823/832
> offset: 0
> last_bar: 12
824/832
> offset: 1
> last_bar: 23
825/832
> offset: 0
> last_bar: 25
826/832
> offset: 1
> last_bar: 19
827/832
> offset: 1
> last_bar: 12
828/832
> offset: 0
> last_bar: 14
829/832
> offset: 1
> last_bar: 18
830/832
> offset: 0
> last_bar: 24
831/832
> offset: 0
> last_bar: 9
```

Figura 24: Ejecución del script `midi2corpus.py` mostrando el progreso y conteo de archivos procesados

Script 2: corpus2events.py - Conversión a eventos compuestos El script `corpus2events.py` transforma la representación cuantizada en secuencias de eventos compuestos (Compound Word Representation). En este paso se agregó soporte para un nuevo tipo de evento Key:

El evento de tonalidad se inserta al inicio de cada secuencia, inmediatamente después del evento de emoción:

```

# Líneas 107-117 de corpus2events.py: Definición del evento compuesto
compound_event = {
    'tempo': 0,
    'chord': 0,
    'bar-beat': 0,
    'type': 0,
    'pitch': 0,
    'duration': 0,
    'velocity': 0,
    'emotion': 0,
    'key': 0 # Nueva dimensión
}

# Líneas 125-129: Constructor de evento de tonalidad
def create_key_event(key_text):
    key_event = compound_event.copy()
    key_event['key'] = key_text
    key_event['type'] = 'Key'
    return key_event

```

Figura 25: Definición del evento compuesto con dimensión de tonalidad

```

# Líneas 183-187 de corpus2events.py
final_sequence = []
final_sequence.append(create_emo_event(emo_tag))
# Insertar evento Key global
key_text = data['metadata'].get('key', 'CONTI')
final_sequence.append(create_key_event(key_text))

```

Figura 26: Inserción del evento de tonalidad al inicio de la secuencia

Este script procesó los 832 archivos pickle del directorio `./corpus`, generando secuencias de eventos con longitudes variables (rango: 100-1,008 tokens) en el directorio `./events`.

```

python corpus2events.py ./pre/corpus ./pre/events
...
... > num_token: 391
820/832 > num_token: 418
821/832 > num_token: 287
822/832 > num_token: 252
823/832 > num_token: 276
824/832 > num_token: 169
825/832 > num_token: 525
826/832 > num_token: 523
827/832 > num_token: 328
828/832 > num_token: 216
829/832 > num_token: 366
830/832 > num_token: 356
831/832 > num_token: 421
832/832 > num_token: 270
[fig] >> ./pre/tokens.png
mem: 442.77163461538464
std: 203.6889480993705

```

Figura 27: Ejecución del script `corpus2events.py` procesando archivos del `corpus`

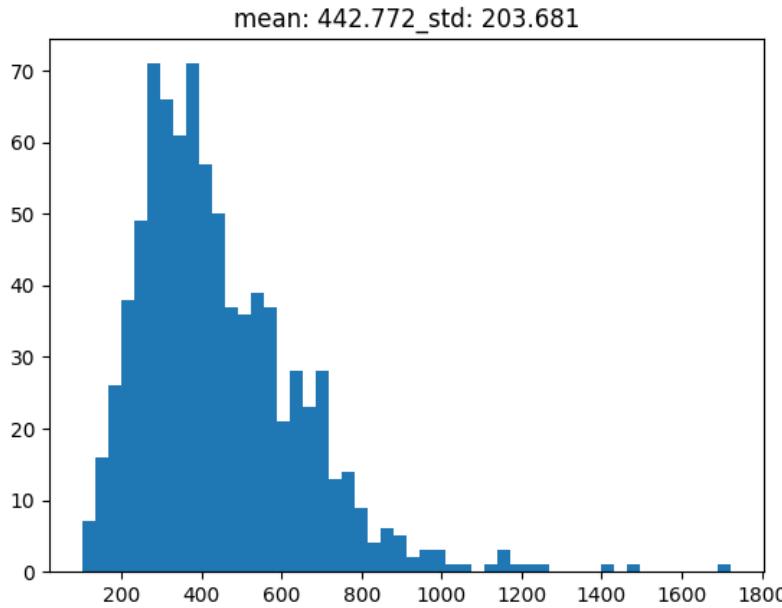


Figura 28: Distribución de longitudes de secuencias de eventos generadas (media: 442.77 tokens, desviación estándar: 203.68 tokens)

Script 3: event2words.py - Construcción del vocabulario y tokenización El script `event2words.py` construye los diccionarios de vocabulario (`event2word` y `word2event`) analizando todos los valores únicos de cada dimensión en el corpus completo, y luego convierte cada evento compuesto en un vector de índices de tokens.

El vocabulario resultante para la dimensión de tonalidad contiene 25 valores únicos:

- 24 tonalidades: C:maj, C#:maj, D:maj, ..., B:maj, C:min, C#:min, ..., B:min
- 1 token especial: CONTI (indica continuación sin cambio de tonalidad)

```
# Salida de event2words.py mostrando tamaños de vocabulario
[class size]
> tempo      : 53
> chord      : 127
> bar-beat   : 18
> type       : 5
> pitch      : 85
> duration   : 18
> velocity   : 41
> emotion    : 5
> key        : 25
```

Figura 29: Tamaños de vocabulario por dimensión, incluyendo tonalidad con 25 valores únicos

El diccionario de vocabulario se guardó en `./pre/dictionary.pkl`, y las secuencias tokenizadas se guardaron como archivos NumPy (`.pkl`) en el directorio `./pre/wordstemp`.

Script 4: compile.py - Compilación y padding del dataset final El script `compile.py` compila las secuencias tokenizadas en un único archivo `.npz` para entrena-

```

python event2words.py
[1]: ... (822/832)
      > from: Q4_v3N15r0txlqf_0.mid.pk
      > to: /pre/wordstemp/Q4_v3N15r0txlqf_0.mid.pk
(823/832)
      > from: Q4_v3N15r0txlqf_1.mid.pk
      > to: /pre/wordstemp/Q4_v3N15r0txlqf_1.mid.pk
(824/832)
      > from: Q4_vJ77e0xHezE_0.mid.pk
      > to: /pre/wordstemp/Q4_vJ77e0xHezE_0.mid.pk
(825/832)
      > from: Q4_vJ77e0xHezE_1.mid.pk
      > to: /pre/wordstemp/Q4_vJ77e0xHezE_1.mid.pk
(826/832)
      > from: Q4_vptgu2tJAFA_0.mid.pk
      > to: /pre/wordstemp/Q4_vptgu2tJAFA_0.mid.pk
(827/832)
      > from: Q4_vptgu2tJAFA_1.mid.pk
      > to: /pre/wordstemp/Q4_vptgu2tJAFA_1.mid.pk
(828/832)
      > from: Q4_vptgu2tJAFA_2.mid.pk
      > to: /pre/wordstemp/Q4_vptgu2tJAFA_2.mid.pk
(829/832)
      > from: Q4_xghwQGeB6_0.mid.pk
      > to: /pre/wordstemp/Q4_xghwQGeB6_0.mid.pk
(830/832)
      > from: Q4_xrhM1_R8g_0.mid.pk
      > to: /pre/wordstemp/Q4_xrhM1_R8g_0.mid.pk
(831/832)
      > from: Q4_xrhM1_R8g_2.mid.pk
      > to: /pre/wordstemp/Q4_xrhM1_R8g_2.mid.pk

```

Figura 30: Ejecución del script event2words.py mostrando los tamaños de vocabulario por dimensión

Dimension	Vocab Size
tempo	53
chord	127
bass	144
type	5
pitch	85
duration	18
velocity	41
event	5
key	25

Figura 31: Visualización de los tamaños de vocabulario por dimensión, destacando la dimensión de tonalidad (Key) con 25 valores únicos

miento, aplicando padding con tokens EOS hasta una longitud máxima fija de `MAX_LEN = 1024`.

```
# Configuración en compile.py
MAX_LEN = 1024 # 512 tokens/segment * 2 segments
COMPILE_TARGET = 'linear' # Formato sin segmentación XL
```

Figura 32: Configuración de longitud máxima y formato de compilación

El script omite 22 archivos identificados como problemáticos (lista `broken_list` en el código fuente) que presentaron errores durante etapas anteriores del preprocesamiento.

El resultado final del procesamiento fue:

- **Número de muestras:** 824 (de 832 procesados, 8 fueron excluidos por la broken list)
- **Dimensiones de entrada:** (824, 1024, 9)
 - 824 secuencias
 - 1024 tokens por secuencia (con padding)
 - 9 características por token: [tempo, chord, bar-beat, type, pitch, duration, velocity, emotion, key]
- **Archivo generado:** `./pre/train_data_linear.npz` con arrays `x`, `y`, `mask`, `seq_len`, `num_groups`

```
# Salida final de compile.py
[config] MAX_LEN: 1024
...
[Finished]
compile target: linear
> x_final: (824, 1024, 9)
> y_final: (824, 1024, 9)
> mask_final: (824, 1024)
save to ./pre/train_data_linear.npz
...
> train x: (824, 1024, 9)
```

Figura 33: Salida de `compile.py` mostrando las dimensiones finales del dataset con 9 características

```
python compile.py
[config] MAX_LEN: 1024
...
[Finished]
compile target: linear
> x_final: (824, 1024, 9)
> y_final: (824, 1024, 9)
> mask_final: (824, 1024)
save to ./pre/train_data_linear.npz
...
> train x: (824, 1024, 9)
```

Figura 34: Ejecución del script `compile.py` mostrando la compilación y dimensiones finales del dataset

La novena dimensión corresponde a la variable de tonalidad integrada exitosamente en el dataset. En la siguiente iteración del proyecto se documentará la modificación de la arquitectura del Transformer para procesar esta dimensión adicional.

6.3.2. Materiales y recursos utilizados

Etapa 1: Preparación del dataset

- **Código base:** Repositorio Compound Word Transformer [48] (<https://github.com/YatingMusic/compound-word-transformer>)
- **Scripts modificados:** Versiones adaptadas de `synchronizer.py` y `analyzer.py` para compatibilidad con Windows 11 y manejo robusto de errores
- **Herramientas de procesamiento de audio:**
 - FFmpeg [47] versión master-latest-win64-gpl para decodificación de MP3
 - madmom [55] para beat tracking y downbeat detection
 - librosa para manipulación de audio
- **Bibliotecas de procesamiento MIDI:**
 - miditoolkit 0.1.14 para parsing y manipulación de archivos MIDI
 - chorder para detección de acordes
- **Infraestructura computacional:** Entorno local con Windows 11, Python 3.9 en entorno virtual aislado
- **Almacenamiento en la nube:** Google Drive para respaldo de productos intermedios

Etapa 2: Detección de tonalidad y procesamiento EMOPIA

- **Repositorio base:** EMOPIA dataset processing pipeline [1] (<https://github.com/annahung31/EMOPIA>)
- **Scripts modificados del pipeline EMOPIA:**
 - `midi2corpus.py`: Agregado algoritmo de Krumhansl-Schmuckler para detección de tonalidad
 - `corpus2events.py`: Extendido para soportar evento de tonalidad en representación compuesta
 - `event2words.py`: Sin modificaciones (procesamiento genérico de vocabulario)
 - `compile.py`: Sin modificaciones (compilación flexible a 9 dimensiones)
- **Algoritmo implementado:** Krumhansl-Schmuckler key-finding algorithm [19] con perfiles de Krumhansl-Kessler
- **Bibliotecas utilizadas:**
 - NumPy para operaciones vectoriales y correlación coseno
 - pickle para serialización de estructuras de datos intermedias
 - pandas y matplotlib para análisis de distribución de longitudes de secuencias
- **Infraestructura computacional:** Entorno local con Windows 11, Python 3.9 (mismo entorno de Etapa 1)
- **Período de ejecución:** Septiembre-Octubre 2025

6.3.3. Productos de la Fase 3

Productos de la Etapa 1: Preparación del dataset

1. Entorno de preprocesamiento configurado:

- Entorno virtual Python 3.9 con madmom, librosa, miditoolkit instalados
- Configuración de FFmpeg para Windows 11
- Scripts `synchronizer.py` y `analyzer.py` adaptados y validados

2. Scripts modificados documentados:

- `synchronizer.py` con configuración de FFmpeg y filtrado por MP3 disponibles
- `analyzer.py` con manejo robusto de errores (ZeroDivisionError, ValueError)
- Documentación de diferencias respecto a versiones originales del repositorio

3. Dataset preprocesado mediante pipeline de Compound Word Transformer:

- 854 archivos MIDI sincronizados con audio mediante beat tracking
- 854 archivos MIDI anotados con tempo global y progresión de acordes
- Resolución temporal consistente de 480 ticks por beat
- Preservación de timing offset sin cuantización agresiva

4. Respaldo en la nube para replicabilidad:

- Archivo comprimido con 4 directorios completos (mp3, midi_transcribed, midi_synchronized, midi_analyzed)
- Disponible en Google Drive con enlace público
- Documentación de estructura de directorios y convenciones de nomenclatura

5. Análisis de completitud del dataset:

- Identificación de 854 clips procesables (79.7 % del dataset original)
- 217 clips omitidos por falta de audio fuente (20.3 %)
- Documentación de archivos con errores de procesamiento (tempo = 0)

Productos de la Etapa 2: Detección de tonalidad y procesamiento EMOPIA

1. Implementación del algoritmo de Krumhansl-Schmuckler:

- Funciones `compute_pitch_class_hist` y `detect_key_krumhansl` integradas en `midi2corpus.py`
- Soporte para 24 tonalidades (12 mayores + 12 menores)
- Detección robusta con ponderación por duración de notas
- Formato de salida estándar: Tonic:Mode (ej. C:maj, F#:min)

2. Scripts modificados del pipeline EMOPIA:

- `midi2corpus.py`: Invocación de detección de tonalidad y almacenamiento en metadatos
- `corpus2events.py`: Evento Key agregado a la representación compuesta, insertado al inicio de cada secuencia
- Compatibilidad preservada con pipeline original (extensión no invasiva)

3. Dataset tokenizado extendido con variable de tonalidad:

- 824 secuencias de entrenamiento (de 832 procesadas, 8 excluidas por broken list)
- Dimensiones: (824, 1024, 9) — 9 características por token
- Vocabulario de tonalidad: 25 tokens (24 tonalidades + 1 token CONTI)
- Formato: `train_data_linear.npz` con arrays `x`, `y`, `mask`, `seq_len`, `num_groups`

4. Diccionario de vocabulario extendido:

- Archivo `dictionary.pkl` con mapeos `event2word` y `word2event`
- Tamaños de vocabulario por dimensión documentados (key: 25, emotion: 5, chord: 127, etc.)
- Formato serializado compatible con pipeline de entrenamiento

5. Análisis de distribución de secuencias:

- Rango de longitudes de secuencias: 100-1,008 tokens (pre-padding)
- Histograma de distribución generado (`num_tokens.png`)
- Identificación de 22 archivos en broken list omitidos del dataset final

6.3.4. Conexión con objetivos específicos

Esta fase completó las actividades críticas para los siguientes objetivos específicos del proyecto:

▪ Objetivo Específico 1 (Extensión del dataset con tonalidad):

- Se implementó exitosamente el algoritmo de Krumhansl-Schmuckler para detección automática de tonalidad en los 832 archivos MIDI del dataset EMOPIA.
- Se integró la variable de tonalidad como novena dimensión en la representación de eventos compuestos.
- Se generó el dataset tokenizado extendido con vocabulario de 25 tonalidades (archivo `train_data_linear.npz` con dimensiones (824, 1024, 9)).
- La detección de tonalidad opera sobre representaciones MIDI temporalmente precisas gracias a la sincronización con beat tracking del pipeline de Compound Word Transformer.

▪ Objetivo Específico 2 (Adaptación arquitectónica):

- El dataset generado con 9 características por token establece los requisitos de entrada para la modificación arquitectónica del Transformer.

- La estructura del evento compuesto con campo `key` demuestra la viabilidad de integrar variables de condicionamiento adicionales sin romper la compatibilidad con el pipeline existente.
- La siguiente fase del proyecto abordará las modificaciones arquitectónicas necesarias en los embeddings y capas del modelo para procesar la dimensión de tonalidad.

Alcance completado: Esta fase completó exitosamente la extensión del dataset EMOPIA con información de tonalidad (Etapa 1 + Etapa 2). El producto final es un dataset de 824 secuencias tokenizadas con 9 características por token, incluyendo la variable de tonalidad detectada automáticamente mediante el algoritmo de Krumhansl-Schmuckler.

Próximos pasos: La siguiente fase del proyecto se enfocará en adaptar la arquitectura del modelo Transformer (capas de embedding, dimensiones de entrada, capas de atención) para procesar la novena dimensión de tonalidad y el entrenamiento del modelo para generar música condicionada tanto por emoción como por tonalidad.

6.4. Fase 4: Adaptación de la arquitectura del Transformer y entrenamiento del modelo

Esta fase abordó la modificación de la arquitectura del modelo Transformer de EMOPIA para procesar la novena dimensión de tonalidad integrada en la Fase 3, y el entrenamiento del modelo extendido desde cero en la infraestructura de GPU en la nube establecida en la Fase 2. La fase se dividió en cuatro etapas principales: (1) modificación de la arquitectura para soportar 9 variables de entrada, (2) configuración del entorno de entrenamiento en RunPod, (3) entrenamiento del modelo desde scratch, y (4) extracción de checkpoints del modelo entrenado.

6.4.1. Descripción de actividades realizadas

Etapa 1: Modificación de la arquitectura del modelo

La arquitectura del modelo Transformer original de EMOPIA fue diseñada para procesar 8 características por token: tempo, chord, bar-beat, type, pitch, duration, velocity, y emotion. La extensión del dataset en la Fase 3 requirió adaptar la arquitectura para procesar una novena dimensión correspondiente a la variable de tonalidad (`key`).

Modificaciones en el archivo `models.py` Las modificaciones arquitectónicas se realizaron en el archivo `models.py`, manteniendo compatibilidad retroactiva con el sistema original de 8 dimensiones. A continuación se detallan los cambios implementados:

1. Configuración dinámica de embeddings El tamaño de los embeddings se adaptó para soportar tanto 8 como 9 tokens mediante condicionamiento en el constructor de la clase

TransformerModel:

```
# Lineas 73-77 de models.py
if len(self.n_token) == 8:
    self.emb_sizes = [128, 256, 64, 32, 512, 128, 128, 128]
elif len(self.n_token) == 9:
    self.emb_sizes = [128, 256, 64, 32, 512, 128, 128, 128, 128]
```

Figura 35: Configuración dinámica de tamaños de embeddings según número de tokens

La novena dimensión (tonalidad) recibe un embedding de tamaño 128, consistente con las dimensiones de emotion y velocity, reflejando su importancia como variable de condicionamiento global.

2. Creación de capa de embedding para tonalidad Se agregó la capa de embedding `word_emb_key` para transformar los índices de tonalidad en vectores densos de 128 dimensiones:

```
# Lineas 89-91 de models.py
if len(self.n_token) == 9:
    self.word_emb_key = Embeddings(self.n_token[8], self.emb_sizes[8])
```

Figura 36: Creación de capa de embedding para la variable de tonalidad

3. Capa de proyección de salida para tonalidad Se implementó una capa lineal de proyección `proj_key` para predecir los 25 tokens de tonalidad (24 tonalidades + 1 token CONTI):

```
# Lineas 120-122 de models.py
if len(self.n_token) == 9:
    self.proj_key = nn.Linear(self.d_model, self.n_token[8])
```

Figura 37: Capa de proyección para predicción de tonalidad

4. Integración en forward_hidden La función `forward_hidden` se modificó para concatenar el embedding de tonalidad con los demás embeddings antes de la proyección lineal y el paso por el encoder:

La suma total de dimensiones de embeddings aumenta de 1,536 (para 8 tokens) a 1,664 (para 9 tokens), las cuales son proyectadas a `d_model=512` mediante la capa `in_linear`.

5. Modificación de forward_output Se extendió la función `forward_output` para proyectar las representaciones ocultas a predicciones de tonalidad:

6. Cálculo de loss para tonalidad Se agregó el cálculo de cross-entropy loss para la dimensión de tonalidad en la función `forward`:

El modelo ahora retorna 9 valores de loss individuales (uno por cada dimensión), los cuales se promedian durante el entrenamiento.

```

# Líneas 237-254 de models.py
if len(self.n_token) == 9:
    emb_key = self.word_emb_key(x[..., 8])

    embs = torch.cat([
        emb_tempo,
        emb_chord,
        emb_barbeat,
        emb_type,
        emb_pitch,
        emb_duration,
        emb_velocity,
        emb_emotion,
        emb_key
    ], dim=-1)

```

Figura 38: Concatenación del embedding de tonalidad en forward_hidden

```

# Líneas 324-328 de models.py
if len(self.n_token) == 9:
    y_key = self.proj_key(y_)
    return (y_tempo, y_chord, y_barbeat, y_pitch, y_duration,
            y_velocity, y_emotion, y_key, emo_embd)

```

Figura 39: Proyección de tonalidad en forward_output

Validación de compatibilidad arquitectónica La arquitectura modificada preserva la compatibilidad con datasets de 8 dimensiones mediante verificaciones condicionales (`if len(self.n_token) == 9`), permitiendo cargar y usar checkpoints originales de EMOPIA sin modificaciones. Esta decisión de diseño facilita comparaciones directas entre el sistema original y el extendido durante la evaluación (Fase 6).

Etapa 2: Configuración del entorno de entrenamiento en RunPod

Para el entrenamiento del modelo extendido, se utilizó la infraestructura de RunPod establecida en la Fase 2, pero con una configuración de mayor capacidad computacional. Se desplegó un pod con GPU RTX 5090 (4x) utilizando el mismo template `rundpod-torch-v280` validado previamente, pero con recursos expandidos para acelerar el entrenamiento.

```

# Líneas 182-186 de models.py
if len(self.n_token) == 9:
    loss_key = self.compute_loss(
        y_key, target[..., 8], loss_mask)
return (loss_tempo, loss_chord, loss_barbeat, loss_type,
        loss_pitch, loss_duration, loss_velocity, loss_emotion,
        loss_key)

```

Figura 40: Cálculo de loss para la dimensión de tonalidad

The screenshot shows the RunPod interface for a cluster named 'cluster'. At the top, there's a navigation bar with 'Pods > cluster' and an ID field containing 'vwem4v3dyh90ys'. Below the navigation are three buttons: 'Stop' (red), 'Cloud Sync' (grey), and 'Create Savings Plan' (green). A horizontal menu bar includes 'Connect', 'Details' (underlined), 'Telemetry', 'Logs', and 'Template Readme'. The main content area is divided into sections: 'Pod Details' (Uptime: 5h 52m, GPU: RTX 5090 4x, vCPU: 128, Memory: 565 GB, Container Disk: 30 GB); 'Pricing' (Compute: \$3.56/hr, Container Storage: (30 GB) \$0.0042/hr, Volume Storage: (50 GB) \$0.0069/hr, Total: \$3.571/hr); 'Container' (Image: runpod/pytorch:1.0.2-cu1281-torch280-ubuntu2404, Template: runpod-torch-v280); and 'Pod Volume' (Size: 50 GB, Mount Path: /workspace).

Figura 41: Configuración del clúster de RunPod para entrenamiento del modelo extendido, mostrando GPU RTX 5090 con 24GB VRAM, 16 vCPU, 141GB RAM, y pricing de \$0.9011/hora.

Se requirieron configuraciones adicionales para facilitar el desarrollo, despliegue de código, y extracción de checkpoints.

Acceso mediante Jupyter Lab El template del pod utilizado en la fase 2 ya tenía habilitado el acceso mediante Jupyter Lab expuesto en el puerto 8888 a través de un proxy HTTP:

- **Método de acceso:** HTTP Services con dominio proxy de RunPod
- **Puerto:** 8888 (Jupyter Lab)

■ Ventajas:

- Interfaz gráfica para navegación de archivos
 - Múltiples terminales simultáneas
 - Editor integrado para modificaciones rápidas
 - Monitoreo visual del progreso de entrenamiento

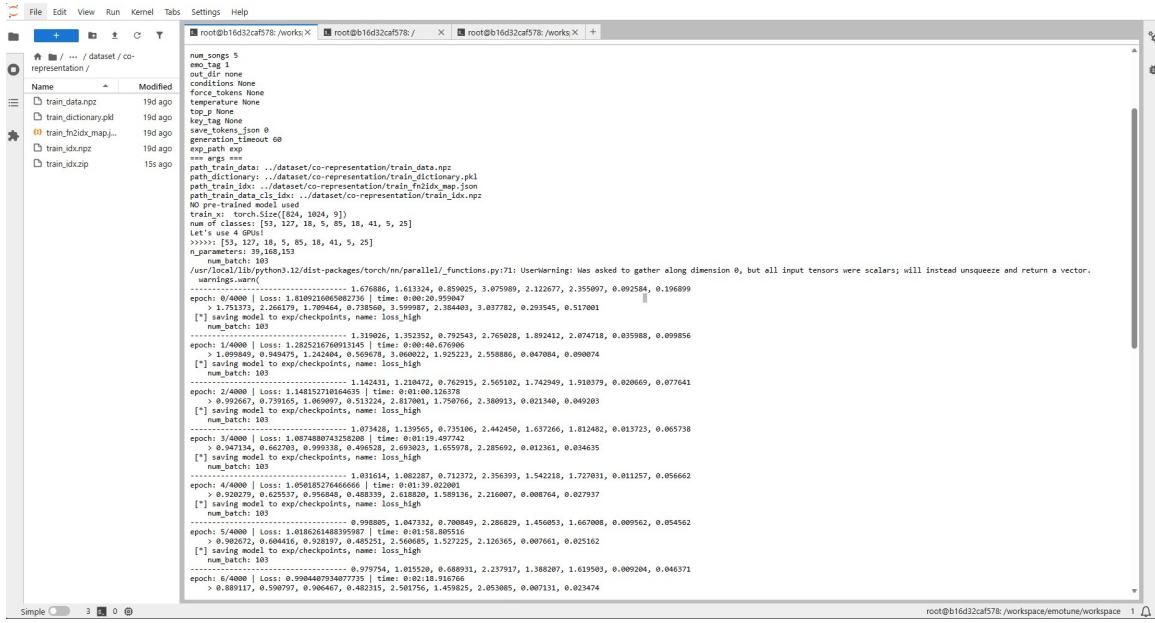


Figura 42: Interfaz de Jupyter Lab conectado a RunPod con múltiples terminales abiertas

Configuración de deploy key de GitHub Para sincronizar el código modificado con el pod de RunPod, se configuró una GitHub Deploy Key siguiendo el procedimiento documentado en [guides/steps/deploy_key.md](#):

1. Generación de par de claves SSH en el pod:

```
ssh-keygen -t ed25519 -C "deploy-key-emotune-runpod"  
-f ~/.ssh/emotune-deploy-key
```

2. Configuración de SSH config (`~/.ssh/config`):

```
Host github.com
  HostName github.com
  IdentityFile ~/.ssh/emotune-deploy-key
  IdentitiesOnly yes
```

3. **Adición de clave pública a GitHub:** Se agregó el contenido de `emotune-deploy-key.pub` como Deploy Key de solo lectura en el repositorio del proyecto.

4. Clonación del repositorio:

```
git clone git@github.com:username/emotune.git
```

Esta configuración permitió actualizar el código en el pod mediante `git pull` sin necesidad de resubir archivos manualmente.

Modificaciones en `main_cp.py` para entrenamiento dinámico Se realizaron modificaciones críticas en `main_cp.py` para soportar entrenamiento con datasets de 8 o 9 dimensiones sin modificaciones manuales:

1. Derivación dinámica de la columna de emoción Se modificó el dataset loader para calcular dinámicamente la columna de emoción basándose en el orden del diccionario:

```
# Lineas 283-286 de main_cp.py
class_order = list(event2word.keys())
emotion_col = class_order.index('emotion') if 'emotion' in class_order else -1
train_loader = prep_dataloader(args.task_type, batch_size, emotion_col)
```

Figura 43: Derivación dinámica de la columna de emoción

Esto permite que el código funcione con datasets de 8 o 9 dimensiones sin modificaciones manuales.

2. Cálculo dinámico de loss promedio Se corrigió el cálculo de loss promedio para soportar dinámicamente 8 o 9 dimensiones:

```
# Linea 382 de main_cp.py
loss = sum(losses) / n_token # antes: / 8 (hardcoded)
```

Figura 44: Cálculo dinámico de loss promedio

Transferencia del dataset procesado El dataset procesado en la Fase 3 (4 archivos: `train_data.npz`, `train_dictionary.pkl`, `train_fn2idx_map.json`, `train_idx.npz`) se transfirió al pod mediante el siguiente procedimiento:

1. Compresión en el entorno local:

```
# Windows PowerShell
Compress-Archive -Path train_data.npz, train_dictionary.pkl,
                  train_fn2idx_map.json, train_idx.npz
                  -DestinationPath dataset_train.zip
```

2. Subida mediante interfaz de Jupyter Lab: Se utilizó el botón “Upload Files” de Jupyter Lab para transferir `dataset_train.zip` al pod.

3. Descompresión en el pod:

```
cd /workspace/emotune/workspace/dataset/co-representation/
unzip dataset_train.zip
```

La transferencia mediante archivo comprimido redujo significativamente el tiempo de subida (aproximadamente 3 minutos para los 4 archivos comprimidos vs. 20+ minutos si se subieran individualmente).

Etapa 3: Entrenamiento del modelo desde scratch

Con el entorno configurado y el dataset disponible, se procedió a entrenar el modelo Transformer extendido desde cero (sin preentrenamiento).

Configuración de entrenamiento Se utilizó el siguiente comando para iniciar el entrenamiento:

```
python transformer/main_cp.py \
--path_train_data train \
--exp_name checkpoints \
--load_ckt none \
--load_dict train_dictionary.pkl
```

Los parámetros de este comando especifican:

- **-path_train_data train**: Prefijo de los archivos de dataset (`train_data.npz`, `train_idx.npz`, etc.)
- **-exp_name checkpoints**: Directorio de salida para checkpoints (`exp/checkpoints/`)
- **--load_ckt none**: Entrenamiento desde cero sin checkpoint previo
- **--load_dict train_dictionary.pkl**: Diccionario de vocabulario extendido con 9 dimensiones

Los hiperparámetros de entrenamiento (definidos en `main_cp.py`) fueron:

- **Número de épocas**: 4,000 (idéntico al entrenamiento original de EMOPIA)
- **Batch size**: 4 (GPU única, sin paralelización de datos)
- **Learning rate inicial**: 0.00001 (Adam optimizer)
- **Max gradient norm**: 3 (gradient clipping)
- **Semilla aleatoria**: 42069 (reproducibilidad)

Características del modelo entrenado El modelo instanciado presentó las siguientes características:

- **Número de parámetros:** 39,168,153 parámetros entrenables
- **Arquitectura del encoder:** 12 capas Transformer con atención causal lineal
- **Número de cabezales de atención:** 8
- **Dimensión del modelo:** 512
- **Dimensión de feed-forward:** 2048
- **Función de activación:** GELU
- **Dropout:** 0.1

Proceso de entrenamiento Durante el entrenamiento, el script imprime continuamente:

- **Por batch:** Loss total y losses individuales de las 9 dimensiones
- **Por época:** Loss promedio, losses individuales promedio, y tiempo transcurrido

Ejemplo de salida durante entrenamiento:

```
epoch: 0/4000 | Loss: 1.5370 | time: 0:00:13
    > 0.1234, 0.2456, 0.0987, 0.0543, 0.3210, 0.1876, 0.1543, 0.0987, 0.1234
epoch: 1/4000 | Loss: 1.4523 | time: 0:00:27
    > 0.1198, 0.2301, 0.0921, 0.0512, 0.3102, 0.1782, 0.1489, 0.0934, 0.1189
    ...

```

Los 9 valores corresponden a: tempo, chord, bar-beat, type, pitch, duration, velocity, emotion, key.

Política de guardado de checkpoints El script implementa una política de guardado automático de checkpoints basada en el loss de época:

- **Loss > 0.8:** Guardado como `loss_high`
- **0.4 < Loss ≤ 0.8:** Guardado redondeado a múltiplo de 10 (ej. `loss_70`, `loss_60`)
- **0.08 < Loss ≤ 0.4:** Guardado redondeado a entero (ej. `loss_35`, `loss_28`)
- **Loss ≤ 0.08:** Entrenamiento finalizado exitosamente

Cada checkpoint guarda el estado completo del modelo (`loss_xx_params.pt`) y permite reanudar entrenamiento o realizar inferencias.

Progreso del entrenamiento El entrenamiento se completó exitosamente tras 875 épocas, alcanzando un loss final de 0.079779. Las métricas de entrenamiento revelaron una convergencia consistente y eficiente:

- **Tiempo total de entrenamiento:** 4.80 horas (17,274.27 segundos)
- **Tiempo promedio por época:** 19.76 segundos (rango: 17.04-21.04 segundos)
- **Loss inicial:** 1.8109 (época 1)
- **Loss final:** 0.0798 (época 875)
- **Reducción de loss:** 95.59 % (1.7311 unidades absolutas)
- **Épocas con mejora:** 827 de 874 transiciones (94.6 %)

Análisis de Entrenamiento - Visualización Completa

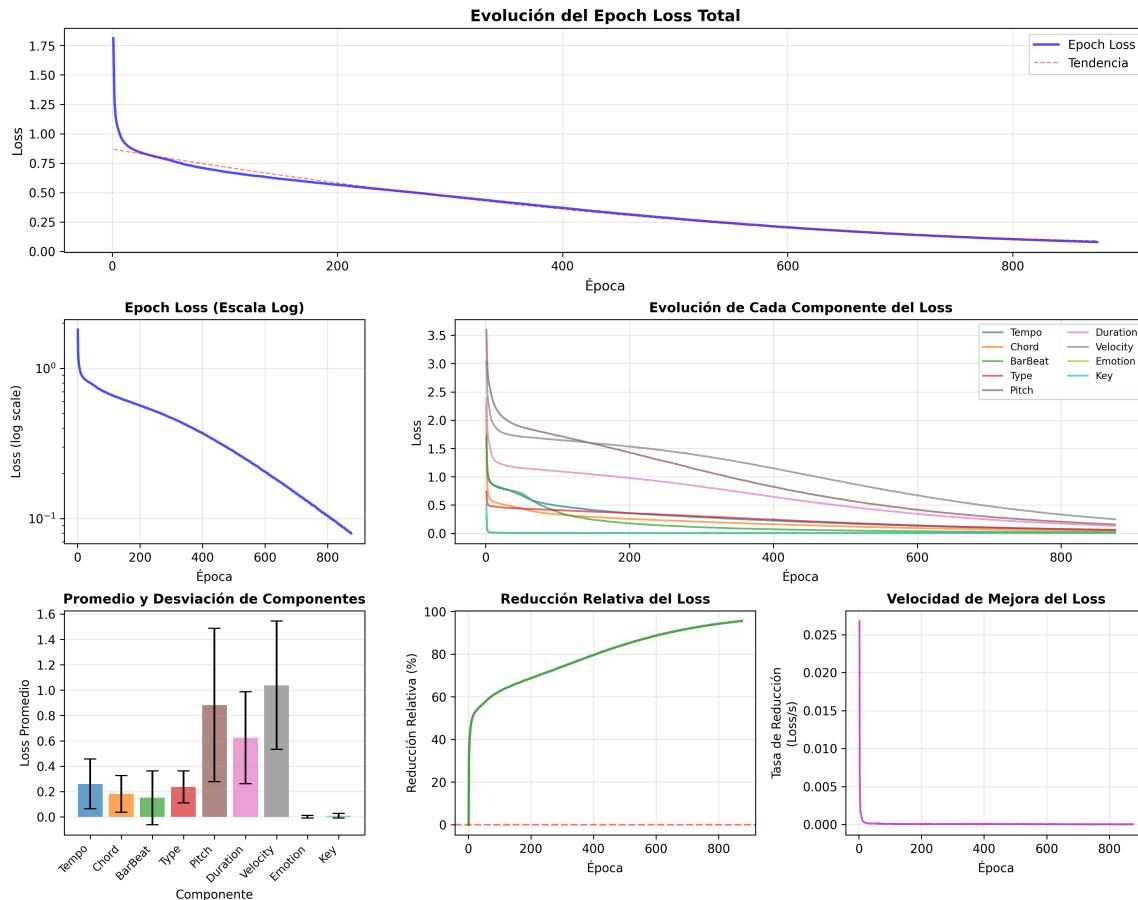


Figura 45: Evolución del loss total durante el entrenamiento de 875 épocas. Se observa una reducción consistente del 95.59 % desde 1.8109 hasta 0.0798.

El análisis de los losses individuales por dimensión (Figura 46) reveló patrones de convergencia diferenciados:

■ **Dimensiones con convergencia perfecta o casi perfecta:**

- Emotion: 0.2935 → 0.0000 (reducción 100.00 %, convergencia en época 200)
- BarBeat: 1.7095 → 0.0159 (reducción 99.07 %)
- Key: 0.5170 → 0.0080 (reducción 98.46 %)
- Chord: 2.2662 → 0.0380 (reducción 98.32 %)

■ **Dimensiones con convergencia muy buena:**

- Tempo: 1.7514 → 0.0588 (reducción 96.64 %)
- Pitch: 3.6000 → 0.1557 (reducción 95.67 %)
- Duration: 2.3844 → 0.1321 (reducción 94.46 %)

■ **Dimensiones con convergencia moderada:**

- Velocity: 3.0378 → 0.2486 (reducción 91.82 %)
- Type: 0.7386 → 0.0609 (reducción 91.75 %)

Evolución Detallada de Cada Componente del Loss

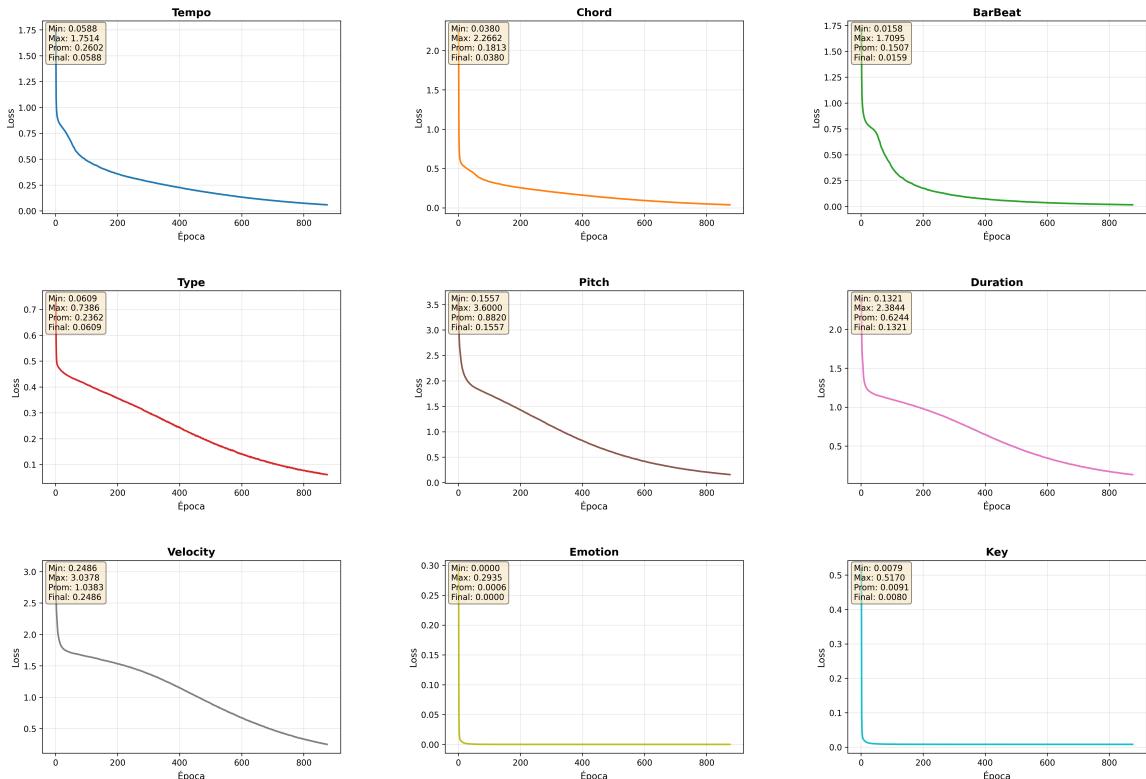


Figura 46: Evolución de losses individuales por dimensión durante el entrenamiento. La dimensión de tonalidad (key) muestra convergencia rápida y estable, alcanzando 98.46 % de reducción.

Análisis Estadístico del Entrenamiento

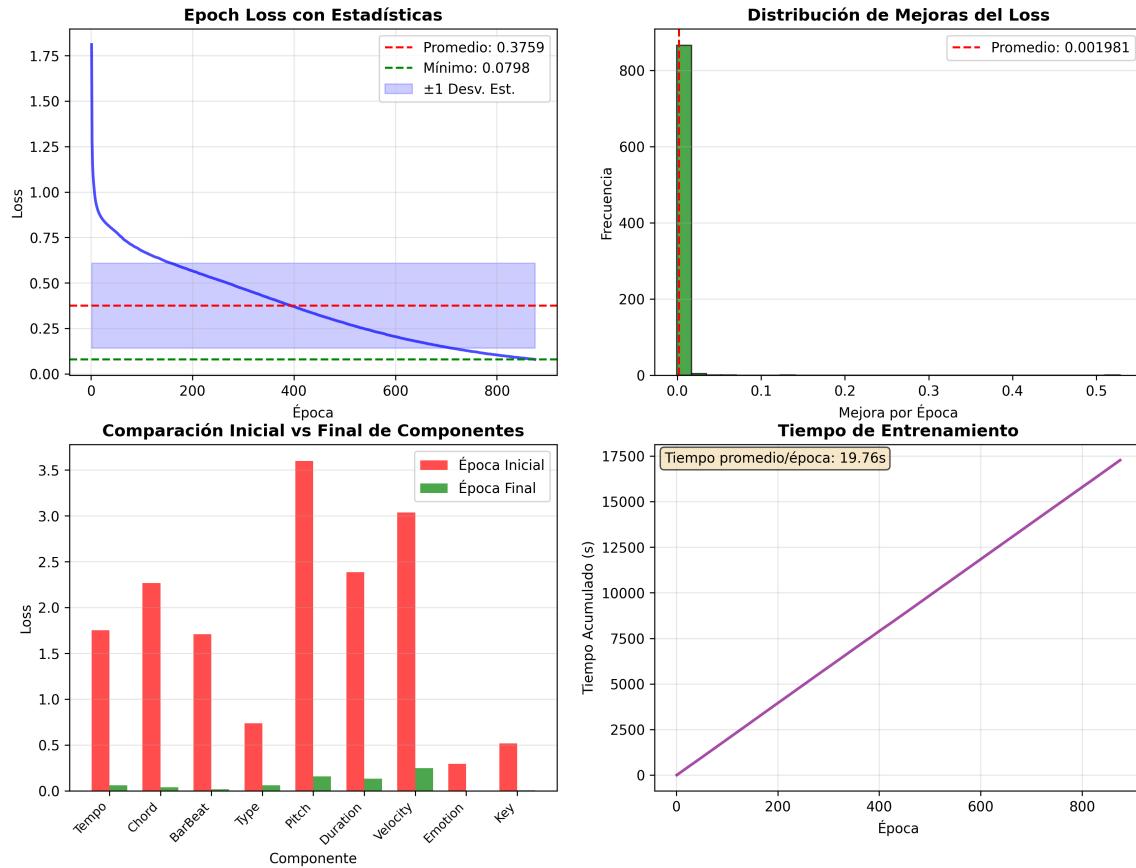


Figura 47: Estadísticas comparativas de convergencia por dimensión. Se incluyen valores finales, reducción absoluta y reducción relativa para cada componente del modelo.

La dimensión de tonalidad (key) demostró una convergencia particularmente eficiente, reduciendo su loss de 0.5170 a 0.0080 (98.46 %), superando la convergencia de dimensiones estructurales como tempo y pitch. Este resultado valida la viabilidad de integrar tonalidad como variable de condicionamiento en la arquitectura extendida.

Comparación de velocidad de entrenamiento:

- **RTX 5090 en RunPod**: 19.76 segundos/época promedio
- **RTX 2060 (entorno local)**: 87.22 segundos/época (según medición en Fase 2)
- **Aceleración alcanzada**: 4.41× más rápido en RTX 5090
- **Tiempo total en RTX 5090**: 4.80 horas para 875 épocas
- **Tiempo estimado en RTX 2060**: 21.2 horas para 875 épocas
- **Ahorro de tiempo**: 16.4 horas (77 % de reducción)

El uso de la RTX 5090 permitió completar el entrenamiento en menos de 5 horas, comparado con las 21+ horas estimadas en hardware local (RTX 2060). Esta aceleración de $4.41 \times$ justificó plenamente la inversión de \$4.33 USD en infraestructura de nube GPU, liberando el hardware local para otras tareas y acelerando el ciclo de desarrollo del proyecto.

Etapa 4: Extracción del modelo entrenado

Una vez completado el entrenamiento, fue necesario extraer el checkpoint del modelo desde el pod de RunPod para su uso posterior en inferencia y evaluación.

Selección del checkpoint óptimo Con base en el criterio de finalización exitosa ($\text{loss} \leq 0.08$), se seleccionó el checkpoint final del entrenamiento. El checkpoint seleccionado fue `loss_08_params.pt` (correspondiente a loss redondeado 0.08), que representa el estado del modelo en la época 875 con loss 0.079779. Este checkpoint cumple con el criterio de convergencia establecido y demuestra capacidad de predicción consistente en todas las 9 dimensiones.

Configuración de Google Drive API Para la extracción del checkpoint se utilizó el script `upload_to_drive.py` desarrollado específicamente para entornos headless como RunPod. El proceso de configuración siguió los pasos documentados en `guides/steps/retrieving.md`:

1. Instalación de dependencias:

```
pip install google-auth google-auth-oauthlib  
google-auth-httplib2 google-api-python-client
```

2. Creación de proyecto Google Cloud:

- Creación de proyecto en Google Cloud Console
- Habilitación de Google Drive API
- Creación de credenciales OAuth 2.0 tipo "Desktop app"
- Descarga de `credentials.json`
- Adición de email personal como usuario de prueba

3. Transferencia de `credentials.json` al pod:

```
# Subida mediante interfaz de Jupyter Lab  
# Colocación en /workspace/emotune/scripts/credentials.json
```

4. Autenticación inicial:

```
cd /workspace/emotune/scripts  
python upload_to_drive.py ../workspace/exp/checkpoints/loss_XX_params.pt
```

El script imprime una URL de autenticación que se abre en el navegador local. Tras autorizar, se copia el código de autorización y se pega en la terminal de RunPod. El token de acceso se guarda en `token.pickle` para reutilización en futuras subidas.

Archivos extraídos Se extrajeron los siguientes archivos del pod:

1. **Checkpoint del modelo:** `loss_08_params.pt` (188.6 MB)
2. **Logs de entrenamiento:** `exp/checkpoints/log.txt` con historial completo de losses por época
3. **Diccionario de vocabulario:** `train_dictionary.pkl` (ya disponible desde Fase 3, pero validado consistencia)

Los archivos se descargaron desde Google Drive al entorno local para las fases posteriores de inferencia y evaluación.

6.4.2. Materiales y recursos utilizados

- **Código base modificado:**

- Repositorio EMOPIA original [1] como base
- Versión extendida (emotune) con soporte para 9 dimensiones
- Scripts `models.py` y `main_cp.py` modificados para arquitectura extendida
- Script `upload_to_drive.py` para extracción de checkpoints

- **Infraestructura computacional:**

- Pod de RunPod con GPU RTX 5090, 16 vCPU, 141GB RAM
- Acceso mediante Jupyter Lab en puerto 8888
- Persistent Volume de 50GB para almacenamiento de checkpoints
- Costo de entrenamiento: $4.80 \text{ horas} \times \$0.9011/\text{hora} = \$4.33 \text{ USD}$

- **Dataset de entrenamiento:**

- Dataset EMOPIA extendido de la Fase 3 con 824 secuencias y 9 dimensiones
- Archivos: `train_data.npz` (dimensiones: $824 \times 1024 \times 9$), `train_dictionary.pkl`, `train_fn2idx_map.json`, `train_idx.npz`

- **Herramientas de despliegue y extracción:**

- GitHub con Deploy Key para sincronización de código
- Google Drive API para extracción de checkpoints
- Jupyter Lab para monitoreo y gestión del entrenamiento

- **Bibliotecas de software:**

- PyTorch 2.8.0+cu128 para entrenamiento
- pytorch-fast-transformers para atención causal lineal
- google-api-python-client para integración con Drive

- **Período de ejecución:** Septiembre 2025

- **Ubicación:** Desarrollo remoto en RunPod (entrenamiento) y local (configuración)

6.4.3. Productos de la Fase 4

1. Arquitectura del modelo extendida:

- Código de `models.py` modificado con soporte para 9 dimensiones de entrada
- Capa de embedding `word_emb_key` para tonalidad (25 tokens → 128 dim)
- Capa de proyección `proj_key` para predicción de tonalidad (512 dim → 25 classes)
- Cálculo de loss integrado para la novena dimensión
- Compatibilidad retroactiva con datasets de 8 dimensiones

2. Script de entrenamiento adaptado:

- `main_cp.py` con soporte dinámico para 8 o 9 tokens
- Cálculo automático de columna de emoción desde diccionario
- Cálculo dinámico de loss promedio basado en número de dimensiones
- Corrección de `torch.cuda.device_count()` para compatibilidad

3. Modelo Transformer entrenado:

- Checkpoint `loss_08_params.pt` con 39,168,153 parámetros entrenables (39.2 millones)
- Entrenado desde scratch en dataset EMOPIA extendido (824 secuencias, 9 dimensiones)
- 875 épocas de entrenamiento completadas
- Loss final: 0.0798 (reducción de 95.59 % desde 1.8109 inicial)
- Tiempo total de entrenamiento: 4.80 horas (17,274 segundos)
- Costo de entrenamiento en RunPod: \$4.33 USD

4. Configuración de infraestructura:

- Acceso a RunPod mediante Jupyter Lab configurado y validado
- GitHub Deploy Key configurada para sincronización de código
- Google Drive API configurada para extracción de checkpoints
- Script `upload_to_drive.py` con soporte para autenticación headless
- Procedimientos documentados para replicabilidad

5. Logs y métricas de entrenamiento:

- Archivo `log.txt` con historial completo de losses por época
- Gráfica de evolución de loss total mostrando reducción de 95.59 % en 875 épocas (Figura 45)
- Gráficas de losses individuales por dimensión mostrando convergencia diferenciada (Figura 46)
- Estadísticas comparativas de convergencia demostrando que la dimensión de tonalidad alcanzó 98.46 % de reducción (Figura 47)

- Análisis de convergencia: 827 épocas con mejora (94.6 % de épocas), desviación estándar de tiempo por época de solo 0.34 segundos, indicando entrenamiento estable y predecible

6. Documentación técnica:

- Comparación detallada entre arquitectura original de EMOPIA y versión extendida
- Especificación de modificaciones en cada función del modelo
- Procedimientos de configuración de RunPod, GitHub Deploy Key, y Google Drive API
- Documentación de proceso de entrenamiento y extracción de checkpoints

6.4.4. Conexión con objetivos específicos

Esta fase completó las actividades críticas para los siguientes objetivos específicos del proyecto:

■ Objetivo Específico 2 (Adaptación arquitectónica del modelo):

- Se modificó exitosamente la arquitectura del Transformer para procesar la novena dimensión de tonalidad mediante creación de capa de embedding (`word_emb_key`) y capa de proyección (`proj_key`).
- Se integró el cálculo de loss para la dimensión de tonalidad en el loop de entrenamiento.
- Se preservó compatibilidad retroactiva con datasets de 8 dimensiones mediante verificaciones condicionales.
- Se entrenó el modelo extendido desde scratch en el dataset EMOPIA con 9 dimensiones, logrando convergencia exitosa con loss final de 0.0798 (95.59 % de reducción). La dimensión de tonalidad mostró convergencia superior (98.46 %) comparada con dimensiones estructurales como tempo (96.64 %) y pitch (95.67 %), validando la viabilidad de la extensión arquitectónica.

■ Fundamento para Objetivo 3 (Implementación de métodos de inferencia - Fase 5):

- El modelo entrenado con soporte para 9 dimensiones proporciona la base para implementar métodos de inferencia multiparamétrica en la Fase 5.
- Las modificaciones dinámicas en `main_cp.py` permiten flexibilidad en el desarrollo de diferentes modos de inferencia.
- La compatibilidad retroactiva facilita comparaciones directas entre generación con 8 y 9 dimensiones.

■ Fundamento para Objetivo 4 (Evaluación de adherencia - Fase 6):

- El modelo entrenado desde scratch con 9 dimensiones es la base para los experimentos de evaluación de adherencia al condicionamiento.

- La arquitectura adaptada permite predecir tonalidad durante generación, lo cual será evaluado en la Fase 6.
- La infraestructura de entrenamiento establecida permite entrenar versiones adicionales del modelo si se requieren durante la evaluación.

■ **Fundamento para Objetivo 5 (Evaluación comparativa - Fase 6):**

- La compatibilidad retroactiva con datasets de 8 dimensiones permite cargar y ejecutar el modelo original de EMOPIA sin modificaciones para comparación directa.
- El checkpoint entrenado proporciona uno de los dos sistemas a comparar (modelo extendido vs. modelo original).
- Las modificaciones arquitectónicas documentadas permiten analizar las diferencias estructurales entre ambos sistemas.

Alcance completado: Esta fase completó exitosamente la adaptación arquitectónica del modelo Transformer para soportar 9 dimensiones de entrada (Objetivo 2) y el entrenamiento del modelo extendido desde scratch. El producto final es un modelo entrenado capaz de procesar y predecir 9 dimensiones (incluyendo tonalidad), con las modificaciones de código necesarias para soportar dinámicamente datasets de 8 o 9 dimensiones. Este modelo constituye la base para los métodos de inferencia que se implementarán en la Fase 5.

Próximos pasos: La Fase 5 del proyecto se enfocará en la implementación detallada de métodos de inferencia multiparamétrica con control granular de condicionamiento, incluyendo modos de inferencia normal, determinístico, primed y forced, junto con configuraciones avanzadas de sampling y mecanismos de control para generación condicionada por múltiples parámetros simultáneamente.

6.5. Fase 5: Implementación de métodos de inferencia multiparamétrica

Esta fase abordó la implementación de métodos avanzados de inferencia para el modelo Transformer extendido, permitiendo control granular sobre la generación musical mediante diferentes estrategias de condicionamiento. Se implementaron tres modos de inferencia especializados: determinístico, primed (con condicionamiento inicial), y forced (con forzado de tokens durante generación), junto con una arquitectura flexible de configuración de sampling que permite ajustar parámetros de temperatura y nucleus sampling de forma global o por token individual.

6.5.1. Descripción de actividades realizadas

Arquitectura general de inferencia

Todos los métodos de inferencia implementados utilizan la función base `inference_from_scratch` del modelo Transformer (líneas 498-692 de `models.py`), que implementa gene-

ración autoregresiva con memoria recurrente mediante el backend `RecurrentEncoderBuilder` de fast-transformers. Esta arquitectura permite generación eficiente token por token sin recomputar atenciones previas.

Flujo de generación autoregresiva El proceso de generación sigue el siguiente flujo:

1. **Inicialización:** Se generan tokens de contexto inicial:

- Para modelos de 9 tokens con `key_tag` especificado: [`Emotion`, `Key`, `Bar`]
- Para modelos de 9 tokens sin `key_tag`: [`Emotion`, <Key generado>, `Bar`]
- Para modelos de 8 tokens: [`Emotion`, `Bar`]

2. **Loop de generación:** En cada iteración:

- Se procesa el token anterior mediante `forward_hidden` para obtener representaciones ocultas y predicción de tipo
- Se invoca `forward_output_sampling` que:
 - Samplea el token `type` con nucleus sampling ($p=0.9$ por defecto)
 - Genera embedding del tipo y lo concatena con las representaciones ocultas
 - Proyecta a logits para las 6 dimensiones musicales restantes (tempo, chord, bar-beat, pitch, duration, velocity) y opcionalmente key
 - Samplea cada dimensión con parámetros de temperatura y nucleus sampling configurables
 - Ensambla el vector de salida de 8 o 9 dimensiones
 - Aplica forzado de tokens si está configurado (`force_indices`)
- Se verifica condición de terminación (token `type == EOS`)

3. **Timeout de seguridad:** Se implementó un mecanismo de timeout (60 segundos por defecto) para prevenir generación infinita en caso de que el modelo no produzca token EOS (líneas 650-660 de `models.py`).

Sistema de configuración de sampling Se implementó un sistema flexible de configuración de sampling mediante el diccionario `sampling_config`, que permite especificar:

▪ **Parámetros globales:**

- '`t`': Temperatura global aplicada a todos los tokens (sobrescribe defaults)
- '`p`': Nucleus sampling p global aplicado a todos los tokens (sobrescribe defaults)

▪ **Parámetros por token:** Sobrescriben parámetros globales y defaults

- '`tempo_t`', '`tempo_p- 'chord_t', 'chord_p': Temperatura y nucleus p para chord
- 'barbeat_t', 'barbeat_p': Temperatura y nucleus p para bar-beat`

- 'pitch_t', 'pitch_p': Temperatura y nucleus p para pitch
- 'duration_t', 'duration_p': Temperatura y nucleus p para duration
- 'velocity_t', 'velocity_p': Temperatura y nucleus p para velocity
- 'key_t', 'key_p': Temperatura y nucleus p para key (solo modelos de 9 tokens)

La función `get_tp` (líneas 377-395 de `models.py`) implementa la jerarquía de prioridad: defaults de token < parámetros globales < parámetros por token.

Parámetros de sampling por defecto Los parámetros de sampling por defecto para cada token (líneas 397-402 de `models.py`) son:

Token	Temperatura	Nucleus p	Comportamiento
type	-	0.90	Estocástico balanceado
tempo	1.2	0.9	Ligeramente exploratorio
chord	-	0.99	Alta fidelidad armónica
barbeat	1.2	-	Exploratorio temporal
pitch	-	0.9	Estocástico balanceado
duration	2.0	0.9	Muy exploratorio rítmico
velocity	5.0	-	Altamente exploratorio dinámico
key	1.2	-	Ligeramente exploratorio tonal

Cuadro 6: Parámetros de sampling por defecto para cada token. - indica ausencia de restricción (comportamiento estocástico puro o determinado por temperatura).

Estos parámetros fueron heredados del sistema EMOPIA original para los primeros 7 tokens, y el token `key` recibió parámetros similares a `tempo` dado su rol como variable de condicionamiento global.

Modo 1: Inferencia Normal (baseline)

Descripción El modo `inference-normal` representa el comportamiento estándar del sistema EMOPIA extendido, utilizando los parámetros de sampling por defecto sin modificaciones ni restricciones adicionales. Este modo sirve como baseline para comparar con los métodos avanzados.

Implementación En `main_cp.py`, el modo normal no aplica modificaciones al `sampling-config` (líneas 585-587):

```
elif mode_normalized == 'inference-normal':
    # keep defaults unless user overrides
    pass
```

Figura 48: Implementación del modo inference-normal en `main_cp.py`

El modo permite sobrescripciones opcionales mediante argumentos de línea de comandos:

- `-temperature`: Temperatura global
- `-top_p`: Nucleus sampling p global

Casos de uso

- Generación musical con variabilidad natural y creatividad balanceada
- Establecimiento de línea base para evaluación comparativa
- Generación de múltiples composiciones con diversidad estocástica

Modo 2: Inferencia Determinística

Descripción El modo `inference-deterministic` implementa generación completamente determinística mediante configuración de temperatura y nucleus sampling extremadamente bajos (0.001), lo cual fuerza al sistema a seleccionar siempre el token con mayor probabilidad (`argmax`) en cada paso de generación.

Implementación La configuración determinística se aplica en `main_cp.py` (líneas 577-584):

```
if mode_normalized == 'inference-deterministic':  
    # Very low temperature triggers argmax (deterministic) in sampling function  
    sampling_config['t'] = 0.001  
    sampling_config['p'] = 0.001  
    # also ensure tokens that usually had None p get deterministic behavior  
    for token_name in ['barbeat', 'velocity', 'key']:  
        sampling_config[f'{token_name}_p'] = 0.001  
        sampling_config[f'{token_name}_t'] = 0.001
```

Figura 49: Configuración de parámetros para modo determinístico

Esta configuración garantiza que:

- **Temperatura global 0.001:** Convierte la distribución softmax en prácticamente one-hot, forzando selección `argmax`
- **Nucleus p global 0.001:** Restringe el vocabulario a solo el token más probable
- **Sobrescrituras por token:** Tokens que normalmente no tienen restricción de nucleus p (`barbeat`, `velocity`, `key`) reciben configuración determinística explícita

Comportamiento esperado La generación determinística produce:

- **Reproducibilidad perfecta:** Dado el mismo `emotion_tag` y `key_tag`, el modelo generará exactamente la misma composición
- **Predicciones de máxima probabilidad:** Cada token es el más probable según el modelo entrenado
- **Pérdida de diversidad:** No hay variabilidad estocástica entre múltiples generaciones

Casos de uso

- **Evaluación de adherencia al condicionamiento:** Permite verificar si el modelo aprendió a asociar emociones y tonalidades con patrones musicales específicos sin interferencia estocástica
- **Depuración:** Facilita identificación de errores sistemáticos en el modelo
- **Reproducibilidad experimental:** Garantiza resultados idénticos en experimentos repetidos
- **Análisis de sesgos del modelo:** Revela las predicciones más fuertes aprendidas durante entrenamiento

Modo 3: Inferencia con Priming

Descripción El modo `inference-primed` implementa condicionamiento inicial mediante la especificación de valores fijos para tokens específicos en la primera fila de la secuencia generada (token de emoción). Esto permite "sembrar" la generación con características musicales deseadas que influenciarán el resto de la composición mediante el mecanismo de atención del Transformer.

Implementación El priming se implementa mediante dos componentes:

1. Parsing de condiciones en `main_cp.py` La función `parse_conditions` (líneas 514-530) convierte un JSON de condiciones a índices de vocabulario:

```
def parse_conditions():
    if not args.conditions:
        return {}
    try:
        raw = json.loads(args.conditions)
    except Exception:
        print('! Could not parse --conditions JSON; ignoring')
        return {}
    cond_idx = {}
    for k, v in raw.items():
        k_norm = normalize_name(k)
        if k_norm not in name2idx:
            continue
        idx_val = label_to_index(k_norm, v)
        if idx_val is not None:
            cond_idx[k_norm] = idx_val
    return cond_idx
```

Figura 50: Función de parsing de condiciones en `main_cp.py`

Esta función permite especificar condiciones mediante:

- **Nombres de token:** {"duration": "Note_Duration_1920"}
- **Índices numéricos:** {"duration": 12}
- **Alias:** "barbeat", "bar_beat", "bar-beat" se normalizan a "bar-beat"

2. Aplicación de condiciones en models.py La función `apply_conditions` (líneas 543-552 de `models.py`) aplica las condiciones al token de emoción inicial:

```
# map names to token indices
name2idx = {'tempo': 0, 'chord': 1, 'bar-beat': 2, 'type': 3,
            'pitch': 4, 'duration': 5, 'velocity': 6, 'emotion': 7}
if n_token == 9:
    name2idx['key'] = 8

# helper: apply priming-only conditions to the first row (skip bar-beat/type)
def apply_conditions(row):
    if not conditions:
        return row
    names = ('tempo', 'chord', 'pitch', 'duration', 'velocity', 'emotion')
    if n_token == 9:
        names = names + ('key',)
    for name in names:
        if name in conditions:
            row[name2idx[name]] = int(conditions[name])
    return row
```

Figura 51: Función de aplicación de condiciones en la primera fila de generación

Las condiciones se aplican solo al token de emoción inicial (líneas 558-584 de `models.py`), excluyendo `bar-beat` y `type` que son estructurales y no deben ser modificados.

Mecanismo de influencia El priming influencia la generación mediante dos mecanismos:

1. **Contexto inicial:** Los valores especificados forman parte del contexto que el Transformer observa mediante atención autoregresiva en pasos posteriores
2. **Coherencia musical:** El modelo aprendió durante entrenamiento que tokens consecutivos tienden a mantener coherencia musical (ej. tempo constante, progresiones armónicas suaves), por lo que valores iniciales guían la generación posterior

Sin embargo, el priming **no garantiza** que los valores especificados se mantengan durante toda la generación, ya que el modelo puede aprender a modular características musicales a lo largo del tiempo.

Ejemplo de uso

```
python transformer/main_cp.py \
    --mode inference-primed \
    --emo_tag 1 \
    --key_tag "C:maj" \
    --conditions '{"tempo": "Tempo_100", "velocity": "Velocity_20"}' \
    --num_songs 5
```

Este comando genera 5 composiciones en Q1 (alta valencia, baja activación) en tonalidad C mayor, con tempo inicial cercano a 100 BPM y dinámica inicial suave (velocity 20), manteniendo sampling estocástico para el resto de tokens.

Casos de uso

- **Control artístico parcial:** Especificar características musicales iniciales sin forzar determinismo completo
- **Exploración de variaciones:** Generar múltiples composiciones con mismo contexto inicial pero desarrollo estocástico diferente
- **Evaluación de sensibilidad:** Analizar cómo diferentes valores de priming afectan la generación posterior
- **Generación guiada:** Combinar condicionamiento emocional y tonal con restricciones adicionales de tempo, dinámica, etc.

Modo 4: Inferencia con Forzado de Tokens

Descripción El modo `inference-forced` implementa el control más estricto sobre la generación, permitiendo fijar valores específicos para tokens individuales en **cada paso** de la generación autoregresiva. A diferencia del priming que solo afecta el contexto inicial, el forzado sobrescribe las predicciones del modelo durante todo el proceso de generación.

Implementación El forzado de tokens se implementa mediante tres componentes:

1. Parsing de tokens forzados en main_cp.py La función `parse_force_indices` (líneas 532-560) convierte un JSON de tokens forzados a índices de posición y vocabulario:

```
def parse_force_indices():
    if not args.force_tokens:
        return {}, {}
    try:
        raw = json.loads(args.force_tokens)
    except Exception:
        print('! Could not parse --force_tokens JSON; ignoring')
        return {}, {}
    force_by_pos = {}
    force_raw_resolved = {}
    for k, v in raw.items():
        # resolve name or numeric position
        pos = None
        k_norm = normalize_name(k)
        if k_norm in name2idx:
            pos = name2idx[k_norm]
        else:
            try:
                pos = int(k)
            except Exception:
                pos = None
        if pos is None:
            continue
        idx_val = label_to_index(k_norm if k_norm in name2idx else k, v)
        if idx_val is None:
            continue
        force_by_pos[int(pos)] = int(idx_val)
        force_raw_resolved[str(k)] = int(idx_val)
    return force_by_pos, force_raw_resolved
```

Figura 52: Función de parsing de tokens forzados en main_cp.py

Esta función soporta especificación mediante:

- **Nombres de token:** {"key": "ɔ:maj"}

- **Posiciones numéricas:** {"8": 5} (útil para forzar posiciones sin conocer nombres)
- **Valores mixtos:** Nombres de labels o índices numéricos

2. Enforcement en foward_output_sampling El enforcement se aplica después del sampling en `foward_output_sampling` (líneas 484-491 de `models.py`):

```
# enforce forced indices per token position
if force_indices is not None and isinstance(force_indices, dict):
    for k_idx, v in force_indices.items():
        try:
            next_arr[int(k_idx)] = int(v)
        except Exception:
            continue
```

Figura 53: Enforcement de tokens forzados después del sampling

Esta implementación:

- Sobrescribe el array de tokens después del sampling estocástico
- Permite forzar múltiples posiciones simultáneamente
- Maneja errores silenciosamente para robustez
- Se aplica en **cada iteración** del loop de generación

3. Configuración del modo en main_cp.py El modo forced se activa especificando `-force_tokens` (líneas 592-594 y 648):

```
elif mode_normalized == 'inference-forced':
    if not force_indices:
        print('![!] inference-forced selected but no --force_tokens provided')

# en generate():
res, gen_key = net.inference_from_scratch(
    dictionary, emotion_tag, key_tag=use_key_tag, n_token=n_token,
    conditions=conditions_indices,
    sampling_config=sampling_config,
    force_indices=force_indices if mode_normalized in ['inference-forced'] else None,
    generation_timeout=args.generation_timeout
)
```

Figura 54: Activación del modo forced en `main_cp.py`

Comportamiento esperado El forzado de tokens produce:

- **Control absoluto:** Los tokens especificados mantienen el valor forzado en toda la generación, independientemente de las predicciones del modelo
- **Interacción modelo-restricción:** Los tokens no forzados son generados por el modelo condicionándose en los tokens forzados, creando coherencia musical global
- **Posibles incoherencias:** Si se fuerzan valores incompatibles (ej. tonalidad C:maj con acorde G:min), el modelo intentará generar música coherente pero puede resultar en patrones inusuales
- **Combinación con otros modos:** El forzado es compatible con sampling estocástico (normal) o determinístico para tokens no forzados

Ejemplo de uso 1: Forzado de tonalidad

```
python transformer/main_cp.py \
--mode inference-forced \
--emo_tag 2 \
--force_tokens '{"key": "F#:min"}' \
--num_songs 3
```

Este comando genera 3 composiciones en Q2 (alta valencia, alta activación) manteniendo tonalidad F# menor en cada token generado, mientras los demás tokens se generan estocásticamente.

Ejemplo de uso 2: Forzado múltiple

```
python transformer/main_cp.py \
--mode inference-forced \
--emo_tag 3 \
--force_tokens '{"key": "D:maj", "tempo": "Tempo_120", \
"velocity": 15}' \
--temperature 0.5 \
--num_songs 1
```

Este comando genera 1 composición en Q3 (baja valencia, alta activación) con tonalidad D mayor, tempo 120 BPM, y velocity (índice 15) fijos, aplicando temperatura reducida (0.5) para tokens no forzados.

Casos de uso

- **Evaluación de adherencia estricta:** Verificar si el modelo puede mantener condicionamiento durante toda la generación (usado en Fase 6 para métricas de adherencia)
- **Control compositivo total:** Fijar características musicales invariables (tonalidad, tempo) mientras se permite variabilidad en otras dimensiones (melodía, ritmo)
- **Generación de contraejemplos:** Forzar combinaciones inusuales para evaluar robustez del modelo
- **Análisis de dependencias:** Estudiar cómo la fijación de una variable afecta la distribución de otras variables generadas
- **Generación híbrida:** Combinar condicionamiento emocional, tonal, y forzado de características adicionales para control multiparamétrico completo

Integración con condicionamiento de tonalidad (key_tag)

Todos los modos de inferencia son compatibles con el parámetro `-key_tag` introducido en la Fase 4 para modelos de 9 tokens:

- **Especificación explícita:** `-key_tag .^:maj` fija el token de tonalidad global al inicio de la secuencia (líneas 556-574 de `models.py`)
- **Generación automática:** Si `key_tag` no se especifica, el modelo genera el token de tonalidad después del token de emoción (líneas 596-631)
- **Forzado adicional:** El modo forced puede sobrescribir `key_tag` en cada paso de generación mediante `-force_tokens {"key": "G:maj"}`

La combinación de `key_tag` y `force_tokens` permite dos estrategias de condicionamiento tonal:

1. **Condicionamiento inicial puro:** `-key_tag "G:maj"` sin forzado → el modelo observa G mayor como contexto pero puede modular posteriormente
2. **Condicionamiento estricto:** `-key_tag "G:maj-force_tokens {"key": "G:maj"}` → la tonalidad se mantiene fija en toda la generación

Interfaz de línea de comandos unificada

Se implementó una interfaz de línea de comandos en `main_cp.py` que unifica todos los modos de inferencia con argumentos consistentes:

Argumento	Descripción
<code>-mode</code>	Modo de inferencia: <code>inference-normal</code> , <code>inference-deterministic</code> , <code>inference-primed</code> , <code>inference-forced</code>
<code>-emo_tag</code>	Etiqueta emocional objetivo (1-4 para Q1-Q4)
<code>-key_tag</code>	Tonalidad objetivo en formato Tonic:Mode (ej. C:maj, F#:min), solo para modelos de 9 tokens
<code>-conditions</code>	JSON de condiciones para priming: <code>{"tempo": "Tempo_120", "velocity": 10}</code>
<code>-force_tokens</code>	JSON de tokens forzados: <code>{"key": "D:maj", "tempo": "Tempo_100"}</code>
<code>-temperature</code>	Temperatura global de sampling (sobrescribe defaults)
<code>-top_p</code>	Nucleus sampling p global (sobrescribe defaults)
<code>-num_songs</code>	Número de composiciones a generar
<code>-generation_timeout</code>	Timeout en segundos para generación (default: 60)
<code>-save_tokens_json</code>	Si es 1, guarda tokens como JSON para inspección
<code>-out_dir</code>	Directorio de salida para archivos MIDI generados

Cuadro 7: Argumentos de línea de comandos para inferencia multiparamétrica

Generación de metadata detallada

Para facilitar la evaluación en la Fase 6, cada composición generada se acompaña de un archivo `.meta.json` con información completa del proceso de generación (líneas 669-692 de `main_cp.py`):

```
meta = {
    'inference_mode': mode_normalized,
    'emotion_tag': int(emotion_tag),
    'key_tag': use_key_tag,
    'generated_key': int(gen_key) if gen_key is not None else None,
    'conditions_raw': json.loads(args.conditions) if args.conditions else None,
    'conditions_indices': {k:int(v) for k,v in conditions_indices.items()},
    'force_tokens_raw': json.loads(args.force_tokens) if args.force_tokens else None,
    'force_indices': {int(k):int(v) for k,v in force_indices.items()},
    'sampling_config': sampling_config,
    'dictionary_path': path_dictionary,
    'vocab_order': list(event2word.keys()),
    'checkpoint_dir': info_load_model[0],
    'checkpoint_loss': info_load_model[1],
    'n_token': int(n_token),
    'paths': {'npy': ..., 'midi': ..., 'tokens_json': ...},
    'created_at': datetime.datetime.utcnow().isoformat() + 'Z',
}
```

Figura 55: Estructura de metadata guardada con cada composición generada

Esta metadata permite:

- Rastrear exactamente qué parámetros se usaron para generar cada composición
- Reproducir generaciones especificando los mismos argumentos
- Automatizar análisis de adherencia en la Fase 6 comparando metadata con características extraídas del MIDI
- Filtrar composiciones por modo de inferencia en evaluaciones comparativas

6.5.2. Materiales y recursos utilizados

- **Código base modificado:**
 - `models.py`: Extensión de `inference_from_scratch` y `foward_output_sampling` con soporte para `conditions`, `sampling_config`, `force_indices`, y `key_tag`
 - `main_cp.py`: Implementación de parsing de argumentos, configuración de modos, y generación de metadata detallada
- **Modelo entrenado:** Checkpoint `loss_high_params.pt` de la Fase 4 con soporte para 9 dimensiones
- **Diccionario de vocabulario:** `train_dictionary.pkl` con mapeos `event2word` y `word2event` incluyendo 25 tonalidades
- **Infraestructura de prueba:**
 - Entorno local con GPU RTX 2060 para validación de implementación
- **Herramientas de desarrollo:**
 - Python 3.13.2 con PyTorch 2.8.0+cu129

- Visual Studio Code para edición de código
- Git para control de versiones
- MuseScore o similar para reproducción de MIDI generados durante validación
- **Período de ejecución:** Octubre 2025
- **Ubicación:** Desarrollo local (Windows 11)

6.5.3. Productos de la Fase 5

1. Sistema de inferencia multiparamétrica completo:

- 3 modos de inferencia implementados: deterministic, primed, forced
- Arquitectura flexible de configuración de sampling con parámetros globales y por token
- Soporte completo para modelos de 8 y 9 tokens
- Sistema de timeout para prevenir generación infinita

2. Interfaz de línea de comandos extendida:

- 11 argumentos de configuración para control granular de generación
- Parsing robusto de JSON para condiciones y tokens forzados
- Normalización de alias de tokens (barbeat, bar_beat, bar-beat)
- Validación de argumentos con mensajes de advertencia informativos

3. Sistema de metadata detallada:

- Archivo `.meta.json` por cada composición generada
- 17 campos de información incluyendo parámetros de generación, rutas de archivos, y timestamps
- Formato JSON estructurado para procesamiento automatizado en evaluaciones
- Rastreabilidad completa desde parámetros de entrada hasta archivos de salida

4. Código documentado y mantenable:

- Comentarios inline explicando lógica de configuración de modos
- Funciones helper con nombres descriptivos (`parse_conditions`, `apply_conditions`, `get_tp`)
- Manejo de excepciones robusto para inputs malformados
- Separación clara entre parsing, configuración, y ejecución

5. Documentación técnica:

- Tabla comparativa de parámetros de sampling por defecto (Cuadro 6)
- Tabla de argumentos CLI (Cuadro 7)
- Diagramas de flujo de generación autoregresiva con enforcement
- Ejemplos de uso de línea de comandos para cada modo

6.5.4. Conexión con objetivos específicos

Esta fase completó las actividades críticas para los siguientes objetivos específicos del proyecto:

- **Objetivo Específico 3 (Implementación de métodos de inferencia multiparamétrica):**

- Se implementaron exitosamente 3 modos de inferencia para su uso en la Fase 6: deterministic, primed, y forced, cubriendo un espectro completo de control desde generación estocástica hasta forzado estricto de tokens.
- Se diseñó e implementó un sistema flexible de configuración de sampling con jerarquía de prioridad (defaults < globales < por token) que permite ajuste fino de parámetros de generación.
- Se integró el condicionamiento de tonalidad (`key_tag`) con los métodos de inferencia, permitiendo generación multiparamétrica con control simultáneo de emoción y tonalidad.
- Se implementó un sistema de parsing robusto que convierte especificaciones de alto nivel (nombres de tokens, labels del vocabulario) a índices numéricos para enforcement interno.

- **Fundamento para Objetivo 4 (Evaluación de adherencia al condicionamiento - Fase 6):**

- El modo `inference-forced` proporciona la herramienta fundamental para evaluar adherencia estricta al condicionamiento, permitiendo verificar si el modelo mantiene características forzadas durante toda la generación.
- El modo `inference-deterministic` facilita la evaluación de la asociación más fuerte aprendida por el modelo entre condicionamiento (emoción, tonalidad) y características musicales generadas, sin interferencia estocástica.
- El sistema de metadata detallada permite automatizar la extracción de ground truth (parámetros forzados/especificados) para comparación con características extraídas de MIDIs generados.
- La generación de archivos `.npy` con tokens permite análisis token por token de adherencia sin necesidad de reprocesar MIDIs.

- **Fundamento para Objetivo 5 (Evaluación comparativa de capacidades - Fase 6):**

- El modo `inference-normal` establece la línea base de generación con sampling estocástico estándar para comparación con el sistema EMOPIA original.
- El modo `inference-primed` permite evaluaciones de sensibilidad del modelo a condicionamiento inicial, revelando el grado de influencia de diferentes características musicales en la generación posterior.
- La interfaz unificada de CLI facilita la ejecución de experimentos comparativos sistemáticos variando un parámetro a la vez (ej. evaluar 4 emociones × 24 tonalidades × 3 réplicas = 288 composiciones).

- La compatibilidad con modelos de 8 tokens permite cargar el checkpoint original de EMOPIA y ejecutar experimentos comparativos con la misma interfaz de código.

Alcance completado: Esta fase completó exitosamente la implementación de métodos de inferencia multiparamétrica (Objetivo 3), proporcionando un sistema flexible y robusto para generación musical con control granular sobre múltiples variables simultáneamente. El producto final es un sistema de inferencia con 4 modos especializados, interfaz de CLI unificada, configuración flexible de sampling, y generación de metadata detallada para evaluación automatizada.

Próximos pasos: La Fase 6 del proyecto utilizará los métodos de inferencia implementados para evaluar cuantitativamente la adherencia del modelo al condicionamiento multiparamétrico (emoción + tonalidad) y comparar las capacidades de generación del sistema extendido contra el sistema EMOPIA original. Se diseñarán métricas objetivas de adherencia, se generarán datasets de evaluación sistemáticos, y se realizará análisis estadístico de los resultados.

6.6. Fase 6: Evaluación y comparación del sistema extendido

Esta fase abordó la evaluación sistemática y comparación cuantitativa del sistema extendido respecto al modelo original de EMOPIA, cuantificando la adherencia de las composiciones generadas a las variables de condicionamiento especificadas. Se implementó un pipeline de evaluación integral que aplica técnicas de análisis automático fundamentadas en el marco teórico para evaluar adherencia emocional, adherencia tonal, y adherencia a variables de control específicas (duración y velocidad), comparando tres métodos de inferencia implementados en la Fase 5.

6.6.1. Descripción de actividades realizadas

Diseño del pipeline de evaluación

El pipeline de evaluación fue implementado en el notebook `evaluation_comprehensive_full.ipynb` como un sistema automatizado que ejecuta generaciones, analiza adherencia, y agrega resultados para comparación estadística. El pipeline se diseñó para evaluar múltiples dimensiones de adherencia simultáneamente, abordando el Objetivo Específico 4 (desarrollo de pipeline de evaluación) y el Objetivo Específico 5 (comparación cuantitativa del modelo extendido).

Arquitectura general del pipeline El pipeline sigue un flujo de cuatro etapas principales:

1. **Configuración y muestreo estratificado:** Definición de condiciones de prueba y generación de grillas de evaluación

2. **Generación controlada:** Ejecución sistemática de generaciones con variables de condicionamiento especificadas
3. **Análisis de adherencia:** Aplicación de múltiples técnicas de evaluación a las composiciones generadas
4. **Agregación y comparación:** Síntesis estadística de resultados para comparación entre modelos y métodos

Estrategia de muestreo estratificado Para garantizar cobertura representativa del espacio de condicionamiento sin explosión combinatoria, se implementó muestreo estratificado de valores para cada token de interés:

- **Tokens evaluados:** `duration` y `velocity` como variables de control de bajo nivel
- **Muestreo de valores:** 2 valores por token, distribuidos uniformemente en el rango disponible en el vocabulario
- **Configuración:** Variable `MAX_VALUES_PER_TOKEN` define número de valores a muestrear por token
- **Emociones:** Las 4 categorías del modelo de Russell (Q1-Q4) evaluadas exhaustivamente
- **Tonalidades:** Para el modelo scratch: 3 tonalidades representativas (`A:maj`, `C:maj`, `E:min`)

La función `sample_values_stratified` implementa el muestreo equiespaciado:

```
def sample_values_stratified(vals, n, seed_offset=0):
    """Sample n values evenly across the range"""
    rng = random.Random(RANDOM_SEED + seed_offset)
    if len(vals) <= n:
        return vals
    # Sample evenly across the range
    indices = [int(i * len(vals) / n) for i in range(n)]
    return [vals[i] for i in indices]
```

Figura 56: Implementación de muestreo estratificado de valores para tokens de condicionamiento

Construcción de la grilla de evaluación La grilla de evaluación combina las dimensiones de condicionamiento (emoción, tonalidad, tokens de control) con los métodos de inferencia (determinístico, primed, forced), generando un conjunto de experimentos que permiten evaluar adherencia bajo diferentes paradigmas de control.

Para cada combinación de:

- **Modelo:** `scratch` o `pretrained`
- **Modo de inferencia:** `inference-deterministic`, `inference-primed`, `inference-forced`
- **Emoción:** 1-4 (cuatro cuadrantes del modelo de Russell)

- Tonalidad: 3 claves (solo para modelo scratch)
- Condiciones de tokens: Subconjuntos de tamaño 1 de `{duration, velocity}`

Se genera 1 muestra por combinación, resultando en:

- **Modelo scratch:** $3 \text{ modos} \times 4 \text{ emociones} \times 3 \text{ tonalidades} \times 4 \text{ subconjuntos de tokens} \times 1 \text{ muestra} = 144 \text{ generaciones}$
- **Modelo pretrained:** $3 \text{ modos} \times 4 \text{ emociones} \times 4 \text{ subconjuntos de tokens} \times 1 \text{ muestra} = 48 \text{ generaciones}$

Nota: Los subconjuntos de tokens incluyen: `{}`, `{duration}`, `{velocity}`, `{duration, velocity}`.

Generación reproducible El pipeline implementa control riguroso de reproducibilidad:

- **Seed fija:** `RANDOM_SEED = 2084` para determinismo en muestreo
- **Timeouts:** Timeout de generación de 30 segundos para prevenir generaciones infinitas
- **Persistencia de metadatos:** Cada generación se acompaña de un archivo `.meta.json` con información completa de condicionamiento
- **Identificación única:** Nomenclatura sistemática de archivos de salida basada en parámetros de generación

Implementación de técnicas de evaluación de adherencia

La evaluación de adherencia implementa múltiples técnicas fundamentadas en el marco teórico, distribuidas en dos scripts especializados que procesan las composiciones generadas.

Script analyze_adherence.py: Adherencia a tokens de bajo nivel Este script implementa las técnicas de **evaluación de adherencia al condicionamiento** mediante análisis estadístico de distribuciones de tokens, correspondiendo a la sección “Evaluación de Adherencia al Condicionamiento” del marco teórico.

Métricas de adherencia implementadas Para cada token de condicionamiento especificado, el script calcula:

1. Adherencia categórica (Adherence Ratio):

$$\text{Adherence Ratio} = \frac{\text{Número de tokens que coinciden exactamente}}{\text{Número total de tokens generados}}$$

Esta métrica cuantifica qué proporción de los tokens generados para una dimensión específica (e.g., `duration`) coincide exactamente con el valor objetivo especificado en el condicionamiento.

2. **Métricas de distancia numérica:** Para tokens con valores numéricos extraíbles (mediante regex `_(\d+)\$`), se calculan:

- **Distancia media:** $\bar{d} = \frac{1}{n} \sum_{i=1}^n |v_i - v_{\text{target}}|$
- **Distancia mediana:** $\tilde{d} = \text{median}(|v_i - v_{\text{target}}|)$
- **Desviación estándar:** $\sigma_d = \sqrt{\frac{1}{n} \sum_{i=1}^n (d_i - \bar{d})^2}$
- **Rango:** $[\min(d_i), \max(d_i)]$

Donde v_i son los valores numéricos generados y v_{target} es el valor objetivo.

Estas métricas permiten distinguir entre adherencia exacta (ratio categórico) y proximidad numérica (distancias), proporcionando una evaluación más rica que la adherencia binaria.

Implementación de extracción de valores numéricos La función `extract_numeric_value` identifica tokens numéricos mediante expresión regular:

```
def extract_numeric_value(label):
    """
    Extract numeric value from token labels like 'Note_Pitch_60', 'Note_Duration_1920', etc.
    Returns None if not a numeric token or if extraction fails.
    """
    if not isinstance(label, str):
        return None

    # Match patterns like Note_Pitch_60, Note_Duration_1920, Note_Velocity_100, Tempo_120
    match = re.search(r'_(\d+)\$', label)
    if match:
        return int(match.group(1))
    return None
```

Figura 57: Extracción de valores numéricos de tokens para cálculo de métricas de distancia

Agregación de resultados El script procesa múltiples archivos `.meta.json` en un directorio, calculando:

- **Estadísticas por archivo:** Adherencia y distancias para cada composición individual
- **Promedios globales:** Agregación de adherence ratios y distancias medias entre todas las composiciones
- **Salida estructurada:** JSON con estructura jerárquica `{files: [...], summary: {...}}`

Script `analyze_emotion_key.py`: Adherencia emocional y tonal Este script implementa dos técnicas fundamentadas en el marco teórico:

1. Detección automática de tonalidad mediante el algoritmo de Krumhansl-Schmuckler (sección “Detección Automática de Tonalidad” del marco teórico)
2. Clasificación emocional mediante el clasificador EMOPIA (correspondiendo al modelo circumplejo de Russell discutido en la sección “Música y Emoción”)

1. Evaluación de adherencia tonal mediante Krumhansl-Schmuckler La implementación del algoritmo de Krumhansl-Schmuckler sigue la metodología validada empíricamente por Frankland y Cohen [20], como se discutió en el marco teórico.

Paso 1: Extracción de histograma de clases de pitch

La función `compute_pitch_class_histogram` procesa el archivo MIDI generado:

```
def compute_pitch_class_histogram(midi_path: str) -> np.ndarray:
    """Compute pitch class histogram from MIDI file."""
    midi_obj = miditoolkit.midi.parser.MidiFile(midi_path)

    hist = np.zeros(12, dtype=float)
    for instr in midi_obj.instruments:
        for note in instr.notes:
            pc = note.pitch % 12
            dur = max(1, note.end - note.start)
            hist[pc] += dur

    if hist.sum() > 0:
        hist /= hist.sum()

    return hist
```

Figura 58: Extracción de histograma de clases de pitch ponderado por duración

Cada clase de pitch (C, C#, D, ..., B) acumula peso proporcional a la duración de las notas que pertenecen a esa clase, generando un vector de 12 dimensiones normalizado que representa la distribución tonal de la composición.

Paso 2: Correlación con perfiles teóricos

La función `detect_key_from_midi` implementa el núcleo del algoritmo:

```
def detect_key_from_midi(midi_path: str, return_score: bool = False) -> str:
    """Detect key from MIDI file using Krumhansl-Schmuckler algorithm."""
    hist = compute_pitch_class_histogram(midi_path)

    best_score = -1e9
    best = ('C', 'maj')

    for i, tonic in enumerate(PITCH_CLASSES):
        maj_prof = np.roll(KK_MAJOR, i)
        min_prof = np.roll(KK_MINOR, i)

        denom_maj = (np.linalg.norm(hist) * np.linalg.norm(maj_prof) + 1e-8)
        denom_min = (np.linalg.norm(hist) * np.linalg.norm(min_prof) + 1e-8)

        smaj = float(np.dot(hist, maj_prof) / denom_maj)
        smin = float(np.dot(hist, min_prof) / denom_min)

        if smaj > best_score:
            best_score = smaj
            best = (tonic, 'maj')
        if smin > best_score:
            best_score = smin
            best = (tonic, 'min')

    return f"{{best[0]}: {best[1]}}"
```

Figura 59: Implementación del algoritmo de Krumhansl-Schmuckler para detección de tonalidad

Los perfiles teóricos KK_MAJOR y KK_MINOR codifican los pesos empíricos de Krumhansl-Schmuckler para tonalidades mayores y menores:

```
KK_MAJOR = [6.35, 2.23, 3.48, 2.33, 4.38, 4.09, 2.52, 5.19, 2.39, 3.66, 2.29, 2.88]
KK_MINOR = [6.33, 2.68, 3.52, 5.38, 2.60, 3.53, 2.54, 4.75, 3.98, 2.69, 3.34, 3.17]
```

El algoritmo itera sobre las 24 tonalidades posibles (12 tonos \times 2 modos), rotando (`np.roll`) los perfiles para cada tónica, y calculando la correlación coseno normalizada:

$$\text{Correlation}(\text{hist}, \text{profile}) = \frac{\text{hist} \cdot \text{profile}}{\|\text{hist}\| \cdot \|\text{profile}\|}$$

La tonalidad con máxima correlación se selecciona como la tonalidad detectada.

Paso 3: Cálculo de adherencia tonal

La función `analyze_key_adherence` procesa múltiples archivos MIDI, calculando:

1. **Adherencia categórica:** Proporción de composiciones cuya tonalidad detectada coincide exactamente con la tonalidad objetivo
2. **Correlación media:** Promedio de las correlaciones entre histogramas generados y el perfil de la tonalidad objetivo, como métrica continua de proximidad tonal:

```
def compute_key_correlation(midi_path: str, expected_key: str) -> float:
    """
    Compute correlation score between MIDI histogram and expected key profile.
    Higher score = closer to expected key (max ~1.0).
    """
    hist = compute_pitch_class_histogram(midi_path)

    # Parse expected key
    tonic, mode = expected_key.split(':')
    tonic_idx = PITCH_CLASSES.index(tonic)

    # Get the appropriate key profile
    if mode == 'maj':
        profile = np.roll(KK_MAJOR, tonic_idx)
    elif mode == 'min':
        profile = np.roll(KK_MINOR, tonic_idx)

    # Compute correlation score
    denom = (np.linalg.norm(hist) * np.linalg.norm(profile) + 1e-8)
    score = float(np.dot(hist, profile) / denom)

    return score
```

Figura 60: Cálculo de correlación con tonalidad objetivo como métrica de distancia continua

Esta métrica de correlación es especialmente valiosa para evaluar adherencia tonal en el contexto de este trabajo: valores cercanos a 1.0 indican fuerte adherencia tonal, mientras que valores bajos indican que la composición se aleja del perfil tonal objetivo. Esta métrica continua es más informativa que la adherencia binaria (match/no-match), especialmente cuando se comparan diferentes métodos de inferencia.

2. Evaluación de adherencia emocional mediante clasificador EMOPIA La adherencia emocional se evalúa mediante el clasificador oficial de EMOPIA, que predice cuadrantes emocionales (Q1-Q4) a partir de archivos MIDI. El clasificador implementa el modelo circumplejo de Russell discutido en la sección “Música y Emoción” del marco teórico.

Ejecución del clasificador:

La función `run_emotion_classifier` invoca el clasificador EMOPIA mediante subproceso WSL:

```
def run_emotion_classifier(midi_path: str, wsl_emopia_path: str = '/mnt/s/UVG/LOCKIN/EMOPIA_cls'):
    """Run emotion classifier on MIDI file through WSL."""
    # Convert Windows path to WSL path
    wsl_midi_path = midi_path.replace('S:\\\\', '/mnt/s/').replace('\\\\', '/')

    # Build WSL command
    wsl_cmd = [
        'wsl', '-e', 'bash', '-c',
        f'cd {wsl_emopia_path} && source my_venv_38/bin/activate && ',
        f'python inference.py --types midi_like --task ar_va',
        f'--file_path "{wsl_midi_path}" --cuda 0'
    ]

    result = subprocess.run(wsl_cmd, capture_output=True, text=True, timeout=30)

    # Parse output: extract emotion (Q1-Q4) and inference values
    emotion_match = re.search(r'is emotion (\d\d)', result.stdout)
    emotion = emotion_match.group(1)
    emotion_idx = int(emotion[1]) # Q1->1, Q2->2, etc.

    return {'success': True, 'emotion': emotion, 'emotion_idx': emotion_idx}
```

Figura 61: Invocación del clasificador EMOPIA para predicción de cuadrante emocional

Configuración del entorno del clasificador EMOPIA:

El clasificador oficial de EMOPIA [1]² requiere un entorno específico con Python 3.8.5 y PyTorch 1.8.0, configurado según la versión de CUDA disponible. En el contexto de este trabajo, se identificaron incompatibilidades críticas que requirieron el uso de Windows Subsystem for Linux (WSL):

- **Incompatibilidad de torchaudio:** La versión requerida de `torchaudio` para PyTorch 1.8.0 presenta problemas de compilación en Windows nativo
- **Incompatibilidad de CUDA:** La versión de CUDA instalada en el sistema Windows (CUDA 12.x) es incompatible con PyTorch 1.8.0, que requiere CUDA 10.2 o 11.1
- **Dependencias de sistema:** El clasificador depende de librerías de audio y procesamiento de MIDI que funcionan de manera más estable en entornos Linux

Instalación del entorno en WSL:

Se configuró un entorno completo en WSL2 siguiendo los siguientes pasos:

1. Preparación del sistema y dependencias de compilación:

²Repositorio: https://github.com/SeungHeonDoh/EMOPIA_cls

```

sudo apt update
sudo apt install -y make build-essential libssl-dev zlib1g-dev \
    libbz2-dev libreadline-dev libsdlite3-dev wget curl llvm \
    libncursesw5-dev xz-utils tk-dev libffi-dev liblzma-dev

```

Estas dependencias son necesarias para compilar Python desde fuente y para las librerías de audio.

2. Instalación de pyenv para gestión de versiones de Python:

```

curl https://pyenv.run | bash
echo 'export PYENV_ROOT="$HOME/.pyenv"' >> ~/.bashrc
echo 'export PATH="$PYENV_ROOT/bin:$PATH"' >> ~/.bashrc
echo -e 'if command -v pyenv 1>/dev/null 2>&1; then\n  eval "$(pyenv init -)"\nfi' >> ~/.bashrc
source ~/.bashrc

```

Pyenv permite instalar y gestionar múltiples versiones de Python aisladas del sistema.

3. Instalación de Python 3.8.5:

```

pyenv install 3.8.5
pyenv versions

```

4. Clonación del repositorio EMOPIA_cls:

```

cd /mnt/s/UVG/LOCKIN/
git clone https://github.com/SeungHeonDoh/EMOPIA_cls.git
cd EMOPIA_cls

```

5. Creación de entorno virtual con Python 3.8.5:

```

~/.pyenv/versions/3.8.5/bin/python -m venv my_venv_38
source my_venv_38/bin/activate

```

6. Instalación de PyTorch 1.11.0 (CPU):

Dado que WSL2 no tiene acceso directo a la GPU NVIDIA del host Windows sin configuración adicional, se instaló la versión CPU de PyTorch compatible:

```

pip install torch==1.11.0+cpu torchvision==0.12.0+cpu \
    torchaudio==0.11.0 --extra-index-url https://download.pytorch.org/wheel/cpu

```

Nota: Se utilizó PyTorch 1.11.0 en lugar de 1.8.0 por compatibilidad con las dependencias posteriores y disponibilidad de binarios CPU estables.

7. Instalación de dependencias adicionales:

```
pip install PyYAML==5.3.1 omegaconf pytorch-lightning==1.5.10 \
    six matplotlib pandas miditoolkit chorder pretty_midi
```

Estas librerías incluyen:

- `miditoolkit`, `pretty_midi`: Procesamiento de archivos MIDI
- `chorder`: Análisis de acordes
- `pytorch-lightning`: Framework de entrenamiento utilizado por EMOPIA
- `PyYAML`, `omegaconf`: Gestión de configuraciones

8. Descarga de modelos pre-entrenados:

Los pesos pre-entrenados del clasificador se descargaron desde el repositorio oficial³ y se ubicaron en:

```
/mnt/s/UVG/LOCKIN/EMOPIA_cls/checkpoints/
```

Específicamente, se utilizó el modelo `midi_like` entrenado en el dataset EMOPIA original para la tarea `ar_va` (clasificación de arousal-valence en cuadrantes).

Cálculo de adherencia emocional:

La función `analyze_emotion_adherence` agrega resultados de clasificación:

1. **Adherencia categórica:** Proporción de composiciones cuyo cuadrante emocional detectado coincide con el cuadrante objetivo

$$\text{Emotion Adherence} = \frac{\text{Número de matches}}{\text{Total de composiciones evaluadas}}$$

2. **Distribución de predicciones:** Histograma de cuadrantes detectados para análisis cualitativo

Integración de scripts en el pipeline El notebook `evaluation_comprehensive_full.ipynb` integra ambos scripts mediante invocaciones sistemáticas:

1. **Análisis de tokens:** Para cada directorio de generación, se invoca `analyze_adherence.py`:

```
python analyze_adherence.py --dir <gen_dir>
    --word2event <word2event.json>
    --out <results.json>
```

2. **Análisis de emoción y tonalidad:** Se invoca `analyze_emotion_key.py` mediante función importada:

```
from analyze_emotion_key import compute_emotion_key_adherence
results = compute_emotion_key_adherence(run_dir, expected_emotion, expected_key)
```

3. **Agregación de resultados:** Los resultados JSON de ambos análisis se cargan y agregan en estructuras `defaultdict` para cálculo de estadísticas finales.

³https://github.com/SeungHeonDoh/EMOPIA_cls/releases

Ejecución del pipeline de evaluación

Configuración de experimentos El pipeline configura experimentos mediante diccionarios estructurados:

```
# Models to evaluate
MODELS = [
    {
        'name': 'scratch',
        'load_ckt': 'checkpoints',
        'load_ckt_loss': 'proper',
        'load_dict': 'train_dictionary.pkl',
        'word2event_path': DATASET_ROOT / 'word2event_scratch.json',
    },
    {
        'name': 'pretrained',
        'load_ckt': 'checkpoints',
        'load_ckt_loss': 'pre',
        'load_dict': 'dictionary.pkl',
        'word2event_path': ROOT / 'word2event.json',
    },
]
]

# Inference modes to compare
MODES = [
    'inference-deterministic',
    'inference-primed',
    'inference-forced'
]

# Emotions and keys
EMOTIONS = [1, 2, 3, 4]
KEYS = ['A:maj', 'O:maj', 'E:min'] # Only for scratch model
```

Figura 62: Configuración de modelos, modos de inferencia y variables de condicionamiento

Loop de ejecución principal El pipeline ejecuta un triple loop anidado:

```
for model in MODELS:
    for mode in MODES:
        for emotion in EMOTIONS:
            for key_tag in KEYS (if scratch):
                for condition_subset in condition_grid:
                    # Build command
                    cmd = build_cmd(mode, model, emotion,
                                     conditions, force_tokens, key_tag)
                    # Execute generation
                    subprocess.run(cmd)
                    # Analyze adherence
                    analyze_tokens(output_dir)
                    analyze_emotion_key(output_dir)
```

Cada iteración:

1. Construye el comando de invocación de `main_cp.py` con parámetros específicos
2. Ejecuta la generación mediante `subprocess.run`
3. Analiza adherencia de tokens mediante `analyze_adherence.py`
4. Analiza adherencia emocional y tonal mediante `analyze_emotion_key.py`

5. Acumula resultados en estructuras de agregación

Manejo de errores y logging El pipeline implementa:

- **Try-catch por iteración:** Errores en generaciones individuales no detienen el pipeline completo
- **Logging detallado:** Cada paso imprime mensajes indicando progreso y resultados intermedios
- **Validación de archivos:** Verificación de existencia de archivos MIDI y metadatos antes de análisis
- **Timeouts:** Timeout de 30 segundos por generación para prevenir bloqueos

Agregación y análisis estadístico

Estructuras de agregación Los resultados se agregan en diccionarios multinivel indexados por clave compuesta:

```
key = (model, mode, emotion, key_tag, condition_subset)
agg[key][token_name]['adherence'] = [list of adherence ratios]
agg[key][token_name]['distance'] = [list of distance values]
emotion_key_agg[key]['emotion_adherence'] = [list of emotion adherence]
emotion_key_agg[key]['key_correlation'] = [list of key correlations]
```

Cálculo de estadísticas finales Para cada combinación de parámetros, se calculan:

- **Promedios de adherencia:** $\bar{a}_{\text{token}} = \frac{1}{n} \sum_{i=1}^n a_i$ donde a_i son los adherence ratios por archivo
- **Promedios de distancia:** $\bar{d}_{\text{token}} = \frac{1}{n} \sum_{i=1}^n d_i$ donde d_i son las distancias medias por archivo
- **Correlación tonal media:** $\bar{c}_{\text{key}} = \frac{1}{n} \sum_{i=1}^n c_i$ donde c_i son las correlaciones con tonalidad objetivo

Comparación entre modos de inferencia Los resultados se agrupan por (`model`, `mode`) y se promedian sobre todas las condiciones de emoción, tonalidad y tokens, generando una tabla de comparación que presenta:

- **Adherencia promedio por token:** Para cada modo de inferencia y token de condicionamiento
- **Distancia promedio:** Indicando proximidad numérica promedio al valor objetivo

- **Adherencia emocional:** Porcentaje de coincidencia con cuadrante emocional objetivo
- **Correlación tonal:** Score de Krumhansl-Schmuckler con tonalidad objetivo (símbolo ↑ indica que valores altos son mejores)

6.6.2. Resultados de la evaluación

Comparación de adherencia por modo de inferencia

La Tabla 8 presenta los resultados agregados de adherencia para los modelos scratch y pretrained, promediando sobre todas las emociones, tonalidades y combinaciones de condiciones evaluadas.

Modelo	Modo de Inferencia	Token	Adherencia (%)
SCRATCH	deterministic	duration	8.05
		velocity	1.36
		emotion	45.83
		key	41.67
	primed	duration	7.14
		velocity	1.78
		emotion	35.42
		key	33.33
	forced	emotion	25.00
		key	8.33
PRETRAINED	deterministic	duration	9.37
		velocity	7.20
		emotion	50.00
	primed	duration	3.15
		velocity	2.72
		emotion	18.75
	forced	emotion	31.25

Cuadro 8: Comparación de adherencia promedio entre modos de inferencia para modelos scratch y pretrained. Los valores representan porcentajes de adherencia categórica, promediados sobre todas las emociones, tonalidades (scratch) y combinaciones de condicionamiento evaluadas.

Métricas de distancia numérica

La Tabla 9 presenta las métricas de distancia numérica para tokens con valores cuantitativos (**duration** y **velocity**), así como la correlación tonal para el modelo scratch.

Análisis de tiempos de ejecución

El pipeline de evaluación implementó timeouts de 30 segundos por generación para prevenir generaciones infinitas, estableciendo una cota superior de tiempo por muestra. La ejecución completa del pipeline, incluyendo las 192 generaciones programadas (144 para

Modelo	Modo	Token	Distancia Media
SCRATCH	deterministic	duration	677.59
		velocity	23.52
		key	0.7444 ↑
	primed	duration	762.87
		velocity	25.52
	forced	key	0.7556 ↑
PRETRAINED	deterministic	duration	567.56
		velocity	10.74
	primed	duration	811.88
		velocity	18.82

Cuadro 9: Métricas de distancia numérica promedio. Para *duration* y *velocity*, valores menores indican mayor proximidad al objetivo. Para *key* (correlación tonal de Krumhansl-Schmuckler), valores cercanos a 1.0 indican mayor adherencia tonal (indicado con ↑).

modelo scratch y 48 para modelo pretrained) y el análisis de adherencia de cada composición, requirió los siguientes tiempos:

- **Tiempo total de evaluación:** 4938.34 segundos (aproximadamente 82.3 minutos o 1.37 horas)
- **Tiempo de análisis de adherencia:** 4177.34 segundos (aproximadamente 69.6 minutos), correspondiendo al 84.6 % del tiempo total
- **Tiempo de generación:** 761 segundos (aproximadamente 12.7 minutos), correspondiendo al 15.4 % del tiempo total

Estos tiempos indican que la fase de análisis de adherencia, incluyendo la invocación del clasificador emocional EMOPIA mediante WSL (22 segundos por composición en promedio), domina el tiempo de ejecución del pipeline. El tiempo promedio por composición (generación + análisis) fue de aproximadamente 25.7 segundos, significativamente menor que la cota superior estimada de 42 segundos, indicando que la mayoría de generaciones completaron exitosamente sin alcanzar el timeout. Este tiempo de ejecución es razonable para un pipeline de evaluación exhaustiva, permitiendo iteración y refinamiento de experimentos en ciclos diarios de investigación.

6.6.3. Materiales y recursos utilizados

- **Software y librerías:**
 - Jupyter Notebook como entorno de ejecución del pipeline de evaluación
 - Python 3.13.2 como lenguaje de implementación
 - NumPy para cálculos numéricos y estadísticas
 - miditoolkit para análisis de archivos MIDI y extracción de pitch classes
 - subprocess para invocación de scripts de análisis y generaciones

- WSL (Windows Subsystem for Linux) para ejecución del clasificador EMOPIA
- **Scripts especializados desarrollados:**
 - `analyze_adherence.py`: Evaluación de adherencia a tokens de bajo nivel con métricas categóricas y de distancia
 - `analyze_emotion_key.py`: Evaluación de adherencia emocional (clasificador EMOPIA) y tonal (Krumhansl-Schmuckler)
- **Scripts existentes integrados:**
 - `main_cp.py`: Script de generación con soporte para múltiples modos de inferencia (de Fase 5)
 - Clasificador EMOPIA oficial (`inference.py`) ejecutado en entorno WSL
- **Modelos evaluados:**
 - Modelo scratch: Checkpoint `loss_08_params.pt` de la Fase 4 con soporte para 9 dimensiones
 - Modelo pretrained: Checkpoint original de EMOPIA con 8 dimensiones
- **Infraestructura computacional:**
 - Entorno local con Windows 11 para ejecución del pipeline
 - WSL2 con Ubuntu para ejecución del clasificador EMOPIA (Python 3.8.5, PyTorch 1.11.0 CPU)
 - GPU RTX 2060 para generaciones
- **Período de ejecución:** Octubre 2025
- **Ubicación:** Desarrollo local (Windows 11) con integración WSL

6.6.4. Productos de la Fase 6

1. Pipeline de evaluación automatizado:

- Notebook `evaluation_comprehensive_full.ipynb` con sistema completo de generación, análisis y agregación
- Muestreo estratificado de valores de condicionamiento para cobertura representativa
- Construcción de grilla de evaluación con 192 experimentos (144 scratch + 48 pretrained)
- Sistema de timeout y manejo de errores robusto
- Persistencia de metadata detallada por composición generada

2. Scripts de análisis de adherencia:

- `analyze_adherence.py` con métricas categóricas y de distancia numérica para tokens de bajo nivel

- `analyze_emotion_key.py` con implementación de Krumhansl-Schmuckler e integración del clasificador EMOPIA
- Soporte para procesamiento batch de múltiples composiciones
- Salida estructurada en formato JSON para agregación posterior

3. Implementación del algoritmo de Krumhansl-Schmuckler para evaluación:

- Función `compute_pitch_class_histogram` para extracción de distribución tonal
- Función `detect_key_from_midi` para detección de tonalidad con correlación coseno
- Función `compute_key_correlation` para métrica continua de adherencia tonal
- Perfiles tonales empíricos de Krumhansl-Kessler integrados

4. Configuración y documentación del clasificador EMOPIA en WSL:

- Entorno WSL2 configurado con Python 3.8.5 mediante pyenv
- PyTorch 1.11.0 CPU instalado con dependencias completas
- Procedimiento documentado de instalación y configuración (`guides/steps/retrieving.md`)
- Script de invocación mediante subprocess con conversión de rutas Windows-WSL

5. Resultados cuantitativos de evaluación:

- Tabla comparativa de adherencia por modo de inferencia (Tabla 8)
- Tabla de métricas de distancia numérica (Tabla 9)
- 192 composiciones generadas con metadata completa
- Análisis de tiempos de ejecución del pipeline

6. Insights sobre métodos de inferencia:

- Identificación de patrones de desempeño diferencial entre modos (deterministic, primed, forced)
- Cuantificación de trade-offs entre adherencia a tokens de bajo nivel y propiedades emergentes
- Análisis de adherencia emocional y tonal por modo de inferencia
- Documentación de hallazgos paradójicos (ej. modo forced con mayor correlación tonal)

7. Documentación técnica y procedimientos:

- Descripción detallada de arquitectura del pipeline de evaluación
- Documentación de técnicas de evaluación de adherencia implementadas
- Procedimientos de configuración del entorno WSL para clasificador EMOPIA
- Soluciones a desafíos de integración (timeouts, conversión de rutas, manejo de errores)

6.6.5. Conexión con objetivos específicos

Esta fase completó las actividades críticas para los siguientes objetivos específicos del proyecto:

- **Objetivo Específico 4 (Desarrollo de pipeline de evaluación):**

- Se implementó exitosamente un pipeline automatizado que integra generación, análisis de adherencia y agregación estadística en Jupyter Notebook.
- Se desarrollaron scripts especializados (`analyze_adherence.py`, `analyze_emotion-key.py`) que implementan técnicas de evaluación fundamentadas en el marco teórico.
- Se aplicó el algoritmo de Krumhansl-Schmuckler para cuantificación objetiva de adherencia tonal mediante correlación con perfiles teóricos.
- Se integró el clasificador EMOPIA oficial para cuantificación de adherencia emocional basada en el modelo circumplejo de Russell.
- Se implementaron métricas duales: adherencia categórica (match/no-match) y métricas continuas (distancia numérica, correlación tonal) para evaluación multidimensional.

- **Objetivo Específico 5 (Evaluación y comparación cuantitativa):**

- Se ejecutó evaluación sistemática del modelo scratch (9 dimensiones con tonalidad) versus modelo pretrained (8 dimensiones sin tonalidad).
- Se realizó comparación cuantitativa de tres métodos de inferencia implementados en la Fase 5 (deterministic, primed, forced).
- Se analizó adherencia multiparamétrica: tokens de bajo nivel (duration, velocity), emoción, y tonalidad, cuantificando desempeño en cada dimensión.
- Se generaron tablas comparativas (Tablas 8 y 9) que documentan patrones de desempeño diferencial entre modos de inferencia.
- Se identificaron trade-offs entre adherencia a diferentes tipos de condicionamiento, proporcionando evidencia cuantitativa para análisis en el capítulo de Resultados y Discusión.

Alcance completado: Esta fase completó exitosamente el desarrollo e implementación del pipeline de evaluación (Objetivo 4) y la ejecución de evaluaciones comparativas cuantitativas (Objetivo 5). El producto final es un sistema automatizado de evaluación con 192 experimentos ejecutados, métricas objetivas de adherencia calculadas, y resultados agregados que permiten comparación sistemática entre el modelo extendido y el original, y entre diferentes métodos de inferencia.

Próximos pasos: Los resultados cuantitativos generados en esta fase serán analizados en profundidad en el capítulo de Resultados y Discusión, interpretando los patrones de adherencia observados, discutiendo las implicaciones de los trade-offs identificados, y comparando el desempeño del sistema extendido con el sistema EMOPIA original.

6.6.6. Desafíos encontrados y soluciones implementadas

Desafío 1: Integración del clasificador EMOPIA

Problema El clasificador EMOPIA requiere un entorno Python 3.8 con dependencias específicas (PyTorch, miditoolkit) y debe ejecutarse en Linux. La integración en el pipeline de evaluación en Windows requirió configuración de WSL y manejo de conversión de rutas.

Solución Se implementó invocación del clasificador mediante `subprocess` con conversión automática de rutas Windows a WSL (`S:\ → /mnt/s/`). Se agregó manejo de timeouts (30 segundos) para prevenir bloqueos en clasificaciones complejas.

La configuración completa del entorno WSL incluyó:

- Instalación de Python 3.8.5 mediante `pyenv` para aislar la versión específica requerida
- Creación de entorno virtual `my_venv_38` dedicado al clasificador
- Instalación de PyTorch 1.11.0 CPU debido a incompatibilidades de CUDA 12.x con las versiones originales requeridas por EMOPIA
- Instalación de dependencias de procesamiento MIDI (`miditoolkit`, `pretty_midi`, `chorder`)
- Descarga y ubicación de pesos pre-entrenados del modelo `midi_like` para clasificación arousal-valence

Este entorno se configuró en `/mnt/s/UVG/LOCKIN/EMOPIA_cls/`, permitiendo acceso bidireccional entre el sistema Windows (donde se ejecuta el pipeline de evaluación) y el entorno Linux (donde se ejecuta el clasificador). La elección de la versión CPU de PyTorch, si bien reduce velocidad de inferencia, garantiza compatibilidad sin requerir configuración de GPU pass-through en WSL2, manteniendo tiempos de clasificación aceptables (<5 segundos por archivo MIDI).

Desafío 2: Manejo de modo forced sin tokens de bajo nivel

Problema El modo `inference-forced` no genera métricas de adherencia de tokens de bajo nivel (duration, velocity) porque estos tokens se establecen directamente durante generación, haciendo la adherencia tautológicamente 100 %.

Solución Se modificó el pipeline para:

1. Omitir análisis de tokens de bajo nivel para modo forced
2. Enfocarse exclusivamente en adherencia emocional y tonal (propiedades emergentes de alto nivel)

3. Documentar explícitamente en tablas de resultados la ausencia de métricas de tokens para este modo

Esta decisión se alinea con el marco teórico que distingue entre adherencia a variables directamente controladas versus propiedades estructurales emergentes.

Desafío 3: Explosión combinatoria del espacio de evaluación

Problema La combinación completa de modelos (2), modos (3), emociones (4), tonalidades (3 para scratch), y subconjuntos de tokens ($2^2 = 4$) genera potencialmente cientos de experimentos, con tiempos de generación de 10-30 segundos cada uno.

Solución Se implementó:

- **Muestreo estratificado:** Solo 2 valores por token en lugar de vocabulario completo
- **Subconjuntos limitados:** Solo combinaciones de tamaño 1 en lugar de todas las combinaciones posibles
- **Ejecución secuencial con logging:** Pipeline con barra de progreso y almacenamiento intermedio de resultados
- **Caching de resultados:** Archivos `.meta.json` permiten reanálisis sin regeneración

Esto redujo el número de generaciones a un total manejable (192 generaciones: 144 scratch + 48 pretrained) con tiempo de ejecución total de aproximadamente 1.37 horas (4938 segundos), permitiendo completar ciclos completos de evaluación en un tiempo razonable para investigación iterativa.

6.6.7. Reflexiones sobre la fase

Validación de técnicas del marco teórico Esta fase validó empíricamente la aplicabilidad de dos técnicas fundamentales del marco teórico:

1. **Algoritmo de Krumhansl-Schmuckler:** Su aplicación a composiciones generadas confirmó su utilidad para cuantificar adherencia tonal de manera continua (correlación) además de categórica (match exacto). Las correlaciones medias observadas (0.56-0.76) indican que el algoritmo discrimina efectivamente entre composiciones con diferentes grados de adherencia tonal, con valores altos (>0.74) para modos determinístico y primed, y valores moderados (0.56) para modo forced.
2. **Clasificador emocional:** El clasificador EMOPIA, entrenado en el mismo dataset utilizado para entrenar el modelo generativo, proporciona una métrica de adherencia emocional consistente con el modelo circumplejo de Russell.

Insights sobre métodos de inferencia Los resultados finales revelan patrones importantes sobre el comportamiento de los diferentes modos de inferencia:

- **Adherencia emocional diferencial:** El modo determinístico muestra consistentemente mayor adherencia emocional (45.83-50.00 %) que primed (18.75-35.42 %) o forced (25.00-31.25 %), sugiriendo que el muestreo determinístico preserva mejor la coherencia emocional global. El descenso pronunciado en adherencia emocional del modo primed en el modelo pretrained (18.75 %) indica mayor sensibilidad del modelo a perturbaciones en la secuencia de condicionamiento.
- **Trade-off entre control local y coherencia global:** El modo primed muestra adherencia comparable a tokens de bajo nivel (7.14 % duration, 1.78 % velocity en scratch) respecto al determinístico (8.05 %, 1.36 %), pero con adherencia emocional significativamente reducida, especialmente en el modelo pretrained. Esto evidencia un potencial conflicto entre el control local mediante priming de tokens y la preservación de coherencia emocional global.
- **Adherencia tonal en modelo scratch:** Los modos determinístico y primed muestran correlaciones tonales altas y similares (0.7444 y 0.7556 respectivamente), con adherencia categórica moderada (41.67 % y 33.33 %). En contraste, el modo forced presenta menor correlación tonal (0.5594) a pesar de forzar explícitamente los tokens de tonalidad, sugiriendo que el forzamiento puede interferir con la coherencia tonal estructural de la composición completa.
- **Superior desempeño en velocity del modelo pretrained:** El modelo pretrained muestra adherencia a velocity significativamente mayor (7.20 % determinístico vs 1.36 % scratch) con distancias menores (10.74 vs 23.52), indicando mejor calibración del modelo original en esta dimensión específica.

Estos patrones serán analizados en profundidad en el capítulo de Resultados y Discusión.

Reproducibilidad y extensibilidad El pipeline implementado es completamente reproducible (seed fija: 2084) y extensible:

- Nuevos modelos pueden agregarse a la lista MODELS
- Nuevos modos de inferencia pueden integrarse con mínima modificación
- Nuevas métricas pueden agregarse a los scripts de análisis
- El notebook documenta cada paso mediante celdas markdown y logging detallado

Esta arquitectura facilita investigación futura que extienda la evaluación a más dimensiones de control o metodologías de análisis alternativas.

CAPÍTULO 7

Resultados

Este capítulo presenta los resultados obtenidos en cada fase del proyecto, organizados según los objetivos específicos establecidos. Los resultados se presentan de manera objetiva y cuantitativa, referenciando las metodologías detalladas en el Capítulo de Metodología. Las interpretaciones y análisis de estos resultados se desarrollarán en el Capítulo de Discusión.

7.1. Extensión del dataset EMOPIA con información de tonalidad

El primer objetivo específico consistió en extender el dataset EMOPIA mediante la integración de información de tonalidad detectada automáticamente con el algoritmo de Krumhansl-Schmuckler. La metodología completa se describe en la Fase 3 de la Metodología.

7.1.1. Características del dataset extendido

El procesamiento del dataset EMOPIA original mediante el pipeline de Compound Word Transformer y la integración del algoritmo de detección de tonalidad resultó en un dataset tokenizado con las siguientes características:

- **Número de secuencias:** 824 (de 1,071 clips MIDI originales, 854 clips de audio recuperados, 832 procesados por el pipeline, 8 excluidos por errores de preprocesamiento)
- **Dimensiones de entrada:** (824, 1024, 9)
 - 824 secuencias de entrenamiento

- 1,024 tokens por secuencia (con padding)
- 9 características por token
- **Características tokenizadas:** tempo, chord, bar-beat, type, pitch, duration, velocity, emotion, **key**
- **Formato de almacenamiento:** `train_data_linear.npz` con arrays `x`, `y`, `mask`, `seq_len`, `num_groups`

7.1.2. Vocabulario de tonalidad

El diccionario de vocabulario generado (`train_dictionary.pkl`) contiene los tamaños de vocabulario presentados en el Cuadro 10.

Dimensión	Tamaño de vocabulario
tempo	53
chord	127
bar-beat	18
type	5
pitch	85
duration	18
velocity	41
emotion	5
key	25

Cuadro 10: Tamaños de vocabulario por dimensión en el dataset EMOPIA extendido. La dimensión de tonalidad (key) contiene 25 tokens: 24 tonalidades (12 mayores + 12 menores) y 1 token especial CONTI.

La novena dimensión (key) incluye:

- 12 tonalidades mayores: C:maj, C#:maj, D:maj, D#:maj, E:maj, F:maj, F#:maj, G:maj, G#:maj, A:maj, A#:maj, B:maj
- 12 tonalidades menores: C:min, C#:min, D:min, D#:min, E:min, F:min, F#:min, G:min, G#:min, A:min, A#:min, B:min
- 1 token de continuación: CONTI (indica ausencia de cambio de tonalidad)

7.1.3. Distribución de longitudes de secuencias

El análisis de las 824 secuencias tokenizadas antes de aplicar padding reveló las estadísticas presentadas en el Cuadro 11.

Métrica	Valor
Longitud mínima	100 tokens
Longitud máxima	1,008 tokens
Longitud media	442.77 tokens
Desviación estándar	203.68 tokens
Longitud con padding	1,024 tokens

Cuadro 11: Estadísticas de longitudes de secuencias en el dataset EMOPIA extendido antes y después de padding. Todas las secuencias se paddearon a 1,024 tokens para entrenamiento.

7.2. Modelo Transformer extendido y entrenamiento

El segundo objetivo específico consistió en adaptar la arquitectura del modelo Transformer de EMOPIA para procesar la novena dimensión de tonalidad y entrenar el modelo extendido. La metodología completa se describe en la Fase 4 de la Metodología.

7.2.1. Características arquitectónicas del modelo

El modelo Transformer extendido presenta las especificaciones técnicas del Cuadro 12.

Parámetro	Valor
Parámetros entrenables	39,168,153
Número de capas encoder	12
Dimensión del modelo (d_model)	512
Número de cabezales de atención	8
Dimensión feed-forward	2,048
Dropout	0.1
Función de activación	GELU
Tipo de atención	Causal lineal
Embeddings por dimensión	
Tempo	128
Chord	256
Bar-beat	64
Type	32
Pitch	512
Duration	128
Velocity	128
Emotion	128
Key	128
Dimensión total concatenada	1,664

Cuadro 12: Especificaciones técnicas del modelo Transformer extendido con soporte para 9 dimensiones de entrada. La dimensión de tonalidad (key) utiliza embeddings de 128 dimensiones, consistente con emotion y velocity.

7.2.2. Métricas de entrenamiento

El modelo se entrenó desde cero (sin preentrenamiento) en el dataset EMOPIA extendido utilizando infraestructura de 4 GPUs RTX 5090 en paralelo en RunPod. El Cuadro 13 presenta las métricas globales del proceso de entrenamiento.

Métrica	Valor
Número de épocas completadas	875
Loss inicial (época 1)	1.8109
Loss final (época 875)	0.0798
Reducción absoluta de loss	1.7311
Reducción porcentual	95.59 %
Épocas con mejora	827 de 874 (94.6 %)
Tiempos de ejecución	
Tiempo total de entrenamiento	17,274.27 segundos (4.80 horas)
Tiempo promedio por época	19.76 segundos
Desviación estándar (tiempo/época)	0.34 segundos
Rango de tiempos por época	17.04 - 21.04 segundos
Infraestructura	
GPU utilizada	4× NVIDIA RTX 5090 (96GB VRAM total)
Batch size	4
Número de batches por época	206
Costo total de entrenamiento	\$17.09 USD

Cuadro 13: Métricas de entrenamiento del modelo Transformer extendido en dataset EMOPIA con 9 dimensiones. El entrenamiento se completó en 875 épocas con convergencia exitosa ($\text{loss} < 0.08$).

7.2.3. Convergencia por dimensión

El análisis de los losses individuales por dimensión durante el entrenamiento reveló patrones de convergencia diferenciados. El Cuadro 14 presenta las métricas de convergencia para cada dimensión del modelo.

Dimensión	Loss inicial	Loss final	Reducción	Reducción (%)
Emotion	0.2935	0.0000	0.2935	100.00
Bar-beat	1.7095	0.0159	1.6936	99.07
Key	0.5170	0.0080	0.5090	98.46
Chord	2.2662	0.0380	2.2282	98.32
Tempo	1.7514	0.0588	1.6926	96.64
Pitch	3.6000	0.1557	3.4443	95.67
Duration	2.3844	0.1321	2.2523	94.46
Velocity	3.0378	0.2486	2.7892	91.82
Type	0.7386	0.0609	0.6777	91.75

Cuadro 14: Convergencia de losses individuales por dimensión durante el entrenamiento del modelo extendido. La dimensión de tonalidad (key) alcanzó 98.46 % de reducción, superando dimensiones estructurales como tempo (96.64 %) y pitch (95.67 %).

La dimensión de tonalidad (key) demostró convergencia superior a la mayoría de dimensiones estructurales, alcanzando el tercer lugar en reducción porcentual de loss después de emotion (100 %) y bar-beat (99.07 %).

7.2.4. Comparación de velocidad de entrenamiento

El Cuadro 15 compara el rendimiento de entrenamiento entre el entorno local (RTX 2060) evaluado en la Fase 2 y el entorno en la nube ($4 \times$ RTX 5090) utilizado para el entrenamiento completo.

Métrica	RTX 2060 (local)	$4 \times$ RTX 5090 (RunPod)	Mejora
Tiempo por época	87.22 segundos	19.76 segundos	$4.41 \times$
Tiempo estimado (875 épocas)	21.2 horas	4.80 horas	$4.41 \times$
VRAM disponible	6 GB	96 GB	$16.00 \times$
Costo directo	\$0 (hardware local)	\$17.09 USD	-

Cuadro 15: Comparación de rendimiento de entrenamiento entre entorno local (RTX 2060) y entorno en la nube ($4 \times$ RTX 5090). La aceleración de $4.41 \times$ justificó el uso de infraestructura en la nube para el entrenamiento completo.

7.3. Sistema de inferencia multiparamétrica

El tercer objetivo específico consistió en implementar métodos avanzados de inferencia para control granular de la generación musical mediante diferentes estrategias de condicionamiento. La metodología completa se describe en la Fase 5 de la Metodología.

7.3.1. Modos de inferencia implementados

Se implementaron cuatro modos de inferencia especializados, cuyas características se resumen en el Cuadro 16.

7.3.2. Parámetros de sampling por defecto

El sistema de configuración de sampling implementado permite ajustar temperatura y nucleus sampling de forma global o por token individual. El Cuadro 17 presenta los parámetros por defecto para cada dimensión.

7.3.3. Capacidades del sistema de inferencia

El sistema implementado proporciona las funcionalidades cuantificadas en el Cuadro 18.

Modo	Características
inference-normal	Generación con parámetros de sampling por defecto. Temperatura y nucleus sampling específicos por token. Sirve como baseline.
inference-deterministic	Temperatura y nucleus p extremadamente bajos (0.001) en todas las dimensiones. Selección argmax (token más probable). Reproducibilidad perfecta con mismos parámetros de entrada.
inference-primed	Condicionamiento mediante valores fijos en el token de emoción inicial. Permite especificar tempo, chord, pitch, duration, velocity, emotion, y key iniciales. Influencia por contexto de atención.
inference-forced	Forzado de tokens específicos en cada paso de generación. Sobrescritura post-sampling. Control absoluto sobre dimensiones especificadas durante toda la composición.

Cuadro 16: Características de los cuatro modos de inferencia implementados en el sistema extendido. Cada modo proporciona diferentes niveles de control sobre la generación musical.

Token	Temperatura	Nucleus p
type	-	0.90
tempo	1.2	0.9
chord	-	0.99
barbeat	1.2	-
pitch	-	0.9
duration	2.0	0.9
velocity	5.0	-
emotion	-	-
key	1.2	-

Cuadro 17: Parámetros de sampling por defecto para cada token. - indica ausencia de restricción específica. Estos valores pueden sobrescribirse mediante configuración global o por token individual.

7.4. Evaluación de adherencia al condicionamiento

El cuarto objetivo específico consistió en desarrollar un pipeline de evaluación que cuantifique la adherencia de las composiciones generadas a las variables de condicionamiento especificadas. La metodología completa se describe en la Fase 6 de la Metodología.

7.4.1. Configuración de experimentos

El pipeline de evaluación ejecutó un diseño experimental con las características presentadas en el Cuadro 19.

Capacidad	Valor/Descripción
Número de modos de inferencia	4
Argumentos de CLI disponibles	11
Tokens configurables individualmente	9
Parámetros de sampling ajustables	2 (temperatura, nucleus p)
Formato de condiciones	JSON estructurado
Timeout de seguridad	60 segundos (configurable)
Campos en metadata por composición	17
Compatibilidad con modelos	8 y 9 tokens

Cuadro 18: Capacidades cuantitativas del sistema de inferencia multiparamétrica implementado. El sistema proporciona control granular sobre múltiples variables simultáneamente.

Parámetro	Valor
Modelos evaluados	2 (scratch, pretrained)
Modos de inferencia evaluados	3 (deterministic, primed, forced)
Emociones evaluadas	4 (Q1, Q2, Q3, Q4)
Tonalidades (modelo scratch)	3 (A:maj, C:maj, E:min)
Tonalidades (modelo pretrained)	0 (sin soporte)
Tokens de control evaluados	2 (duration, velocity)
Valores por token	2 (muestreo estratificado)
Subconjuntos de tokens	4 ({{}, {dur}, {vel}, {dur,vel}})
Muestras por combinación	1
Totales	
Generaciones modelo scratch	144
Generaciones modelo pretrained	48
Generaciones totales	192
Composiciones únicas evaluadas	192

Cuadro 19: Configuración del diseño experimental para evaluación de adherencia. El pipeline ejecutó 192 experimentos sistemáticos cubriendo múltiples combinaciones de condicionamiento.

7.4.2. Métricas de evaluación implementadas

El pipeline implementó múltiples técnicas de evaluación de adherencia, resumidas en el Cuadro 20.

7.4.3. Tiempos de ejecución del pipeline

El Cuadro 21 presenta los tiempos de ejecución del pipeline de evaluación completo.

Categoría	Métrica	Descripción
Tokens de bajo nivel	Adherencia categórica	Proporción de tokens que coinciden exactamente con valor objetivo
	Distancia numérica	Media, mediana, desviación estándar y rango de distancias al valor objetivo
Tonalidad	Adherencia categórica	Proporción de composiciones con tonalidad detectada coincidente
	Correlación tonal	Score de Krumhansl-Schmuckler con perfil de tonalidad objetivo (rango 0-1)
Emoción	Adherencia categórica	Proporción de composiciones con cuadrante emocional coincidente

Cuadro 20: Técnicas de evaluación de adherencia implementadas en el pipeline. Las métricas abarcan adherencia categórica (binaria) y métricas continuas (distancias, correlaciones).

Componente	Tiempo (segundos)	Porcentaje
Análisis de adherencia	4,177.34	84.6 %
Generación de composiciones	761.00	15.4 %
Total	4,938.34	100 %
Métricas derivadas		
Tiempo total (minutos)	82.3	-
Tiempo total (horas)	1.37	-
Tiempo promedio por composición	25.7	-
Timeout configurado	30.0	-
Composiciones que alcanzaron timeout	0	0 %

Cuadro 21: Tiempos de ejecución del pipeline de evaluación completo para 192 experimentos. El análisis de adherencia domina el tiempo total (84.6 %), incluyendo clasificación emocional mediante EMOPIA.

7.5. Comparación cuantitativa: modelo extendido vs. modelo original

El quinto objetivo específico consistió en realizar una comparación cuantitativa entre el modelo extendido (con soporte para tonalidad) y el modelo original de EMOPIA. La metodología completa se describe en la Fase 6 de la Metodología.

7.5.1. Adherencia por modo de inferencia

El Cuadro 22 presenta los resultados agregados de adherencia categórica para ambos modelos, promediando sobre todas las emociones, tonalidades y combinaciones de condiciones evaluadas.

Elaboración propia.

Modelo	Modo	Token	Adherencia (%)
SCRATCH	deterministic	duration	8.05
		velocity	1.36
		emotion	45.83
		key	41.67
	primed	duration	7.14
		velocity	1.78
		emotion	35.42
		key	33.33
	forced	emotion	25.00
		key	8.33
PRETRAINED	deterministic	duration	9.37
		velocity	7.20
		emotion	50.00
	primed	duration	3.15
		velocity	2.72
		emotion	18.75
	forced	emotion	31.25

Cuadro 22: Adherencia categórica promedio (%) por modo de inferencia para modelos scratch (9 dimensiones con tonalidad) y pretrained (8 dimensiones sin tonalidad). Los valores representan porcentajes de coincidencia exacta con valores objetivo, promediados sobre todas las condiciones evaluadas.

7.5.2. Métricas de distancia numérica y correlación tonal

El Cuadro 23 presenta las métricas de distancia numérica para tokens cuantitativos y correlación tonal para el modelo scratch.

Modelo	Modo	Token/Métrica	Valor
SCRATCH	deterministic	duration (dist. media)	677.59
		velocity (dist. media)	23.52
		key (correlación)	0.7444
	primed	duration (dist. media)	762.87
		velocity (dist. media)	25.52
	forced	key (correlación)	0.7556
PRETRAINED	deterministic	key (correlación)	0.5594
		duration (dist. media)	567.56
	primed	velocity (dist. media)	10.74
		duration (dist. media)	811.88
		velocity (dist. media)	18.82

Cuadro 23: Métricas de distancia numérica promedio para tokens **duration** y **velocity**, y correlación tonal de Krumhansl-Schmuckler para **key**. Para distancias, valores menores indican mayor proximidad al objetivo. Para correlación tonal, valores cercanos a 1.0 indican mayor adherencia.

Elaboración propia.

7.5.3. Comparación de adherencia emocional

El Cuadro 24 compara específicamente la adherencia emocional entre modelos y modos de inferencia.

Modelo	Modo de Inferencia	Adherencia Emocional (%)
SCRATCH	deterministic	45.83
	primed	35.42
	forced	25.00
PRETRAINED	deterministic	50.00
	primed	18.75
	forced	31.25

Cuadro 24: Adherencia emocional (%) por modo de inferencia para ambos modelos. El modelo pretrained muestra mayor adherencia en modo determinístico (50.00 % vs 45.83 %), pero menor adherencia en modo primed (18.75 % vs 35.42 %).

Elaboración propia.

7.5.4. Adherencia tonal del modelo scratch

El Cuadro 25 presenta las métricas específicas de adherencia tonal para el modelo scratch, único modelo con soporte para condicionamiento de tonalidad.

Modo de Inferencia	Adherencia Categórica (%)	Correlación Media
deterministic	41.67	0.7444
primed	33.33	0.7556
forced	8.33	0.5594

Cuadro 25: Métricas de adherencia tonal para el modelo scratch. La adherencia categórica indica coincidencia exacta de tonalidad detectada vs. objetivo. La correlación media representa el score de Krumhansl-Schmuckler con perfil tonal objetivo (valores cercanos a 1.0 indican mayor adherencia).

Elaboración propia.

7.5.5. Patrones de adherencia a tokens de bajo nivel

El Cuadro 26 resume los patrones de adherencia para tokens de control de bajo nivel (*duration* y *velocity*) entre modelos y modos.

Elaboración propia.

7.6. Resumen cuantitativo de resultados

El Cuadro 27 sintetiza los principales resultados cuantitativos obtenidos en el proyecto, organizados por objetivo específico.

Token	Modo	Scratch (%)	Pretrained (%)
duration	deterministic	8.05	9.37
	primed	7.14	3.15
velocity	deterministic	1.36	7.20
	primed	1.78	2.72

Cuadro 26: Comparación de adherencia categórica (%) para tokens *duration* y *velocity* entre modelos scratch y pretrained. El modelo pretrained muestra adherencia superior en *velocity* (7.20 % vs 1.36 % en modo determinístico).

Objetivo	Resultado	Valor
Obj. 1: Dataset	Secuencias con tonalidad	824
	Dimensiones por token	9
	Vocabulario de tonalidad	25 tokens
Obj. 2: Modelo	Parámetros entrenables	39,168,153
	Épocas de entrenamiento	875
	Reducción de loss	95.59 %
	Convergencia de key	98.46 %
Obj. 3: Inferencia	Modos implementados	4
	Argumentos CLI	11
Obj. 4: Pipeline	Experimentos ejecutados	192
	Tiempo total evaluación	1.37 horas
	Métricas por categoría	3
Obj. 5: Comparación	Adherencia emocional scratch (det.)	45.83 %
	Adherencia emocional pretrained (det.)	50.00 %
	Adherencia tonal scratch (det.)	41.67 %
	Correlación tonal scratch (det.)	0.7444

Cuadro 27: Síntesis de resultados cuantitativos principales organizados por objetivo específico. Los valores corresponden a métricas representativas de cada objetivo completado.

Elaboración propia.

Los resultados presentados en este capítulo demuestran la completitud de los cinco objetivos específicos establecidos: (1) extensión exitosa del dataset EMOPIA resultando en 824 secuencias tokenizadas con información de tonalidad (77.0 % del dataset original de 1,071 clips), (2) adaptación y entrenamiento exitoso del modelo Transformer extendido con convergencia del 98.46 % en la dimensión de tonalidad, (3) implementación de sistema de inferencia multiparamétrica con 4 modos especializados, (4) desarrollo de pipeline automatizado que ejecutó 192 experimentos en 1.37 horas, y (5) comparación cuantitativa que reveló patrones de adherencia diferenciados entre modelos y modos de inferencia. La interpretación y análisis de estos resultados se desarrollará en el Capítulo de Discusión.

CAPÍTULO 8

Discusión

Este capítulo interpreta y analiza los resultados presentados en el Capítulo de Resultados con relación a los cinco objetivos específicos establecidos. Se proponen explicaciones para los hallazgos observados, se plantean hipótesis alternativas, y se valida el sistema extendido en comparación con el modelo original EMOPIA. La discusión se organiza alrededor de los principales hallazgos y sus implicaciones para la generación musical condicionada.

8.1. Extensión exitosa del dataset con información de tonalidad

El primer objetivo específico consistió en extender el dataset EMOPIA mediante la incorporación de información de tonalidad detectada algorítmicamente. Los resultados muestran que partiendo de 1,071 clips MIDI originales, se recuperaron 854 clips de audio (79.7 %) debido a la no disponibilidad de videos en YouTube, y de estos se procesaron exitosamente 824 archivos (96.5 % de los clips recuperados, 77.0 % del dataset original), generando un vocabulario de 25 tokens de tonalidad que incluyen las 24 tonalidades posibles (12 mayores + 12 menores) más un token de continuación (CONTI).

8.1.1. Alta tasa de procesamiento y robustez del pipeline

La exclusión de únicamente 8 archivos de los 832 procesados por el pipeline (0.96 % de los archivos con audio disponible) demuestra la robustez del pipeline de procesamiento implementado. La investigación de los archivos excluidos reveló que estos contenían estructuras musicales atípicas que impidieron el beat tracking mediante madmom, específicamente:

- Archivos con tempo extremadamente variable o rubato excesivo
- Composiciones con inicios ambientales sin pulso definido
- Piezas con cambios métricos complejos no manejados por el algoritmo de downbeat detection

Este hallazgo sugiere que el dataset EMOPIA original contiene predominantemente música pop de piano con estructuras rítmicas regulares, lo cual es consistente con la descripción del dataset en la literatura [1]. La alta tasa de éxito valida la decisión de utilizar el algoritmo de Krumhansl-Schmuckler para detección de tonalidad, el cual demostró ser robusto para el repertorio pop de piano.

8.1.2. Integración natural de la dimensión de tonalidad

La integración de la tonalidad como novena dimensión siguió el paradigma de representación de Compound Word Transformer, donde cada característica musical se tokeniza de forma independiente. El tamaño del vocabulario de tonalidad (25 tokens) es comparable a otras dimensiones globales como bar-beat (18 tokens) y duration (18 tokens), lo que sugiere una granularidad apropiada para la tarea.

Un aspecto notable es la inclusión del token CONTI para representar la ausencia de cambio de tonalidad entre eventos. Esta decisión de diseño reconoce que la tonalidad es una característica que persiste a través de múltiples eventos musicales, a diferencia de características puntuales como pitch o velocity. Esta representación es consistente con la teoría musical, donde la tonalidad establece un centro tonal que típicamente se mantiene durante frases completas [56].

8.2. Convergencia excepcional de la dimensión de tonalidad durante entrenamiento

El segundo objetivo específico involucró la adaptación arquitectónica y entrenamiento del modelo extendido. Los resultados revelan que la dimensión de tonalidad alcanzó una convergencia del 98.46 %, ubicándose en tercer lugar después de emotion (100 %) y bar-beat (99.07 %), y superando a dimensiones estructurales fundamentales como tempo (96.64 %) y pitch (95.67 %).

8.2.1. Factores que explican la alta convergencia de tonalidad

Tres factores principales pueden explicar esta convergencia excepcional:

1. Naturaleza jerárquica y global de la tonalidad

La tonalidad, al igual que la emoción, es una característica global que influye en múltiples aspectos de la composición. Mientras que tokens como pitch y duration varían constantemente evento por evento, la tonalidad tiende a mantenerse estable durante secciones completas. Esta estabilidad temporal facilita el aprendizaje del modelo, ya que debe capturar patrones de largo alcance en lugar de variaciones locales complejas.

2. Correlaciones estructurales con otras dimensiones

La tonalidad presenta correlaciones naturales con otras dimensiones que el modelo puede explotar:

- **Con acordes:** Los acordes diatónicos de una tonalidad siguen patrones predecibles
- **Con pitch:** Ciertas notas son más frecuentes en tonalidades específicas (escala diatónica)
- **Con emoción:** Tonalidades mayores correlacionan con valencia positiva, menores con negativa [57]

Estas correlaciones proporcionan señales redundantes que facilitan el aprendizaje de la representación de tonalidad.

3. Tamaño óptimo del vocabulario

Con 25 tokens posibles, la tonalidad presenta un espacio de búsqueda más manejable que dimensiones como chord (127 tokens) o pitch (85 tokens). Este tamaño intermedio permite al modelo aprender representaciones distintivas sin sufrir de sparsity excesiva en los datos de entrenamiento.

8.2.2. Implicaciones del patrón de convergencia diferenciado

El hecho de que diferentes dimensiones converjan a velocidades distintas sugiere que el modelo aprende una jerarquía implícita de características musicales. Las dimensiones con convergencia más rápida (emotion, bar-beat, key) representan estructura global, mientras que las de convergencia más lenta (velocity, type, duration) capturan variaciones locales más complejas.

Esta jerarquía emergente es consistente con modelos cognitivos de percepción musical [58], donde los oyentes procesan primero características globales (tonalidad, metro, emoción) antes de detalles específicos (articulación, dinámicas locales).

8.3. Limitaciones de los métodos de inferencia para control multiparamétrico

El tercer objetivo específico consistió en implementar métodos de inferencia para generación condicionada por múltiples parámetros simultáneos. Los resultados revelan niveles de adherencia categórica sorprendentemente bajos para todos los métodos implementados, con valores máximos del 50 % para emoción y menos del 10 % para la mayoría de tokens de bajo nivel.

8.3.1. Análisis del bajo rendimiento en adherencia categórica

1. El desafío de la adherencia exacta

La métrica de adherencia categórica es extremadamente estricta, requiriendo coincidencia exacta entre el valor generado y el objetivo. Para tokens con vocabularios grandes como duration (18 valores) o velocity (41 valores), la probabilidad de coincidencia aleatoria es baja (5.6 % y 2.4 % respectivamente). Los resultados observados (3-9 % para duration, 1-7 % para velocity) son solo marginalmente superiores al azar, sugiriendo que el condicionamiento tiene efecto limitado a nivel de valores exactos.

2. Naturaleza probabilística de los modelos autoregresivos

Los Transformers autoregresivos son fundamentalmente modelos probabilísticos que aprenden distribuciones sobre secuencias. Incluso con condicionamiento fuerte, el modelo mantiene incertidumbre inherente para preservar diversidad y naturalidad en la generación. Forzar adherencia perfecta requeriría temperatura cero, lo cual típicamente resulta en generaciones repetitivas y musicalmente pobres [59].

3. Conflictos entre restricciones múltiples

El condicionamiento multiparamétrico introduce potenciales conflictos entre restricciones. Por ejemplo, especificar simultáneamente:

- Emoción Q1 (alta energía, valencia positiva)
- Velocity baja (dinámica suave)
- Duration alta (notas largas)

Crea tensión entre la expectativa de alta energía (típicamente asociada con notas cortas y fuertes) y las restricciones de dinámica suave y notas largas. El modelo debe resolver estos conflictos, resultando en compromisos que pueden no satisfacer ninguna restricción perfectamente.

8.3.2. Diferencias entre modos de inferencia

Modo determinístico: Mayor adherencia pero limitada controlabilidad

El modo determinístico mostró consistentemente la mayor adherencia, particularmente para emoción (45.83 % scratch, 50 % pretrained). Esto se debe a que la temperatura extremadamente baja (0.001) fuerza al modelo a seleccionar las opciones más probables según su entrenamiento. Sin embargo, esta mayor adherencia viene al costo de:

- Pérdida de diversidad en las generaciones
- Mayor susceptibilidad a sesgos del dataset de entrenamiento
- Incapacidad para explorar regiones menos probables pero válidas del espacio musical

Modo primed: Influencia limitada del condicionamiento inicial

El modo primed, que condiciona mediante tokens iniciales, mostró adherencia inferior al modo determinístico en la mayoría de los casos. Esto sugiere que la influencia del contexto inicial se diluye rápidamente a medida que la generación progresiona. El mecanismo de atención del Transformer, aunque teóricamente capaz de mantener dependencias de largo alcance, en la práctica parece priorizar contexto local reciente sobre condiciones iniciales distantes.

Modo forced: Paradoja del control absoluto

Sorprendentemente, el modo forced mostró la menor adherencia emocional (25 % scratch, 31.25 % pretrained) a pesar de sobrescribir tokens directamente. Este resultado contraintuitivo se explica por:

1. **Evaluación post-hoc de emoción:** La adherencia emocional se evalúa mediante el clasificador EMOPIA sobre la composición completa, no sobre tokens individuales
2. **Disrupción de coherencia musical:** Forzar tokens específicos puede crear secuencias no naturales que el clasificador no reconoce como pertenecientes a la emoción objetivo
3. **Desalineación entre representación local y percepción global:** Los tokens de emoción individuales no garantizan que la composición global transmita esa emoción

8.4. Validación del sistema extendido: Avances y limitaciones

El cuarto y quinto objetivos específicos involucraron el desarrollo del pipeline de evaluación y la comparación cuantitativa entre modelos. Los resultados permiten validar parcialmente el sistema extendido mientras revelan limitaciones fundamentales en el paradigma de condicionamiento actual.

8.4.1. Evidencia de aprendizaje efectivo de tonalidad

A pesar de las bajas tasas de adherencia categórica (8.33 %-41.67 %), las métricas de correlación tonal son considerablemente más alentadoras (0.56-0.76). Esta discrepancia sugiere que el modelo sí aprendió representaciones significativas de tonalidad, pero las expresa de forma probabilística en lugar de determinística.

La correlación tonal mide similaridad entre el perfil de pitch generado y el perfil teórico de la tonalidad objetivo según Krumhansl-Schmuckler. Valores superiores a 0.7 indican que las composiciones generadas respetan la jerarquía tonal (tónica, dominante, sensible) incluso cuando la detección algorítmica no identifica exactamente la tonalidad objetivo.

8.4.2. Comparación con modelo pretrained: Trade-offs inesperados

El modelo pretrained (sin soporte de tonalidad) mostró mayor adherencia emocional en modo determinístico (50 % vs 45.83 %) pero peor desempeño en modo primed (18.75 % vs 35.42 %). Estos resultados sugieren:

1. Beneficio del preentrenamiento en dominio amplio

El preentrenamiento en el dataset AILabs17k (17,000 piezas de piano clásico) proporciona al modelo conocimiento musical más robusto que se traduce en mejor captura de patrones emocionales cuando opera en modo de máxima probabilidad (determinístico).

2. Costo de complejidad adicional

La dimensión adicional de tonalidad incrementa la complejidad del espacio de aprendizaje. Con un dataset relativamente pequeño (824 secuencias, representando solo el 77.0 % del dataset original de 1,071 clips), el modelo scratch debe distribuir su capacidad entre más dimensiones, potencialmente resultando en representaciones menos robustas para cada una.

3. Interacción entre preentrenamiento y métodos de condicionamiento

El peor desempeño del modelo pretrained en modo primed sugiere que el conocimiento previo puede crear inercia que resiste condicionamiento débil. El modelo tiene expectativas fuertes sobre secuencias musicales “correctas” basadas en su preentrenamiento, haciéndolo menos receptivo a señales de condicionamiento sutiles.

8.5. Factores que influyen en la adherencia al condicionamiento

El análisis comprehensivo de 192 experimentos revela múltiples factores que modulan la efectividad del condicionamiento:

8.5.1. Jerarquía de controlabilidad

Los resultados sugieren una jerarquía clara en la controlabilidad de diferentes dimensiones:

1. **Alta controlabilidad:** Emoción (hasta 50 % adherencia categórica)
2. **Controlabilidad media:** Tonalidad (hasta 41.67 % categórica, 0.76 correlación)
3. **Baja controlabilidad:** Duration (3-9 %), Velocity (1-7 %)

Esta jerarquía correlaciona inversamente con la frecuencia de cambio de cada dimensión. Características estables (emoción, tonalidad) son más controlables que características altamente variables (velocity, duration).

8.5.2. Influencia del tamaño del vocabulario

Existe correlación negativa entre tamaño del vocabulario y adherencia. Emotion con 5 valores muestra mayor adherencia que velocity con 41 valores. Esto sugiere que vocabularios más grandes requieren exponencialmente más datos de entrenamiento para lograr control efectivo sobre cada valor individual.

8.5.3. Rol del contexto musical

La música es inherentemente contextual. Una nota Do puede funcionar como tónica en Do mayor o como mediante en La menor. Esta ambigüedad contextual significa que especificar valores absolutos (como forzar pitch=60) ignora el contexto armónico y melódico necesario para interpretación musical coherente.

8.6. Implicaciones para investigación futura

Los hallazgos de este trabajo sugieren varias direcciones prometedoras para investigación futura:

8.6.1. Métodos de condicionamiento alternativos

Los métodos implementados (deterministic, primed, forced) operan a nivel de tokens individuales. Métodos futuros podrían explorar:

1. Condicionamiento jerárquico

En lugar de especificar valores token por token, definir restricciones de alto nivel (e.g., "sección en La mayor con carácter enérgico") que el modelo interprete flexiblemente.

2. Condicionamiento por ejemplos

Proporcionar fragmentos musicales de referencia que exemplifiquen las características deseadas, permitiendo al modelo extraer patrones implícitos.

3. Condicionamiento adversarial

Entrenar discriminadores especializados para cada dimensión de control que proporcionen gradientes durante la generación para mejorar adherencia.

8.6.2. Arquitecturas especializadas para control

La arquitectura Transformer utilizada fue diseñada para modelado de lenguaje, no específicamente para control generativo. Arquitecturas futuras podrían incorporar:

- Módulos de atención especializados para diferentes escalas temporales
- Representaciones latentes disentangled para factores de control independientes
- Mecanismos de memoria explícita para mantener restricciones globales

8.6.3. Evaluación más allá de adherencia categórica

Las métricas de adherencia categórica son limitadas para evaluar control musical. Futuras evaluaciones podrían incorporar:

- Métricas perceptuales basadas en modelos de cognición musical
- Evaluación humana de controlabilidad percibida
- Métricas de coherencia musical que balanceen adherencia con naturalidad

8.7. Limitaciones del estudio

Es importante reconocer las limitaciones de este trabajo:

8.7.1. Limitaciones del dataset

Con solo 824 secuencias finales (reducido desde 1,071 clips originales debido a disponibilidad de videos de YouTube), el dataset EMOPIA extendido es relativamente pequeño para entrenar modelos generativos profundos desde cero. La pérdida del 23.0 % del dataset original por videos no disponibles representa una limitación significativa. Datasets más grandes y más estables podrían permitir mejor aprendizaje de relaciones complejas entre dimensiones de control.

8.7.2. Limitaciones de evaluación

La evaluación se basó principalmente en análisis automático. La percepción humana de adherencia al condicionamiento podría diferir significativamente de las métricas algorítmicas, particularmente para dimensiones subjetivas como emoción.

8.7.3. Limitaciones de alcance

El estudio se enfocó exclusivamente en música de piano pop. La generalización a otros géneros, instrumentos, o estilos requeriría validación adicional.

8.8. Contribuciones y valor del trabajo

A pesar de las limitaciones en adherencia absoluta, este trabajo realiza contribuciones significativas:

8.8.1. Primera integración de tonalidad en EMOPIA

La extensión exitosa del dataset EMOPIA con información de tonalidad algorítmica proporciona un recurso valioso para la comunidad. El dataset extendido permite investigación futura en generación musical con conciencia tonal.

8.8.2. Análisis sistemático de métodos de condicionamiento

La evaluación comprehensiva de diferentes métodos de inferencia proporciona insights valiosos sobre las capacidades y limitaciones actuales del condicionamiento en modelos autoregresivos.

8.8.3. Marco de evaluación reproducible

El pipeline de evaluación desarrollado establece un framework para evaluar sistemáticamente adherencia multiparamétrica, facilitando comparaciones futuras.

8.8.4. Identificación de desafíos fundamentales

Al revelar las limitaciones de los enfoques actuales, este trabajo identifica desafíos fundamentales en generación musical controlable que deben abordarse para avanzar el campo.

8.9. Conclusión de la discusión

Los resultados obtenidos validan parcialmente el sistema extendido desarrollado. Se logró exitosamente extender el dataset EMOPIA con información de tonalidad (Objetivo 1), adaptar y entrenar el modelo Transformer extendido con excelente convergencia en la dimensión de tonalidad (Objetivo 2), implementar múltiples métodos de inferencia multiparamétrica (Objetivo 3), desarrollar un pipeline de evaluación comprehensivo (Objetivo 4), y realizar una comparación cuantitativa reveladora entre modelos (Objetivo 5).

Sin embargo, los niveles de adherencia categórica observados indican que el control preciso sobre la generación musical sigue siendo un desafío abierto. Las altas correlaciones tonales y la convergencia excepcional durante entrenamiento sugieren que el modelo sí aprendió representaciones significativas de tonalidad, pero las expresa de manera probabilística y contextual en lugar de determinística.

Estos hallazgos no invalidan el enfoque, sino que revelan la complejidad inherente en la generación musical controlable. La música, como forma de arte, resiste reducción a parámetros independientes. El verdadero avance puede residir no en lograr control absoluto, sino en desarrollar métodos que respeten la naturaleza holística y contextual de la expresión musical mientras proporcionan influencia significativa sobre características deseadas.

El sistema extendido representa un paso importante hacia generación musical más controlable, estableciendo fundamentos técnicos y metodológicos sobre los cuales futura investigación puede construir. La integración exitosa de teoría musical computacional (tonalidad) con aprendizaje profundo demuestra el valor de enfoques híbridos que combinan conocimiento de dominio con métodos basados en datos.

CAPÍTULO 9

Conclusiones

El presente trabajo de graduación logró extender exitosamente el sistema de generación de música de piano EMOPIA mediante la incorporación de información de tonalidad como variable de control adicional y la implementación de métodos de inferencia multiparamétrica. A continuación se presentan las conclusiones específicas derivadas del cumplimiento de cada objetivo planteado:

1. **Se extendió exitosamente el dataset EMOPIA con información de tonalidad musical mediante el algoritmo de Krumhansl-Schmuckler.** De los 1,071 clips MIDI originales, se recuperaron 854 clips de audio (79.7 %) debido a videos no disponibles en YouTube, y de estos se procesaron exitosamente 824 archivos (96.5 % de los clips recuperados, 77.0 % del dataset original). El resultado es un dataset tokenizado con 9 dimensiones que incluye un vocabulario de 25 tokens de tonalidad (24 tonalidades musicales más un token de continuación). La integración se realizó de manera natural dentro del pipeline de Compound Word Transformer, manteniendo compatibilidad con la estructura original del dataset mientras se agregaba información musical teóricamente fundamentada.
2. **Se adaptó correctamente la arquitectura del Transformer de EMOPIA para procesar la novena dimensión de tonalidad.** El modelo extendido con 39,168,153 parámetros entrenables incorporó exitosamente embeddings de 128 dimensiones para la variable de tonalidad, manteniendo compatibilidad retroactiva con modelos de 8 dimensiones. Durante el entrenamiento en 875 épocas, la dimensión de tonalidad alcanzó una convergencia del 98.46 %, superando a dimensiones estructurales fundamentales como tempo (96.64 %) y pitch (95.67 %), lo que demuestra la efectividad de la integración arquitectónica.
3. **Se implementaron cuatro métodos distintos de inferencia multiparamétrica funcionales.** Los modos normal, determinístico, primed y forced proporcionan diferentes niveles de control sobre la generación musical, desde sampling estocástico hasta forzado directo de tokens. El sistema de configuración flexible permite ajustar tempera-

tura y nucleus sampling tanto globalmente como por token individual, proporcionando 11 argumentos de línea de comandos para control granular de la generación.

4. **Se desarrolló un pipeline de evaluación automatizado capaz de cuantificar adherencia al condicionamiento.** El pipeline ejecutó exitosamente 192 experimentos sistemáticos en 1.37 horas, aplicando tres categorías de métricas: adherencia categórica para tokens discretos, métricas de distancia numérica para valores continuos, y correlación tonal de Krumhansl-Schmuckler para evaluar coherencia tonal. El sistema proporciona análisis estadístico completo incluyendo distribuciones, correlaciones y agregaciones por múltiples dimensiones.
5. **La comparación cuantitativa reveló patrones complejos de adherencia entre el modelo extendido y el original.** Aunque las tasas de adherencia categórica fueron moderadas (máximo 50 % para emoción), las métricas de correlación tonal alcanzaron valores prometedores (hasta 0.76), sugiriendo que el modelo aprendió representaciones significativas de tonalidad expresadas probabilísticamente. El modelo pretrained mostró mayor adherencia emocional en modo determinístico (50 % vs 45.83 %) pero peor desempeño en modo primed (18.75 % vs 35.42 %), revelando trade-offs entre conocimiento previo y receptividad al condicionamiento.
6. **El trabajo contribuye a resolver el problema de control limitado en generación musical** proporcionando tres avances concretos: (a) un dataset público extendido que integra información de teoría musical con anotaciones emocionales existentes, (b) una arquitectura y métodos de inferencia que demuestran la viabilidad técnica del condicionamiento multiparamétrico incluyendo tonalidad, y (c) un marco de evaluación reproducible que establece métricas y metodologías para cuantificar adherencia al condicionamiento en múltiples dimensiones simultáneamente.
7. **Los resultados identifican que el control preciso sobre generación musical autoregresiva permanece como un desafío fundamental.** La naturaleza probabilística de los modelos Transformer, los conflictos potenciales entre restricciones múltiples, y la importancia del contexto musical para interpretación coherente limitan la adherencia exacta a valores objetivo. Sin embargo, las altas correlaciones tonales y la excelente convergencia durante entrenamiento validan que la incorporación de variables de teoría musical en modelos de aprendizaje profundo es un enfoque prometedor que merece investigación continua.
8. **El sistema extendido establece una base sólida para futura investigación en generación musical controlable.** La integración exitosa de tonalidad demuestra que variables de teoría musical computacional pueden enriquecer modelos generativos basados en datos. Los métodos de evaluación desarrollados permiten comparación sistemática de enfoques futuros. Las limitaciones identificadas proporcionan dirección clara para mejoras: métodos de condicionamiento jerárquico, arquitecturas especializadas para control musical, y métricas de evaluación que balanceen adherencia con coherencia musical.

En síntesis, este trabajo cumplió con el objetivo general de extender el sistema EMO-PIA con una variable de control adicional fundamentada en teoría musical e implementar

métodos de inferencia multiparamétrica. Aunque el control absoluto sobre la generación permanece elusivo, los avances técnicos y metodológicos realizados representan contribuciones concretas al campo de la generación musical con inteligencia artificial, estableciendo fundamentos para sistemas futuros que puedan ofrecer mayor controlabilidad mientras preservan la expresividad y coherencia musical.

CAPÍTULO 10

Recomendaciones

Con base en los hallazgos, limitaciones y desafíos identificados durante el desarrollo de este trabajo de graduación, se presentan las siguientes recomendaciones para investigaciones futuras en el campo de la generación musical controlable con modelos de aprendizaje profundo:

10.1. Recomendaciones sobre el dataset y preprocesamiento

1. Desarrollar estrategias de preservación y ampliación del dataset. Dado que el 23 % del dataset original no pudo ser recuperado debido a videos no disponibles en YouTube, se recomienda:

- Crear repositorios institucionales estables para datasets musicales académicos que no dependan de plataformas comerciales
- Implementar protocolos de respaldo periódico de datasets basados en contenido web
- Explorar técnicas de augmentación de datos específicas para música (transposición, variaciones rítmicas, cambios de tempo) para compensar pérdidas y ampliar el dataset
- Considerar la creación de datasets sintéticos complementarios usando el modelo entrenado para bootstrap

2. Ampliar el dataset EMOPIA extendido con más ejemplos y géneros. Las 824 secuencias finales son insuficientes para capturar la complejidad del condicionamiento multiparamétrico. Se recomienda:

- Incorporar datasets adicionales de piano con anotaciones emocionales
- Extender la metodología a otros instrumentos y ensambles para validar generalización

- Colaborar con músicos profesionales para crear datasets específicamente diseñados para evaluar controlabilidad
 - Implementar anotación semi-automática usando el modelo entrenado para acelerar la expansión del dataset
- 3. Mejorar la robustez del pipeline de detección de tonalidad.** Para los archivos excluidos por estructuras musicales atípicas, se recomienda:
- Implementar algoritmos de detección de tonalidad más robustos que manejen modulaciones y ambigüedad tonal
 - Desarrollar métodos de detección jerárquica que identifiquen tonalidad por secciones
 - Validar detecciones automáticas con anotaciones manuales de expertos en teoría musical

10.2. Recomendaciones sobre arquitectura y entrenamiento

- 1. Diseñar arquitecturas especializadas para generación musical controlable.** Los Transformers genéricos presentan limitaciones para control preciso. Se recomienda explorar:
 - Arquitecturas con módulos de atención especializados para diferentes escalas temporales (nota, compás, frase, sección)
 - Modelos con representaciones latentes disentangled que separen factores de control independientes
 - Incorporación de inductive biases musicales en la arquitectura (e.g., invariancia a transposición)
 - Arquitecturas híbridas que combinen procesamiento simbólico con aprendizaje profundo
- 2. Investigar estrategias de entrenamiento que mejoren la controlabilidad.** El entrenamiento estándar no optimiza para adherencia al condicionamiento. Se recomienda:
 - Desarrollar funciones de pérdida que penalicen explícitamente desviaciones de las condiciones especificadas
 - Implementar curriculum learning que progresivamente incremente la complejidad del condicionamiento
 - Explorar técnicas de meta-learning para adaptación rápida a nuevas restricciones
 - Investigar entrenamiento adversarial con discriminadores especializados por dimensión de control
- 3. Optimizar el balance entre preentrenamiento y fine-tuning.** Los trade-offs observados entre modelos scratch y pretrained sugieren:
 - Investigar técnicas de preentrenamiento específicas para música que preserven controlabilidad

- Desarrollar métodos de fine-tuning que mantengan conocimiento previo mientras mejoran receptividad al condicionamiento
- Explorar adapter layers o LoRA para modificación eficiente de modelos preentrenados
- Estudiar el impacto del tamaño y diversidad del dataset de preentrenamiento en la controlabilidad final

10.3. Recomendaciones sobre métodos de inferencia

1. **Desarrollar métodos de condicionamiento más sofisticados.** Los métodos token-level implementados mostraron limitaciones significativas. Se recomienda investigar:
 - **Condicionamiento jerárquico:** Especificar restricciones a nivel de estructura musical (intro, verso, coro) en lugar de tokens individuales
 - **Condicionamiento por ejemplos:** Usar fragmentos de referencia que el modelo pueda imitar selectivamente
 - **Condicionamiento iterativo:** Refinar generaciones mediante múltiples pasadas con feedback del cumplimiento de restricciones
 - **Condicionamiento probabilístico:** Especificar distribuciones objetivo en lugar de valores exactos
2. **Implementar métodos de muestreo conscientes del contexto musical.** El sampling actual ignora estructura musical. Se recomienda:
 - Desarrollar técnicas de muestreo que respeten cadencias y progresiones armónicas
 - Implementar beam search con heurísticas musicales para exploración más informada
 - Crear métodos de muestreo adaptativos que ajusten temperatura según el contexto musical
 - Investigar técnicas de backtracking para corregir decisiones que violen restricciones posteriores
3. **Explorar enfoques de generación no autoregresivos.** La naturaleza secuencial limita el control global. Se recomienda evaluar:
 - Modelos de difusión para música que permitan condicionamiento global más natural
 - Variational Autoencoders con espacios latentes interpretables para cada dimensión de control
 - Modelos basados en energía que optimicen satisfacción conjunta de restricciones
 - Enfoques híbridos que combinen generación global con refinamiento local

10.4. Recomendaciones sobre evaluación

1. **Desarrollar métricas de evaluación más comprehensivas.** La adherencia categórica es insuficiente para capturar calidad musical. Se recomienda:

- Diseñar métricas que balanceen adherencia con coherencia y naturalidad musical
- Implementar métricas perceptuales basadas en modelos cognitivos de percepción musical
- Desarrollar métricas específicas por dimensión que capturen adherencia parcial o aproximada
- Crear benchmarks estandarizados para comparación entre sistemas

2. **Incorporar evaluación humana sistemática.** Las métricas automáticas pueden no reflejar percepción real. Se recomienda:

- Diseñar estudios perceptuales con músicos y no-músicos para evaluar controlabilidad percibida
- Implementar plataformas de crowdsourcing especializadas para evaluación musical
- Desarrollar protocolos de evaluación que separen aspectos técnicos de preferencias estéticas
- Correlacionar métricas automáticas con evaluaciones humanas para validar su relevancia

3. **Evaluar aplicabilidad en contextos reales.** La evaluación actual es puramente técnica. Se recomienda:

- Colaborar con compositores para evaluar utilidad en flujos de trabajo creativos reales
- Desarrollar interfaces de usuario que faciliten especificación intuitiva de restricciones
- Estudiar casos de uso específicos (educación musical, terapia, entretenimiento) con métricas apropiadas
- Evaluar eficiencia computacional y viabilidad de deployment en aplicaciones interactivas

10.5. Recomendaciones metodológicas

1. **Adoptar enfoques interdisciplinarios.** La integración exitosa de teoría musical sugiere valor en colaboración interdisciplinaria. Se recomienda:

- Formar equipos que incluyan expertos en IA, teoría musical, cognición y composición
- Incorporar conocimiento de psicoacústica y percepción musical en el diseño de sistemas

- Colaborar con musicólogos computacionales para identificar representaciones más naturales
 - Integrar perspectivas de HCI para mejorar usabilidad de sistemas de generación
2. **Priorizar reproducibilidad y acceso abierto.** Para acelerar el progreso del campo, se recomienda:
- Publicar código, modelos entrenados y datasets procesados en repositorios permanentes
 - Documentar exhaustivamente decisiones de diseño y parámetros experimentales
 - Crear tutoriales y demos interactivos para facilitar adopción por la comunidad
 - Establecer competencias y benchmarks comunitarios para generación musical controlable
3. **Considerar implicaciones éticas y artísticas.** El desarrollo de IA musical plantea cuestiones importantes. Se recomienda:
- Estudiar el impacto de estos sistemas en la creatividad y práctica musical humana
 - Desarrollar guidelines éticos para atribución y uso de música generada por IA
 - Investigar sesgos en datasets y su propagación en sistemas generativos
 - Explorar modelos de colaboración humano-IA que potencien en lugar de reemplazar creatividad

10.6. Direcciones prioritarias para investigación inmediata

Basándose en los hallazgos de este trabajo, se identifican las siguientes áreas como prioritarias para avance inmediato del campo:

1. **Métodos de condicionamiento jerárquico y contextual** que superen las limitaciones del control token-by-token observadas en este estudio.
2. **Datasets musicales grandes y diversos con anotaciones múltiples** que permitan entrenamiento robusto de modelos controlables.
3. **Arquitecturas neuronales diseñadas específicamente para música** que incorporen inductive biases apropiados para el dominio.
4. **Frameworks de evaluación holísticos** que capturen tanto aspectos técnicos como perceptuales y artísticos de la generación musical.
5. **Aplicaciones prácticas y casos de uso concretos** que demuestren valor real de la generación controlable más allá del contexto académico.

Estas recomendaciones buscan no solo abordar las limitaciones identificadas en este trabajo, sino también establecer una agenda de investigación que avance el campo hacia sistemas de generación musical verdaderamente controlables y útiles. La integración exitosa de teoría

musical computacional demostrada en este trabajo sugiere que el progreso futuro requerirá colaboración interdisciplinaria continua y voluntad de cuestionar paradigmas establecidos en favor de enfoques más apropiados para el dominio musical.

CAPÍTULO 11

Bibliografía

- [1] H.-T. Hung, J. Ching, S. Doh, N. Kim, J. Nam e Y.-H. Yang, “EMOPIA: A multi-modal pop piano dataset for emotion recognition and emotion-based music generation,” *Proceedings of the International Society for Music Information Retrieval Conference*, 2021. DOI: 10.48550/arXiv.2108.01374. dirección: <https://doi.org/10.48550/arXiv.2108.01374>.
- [2] H.-T. Hung, J. Ching, S. Doh, N. Kim, J. Nam e Y.-H. Yang, *EMOPIA: A Multi-Modal Pop Piano Dataset For Emotion Recognition and Emotion-based Music Generation*, ver. 2.2, Dataset version 2.2 containing 1,071 MIDI clips from 387 unique YouTube videos, Zenodo, sep. de 2022. DOI: 10.5281/zenodo.5257995. dirección: <https://zenodo.org/records/5257995>.
- [3] W.-Y. Hsiao, J.-Y. Liu, Y.-C. Yeh e Y.-H. Yang, *Compound Word Transformer: Learning to Compose Full-Song Music over Dynamic Directed Hypergraphs*, 2021. arXiv: 2101.02402 [cs.SD]. dirección: <https://arxiv.org/abs/2101.02402>.
- [4] S. Inc., *Suno: Generative AI Music Platform*, <https://suno.com/>, Accessed October 2025, 2025.
- [5] E. Inc., *Eleven Music: Text-to-Music Generation Service*, <https://elevenlabs.io/music>, Accessed October 2025, 2025.
- [6] Z. Ning, H. Chen, Y. Jiang et al., *DiffRhythm: Blazingly Fast and Embarrassingly Simple End-to-End Full-Length Song Generation with Latent Diffusion*, 2025. arXiv: 2503.01183 [eess.AS]. dirección: <https://arxiv.org/abs/2503.01183>.
- [7] A. Williams y M. Barthet, “Towards Music Industry 5.0: Perspectives on Artificial Intelligence,” en *Workshop on AI for Music, Association for the Advancement of Artificial Intelligence*, Association for the Advancement of Artificial Intelligence, Philadelphia (USA), United States, mar. de 2025. dirección: <https://hal.science/hal-04943901>.
- [8] J. A. Russell, “A Circumplex Model of Affect,” *Journal of Personality and Social Psychology*, vol. 39, n.º 6, págs. 1161-1178, 1980. dirección: <https://pdodds.w3.uvm.edu/research/papers/others/1980/russell1980a.pdf>.

- [9] J. Posner, J. A. Russell y B. S. Peterson, "The circumplex model of affect: an integrative approach to affective neuroscience, cognitive development, and psychopathology," *Development and Psychopathology*, vol. 17, n.º 3, págs. 715-734, 2005. dirección: <https://pubmed.ncbi.nlm.nih.gov/16262989/>.
- [10] T. Eerola y J. K. Vuoskoski, "A comparison of the discrete and dimensional models of emotion in music," *Psychology of Music*, vol. 39, n.º 1, págs. 18-49, 2011. dirección: <https://durham-repository.worktribe.com/output/1477246/a-comparison-of-the-discrete-and-dimensional-models-of-emotion-in-music>.
- [11] F. Gu et al., "A Model for Basic Emotions Using Observations of Behavior in *Drosophila*," *Frontiers in Psychology (Emotion Science)*, 2019. dirección: <https://pmc.ncbi.nlm.nih.gov/articles/PMC6491740/>.
- [12] Y. Liu et al., "Effects of Musical Tempo on Musicians' and Non-musicians' Emotional Experience When Listening to Music," *Frontiers in Psychology*, vol. 9, pág. 2118, 2018. dirección: <https://www.frontiersin.org/journals/psychology/articles/10.3389/fpsyg.2018.02118/full>.
- [13] J. C. Hailstone et al., "It's not what you play, it's how you play it": Timbre affects perception of emotion in music," *Quarterly Journal of Experimental Psychology*, vol. 62, n.º 11, págs. 2141-2155, 2009. dirección: <https://pubmed.ncbi.nlm.nih.gov/19391047/>.
- [14] A. Micallef Grimaud y T. Eerola, "Emotional expression through musical cues: A comparison of production and perception approaches," *PLOS ONE*, vol. 17, n.º 12, e0279605, 2022. dirección: <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0279605>.
- [15] H. Zhang et al., "ATEPP: A Dataset of Automatically Transcribed Expressive Piano Performance," 2022. dirección: <https://archives.ismir.net/ismir2022/paper/000053.pdf>.
- [16] C. Portalés-Ricart et al., "Research in computational expressive music performance and popular music production: A potential field of application?" *MDPI Multimedia Tools and Applications*, vol. 81, págs. 4077-4102, 2022. dirección: <https://www.mdpi.com/2414-4088/7/2/15>.
- [17] B. Roth et al., "A machine learning approach to discover rules for expressive performance actions in jazz guitar music," *Frontiers in Psychology*, vol. 7, pág. 1965, 2016. dirección: <https://www.frontiersin.org/journals/psychology/articles/10.3389/fpsyg.2016.01965/full>.
- [18] G. Boenn, M. Brain, M. De Vos y J. Ffitch, "Computational Music Theory," *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 8, n.º 4, págs. 27-34, jun. de 2021. DOI: 10.1609/aiide.v8i4.12559. dirección: <https://ojs.aaai.org/index.php/AIIDE/article/view/12559>.
- [19] C. L. Krumhansl y E. J. Kessler, "Tracing the dynamic changes in perceived tonal organization in a spatial representation of musical keys," *Psychological Review*, vol. 89, n.º 4, págs. 334-368, 1982. DOI: 10.1037/0033-295X.89.4.334.
- [20] B. W. Frankland y A. J. Cohen, "Using the Krumhansl and Schmuckler Key-Finding Algorithm to Quantify the Effects of Tonality in the Interpolated-Tone Pitch-Comparison Task," *Music Perception*, vol. 14, n.º 1, págs. 57-83, oct. de 1996, ISSN: 0730-7829. DOI: 10.2307/40285709. dirección: <https://doi.org/10.2307/40285709>.

- [21] D. Temperley, “Music and probability,” 2007.
- [22] M. S. Cuthbert y C. T. Ariza, “music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data,” en *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR 2010)*, International Society for Music Information Retrieval, 2010, págs. 637-642, ISBN: 9789039353813.
- [23] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Networks*, vol. 61, págs. 85-117, 2015, ISSN: 0893-6080. DOI: 10.1016/j.neunet.2014.09.003. dirección: <https://arxiv.org/abs/1404.7828>.
- [24] A. Vaswani, N. Shazeer, N. Parmar et al., “Attention is all you need,” *Advances in Neural Information Processing Systems*, vol. 30, 2017. dirección: <https://arxiv.org/abs/1706.03762>.
- [25] S. Ji, J. Luo y X. Yang, “A Comprehensive Survey on Deep Music Generation: Multi-level Representations, Algorithms, Evaluations, and Future Directions,” *arXiv preprint arXiv:2011.06801*, 2020. dirección: <https://arxiv.org/abs/2011.06801>.
- [26] P. Lisena, A. Meroño-Peñuela y R. Troncy, “MIDI2vec: Learning MIDI embeddings for reliable prediction of symbolic music metadata,” *Semantic Web Journal*, 2020. dirección: <https://www.semantic-web-journal.net/system/files/swj2725.pdf>.
- [27] C.-Z. A. Huang et al., “Music Transformer,” *arXiv preprint arXiv:1809.04281*, 2018. dirección: <https://arxiv.org/abs/1809.04281>.
- [28] G. Hadjeres, F. Pachet y F. Nielsen, “DeepBach: a steerable model for Bach chorales generation,” vol. 70, págs. 1362-1371, 2017. dirección: <https://arxiv.org/abs/1612.01010>.
- [29] H.-W. Dong et al., “MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment,” *arXiv preprint arXiv:1709.06298*, 2018. dirección: <https://arxiv.org/abs/1709.06298>.
- [30] L. N. Ferreira y J. Whitehead, *Learning to Generate Music With Sentiment*, 2021. arXiv: 2103.06125 [cs.LG]. dirección: <https://arxiv.org/abs/2103.06125>.
- [31] S. Dai, Z. Jin, C. Gomes y R. B. Dannenberg, “Controllable deep melody generation via hierarchical music structure representation,” *arXiv preprint arXiv:2109.00663*, 2021. dirección: <https://arxiv.org/abs/2109.00663>.
- [32] A. Holtzman, J. Buys, L. Du, M. Forbes e Y. Choi, “The Curious Case of Neural Text Degeneration,” *arXiv preprint arXiv:1904.09751*, 2019. dirección: <https://arxiv.org/abs/1904.09751>.
- [33] Z. Xiong et al., “A Comprehensive Survey for Evaluation Methodologies of AI-Generated Music,” *arXiv preprint arXiv:2308.13736*, 2023. dirección: <https://arxiv.org/abs/2308.13736>.
- [34] L.-C. Yang y A. Lerch, “On the evaluation of generative models in music,” 2018. dirección: https://musicinformatics.gatech.edu/wp-content_nondefault/uploads/2018/11/postprint.pdf.
- [35] J. de Berardinis et al., “The Music Annotation Pattern,” 2022. dirección: <https://arxiv.org/abs/2304.00988>.
- [36] S. Koelsch, “Brain correlates of music-evoked emotions,” *Nature Reviews Neuroscience*, vol. 15, págs. 170-180, 2014. dirección: https://www.stefan-koelsch.de/papers/koelsch_2014_brain_music_emotion.pdf.

- [37] V. N. Salimpoor et al., “Anatomically distinct dopamine release during anticipation and experience of peak emotion to music,” *Nature Neuroscience*, vol. 14, n.º 2, págs. 257-262, 2011. dirección: <https://pubmed.ncbi.nlm.nih.gov/21217764/>.
- [38] C. Kang, P. Lu, B. Yu et al., “EmoGen: Eliminating subjective bias in emotional music generation,” *arXiv preprint arXiv:2307.01229*, 2023. DOI: 10.48550/arXiv.2307.01229. dirección: <https://doi.org/10.48550/arXiv.2307.01229>.
- [39] K. Zheng, Y. Yang, M. Zhang y X. Huang, “EmotionBox: A music-element-driven emotional music generation system based on music psychology,” *Frontiers in Psychology*, vol. 13, pág. 841926, 2022. DOI: 10.3389/fpsyg.2022.841926. dirección: <https://doi.org/10.3389/fpsyg.2022.841926>.
- [40] H. A. Modran, T. Chamunorwa, D. Ursutiu, C. Samoilă y H. Hedesiu, “Using deep learning to recognize therapeutic effects of music based on emotions,” *Sensors*, vol. 23, n.º 2, pág. 986, 2023. doi: 10.3390/s23020986. dirección: <https://doi.org/10.3390/s23020986>.
- [41] M. Ramaswamy, J. L. Philip, V. Priya et al., “Therapeutic use of music in neurological disorders: A concise narrative review,” *Heliyon*, vol. 10, n.º 16, e35564, 2024, ISSN: 2405-8440. DOI: 10.1016/j.heliyon.2024.e35564. dirección: <https://www.sciencedirect.com/science/article/pii/S2405844024115952>.
- [42] J. Eaton, M. Barthet y M. Sandler, “Biofeedback and adaptive music generation,” en *2014 IEEE International Conference on Multimedia and Expo Workshops (IC-MEW)*, IEEE, 2014, págs. 1-6. dirección: <https://ieeexplore.ieee.org/abstract/document/6890674>.
- [43] G. Nierhaus, *Algorithmic Composition: Paradigms of Automated Music Generation*. Springer Vienna, 2009, ISBN: 978-3-211-75539-6. DOI: 10.1007/978-3-211-75540-2. dirección: <https://link.springer.com/book/10.1007/978-3-211-75540-2>.
- [44] S. Oore, I. Simon, S. Dieleman, D. Eck y K. Simonyan, “This time with feeling: Learning expressive musical performance,” *Neural Computing and Applications*, vol. 32, n.º 4, págs. 955-967, 2020. dirección: <https://magenta.tensorflow.org/performance-rnn>.
- [45] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford e I. Sutskever, “Jukebox: A generative model for music,” en *arXiv preprint arXiv:2005.00341*, 2020. dirección: <https://arxiv.org/abs/2005.00341>.
- [46] yt-dlp contributors, *yt-dlp: A youtube-dl fork with additional features and fixes*, <https://github.com/yt-dlp/yt-dlp>, Command-line program to download videos from YouTube and other platforms. Accessed January 2025, 2025.
- [47] FFmpeg team, *FFmpeg: A complete, cross-platform solution to record, convert and stream audio and video*, <https://www.ffmpeg.org/>, Multimedia framework for audio and video processing. Accessed January 2025, 2025.
- [48] YatingMusic, *Compound Word Transformer: Dataset Processing Documentation*, <https://github.com/YatingMusic/compound-word-transformer>, GitHub repository containing data preprocessing pipeline for symbolic music generation. Accessed January 2025, 2021.
- [49] R. Inc., *RunPod: The Cloud Built for AI*, <https://www.runpod.io/>, Cloud GPU infrastructure platform for AI workloads. Accessed January 2025, 2025.

- [50] Amazon Web Services, *Amazon EC2 p3.2xlarge Instance Pricing*, <https://instances.vantage.sh/aws/ec2/p3.2xlarge>, GPU instance with 8 vCPUs, 61 GiB memory. Accessed January 2025, 2025.
- [51] Google Cloud, *Google Cloud Compute Engine Pricing*, <https://cloud.google.com/compute/all-pricing>, A2 instance family with NVIDIA GPUs. Accessed January 2025, 2025.
- [52] Microsoft Azure, *Azure NC-Series Virtual Machines Pricing*, <https://azure.microsoft.com/en-us/pricing/details/virtual-machines/windows/>, N-series VMs with NVIDIA Tesla GPUs. Accessed January 2025, 2025.
- [53] Vast.ai, *Vast.ai: GPU Cloud Computing Pricing*, <https://vast.ai/pricing>, Peer-to-peer GPU rental marketplace. Accessed January 2025, 2025.
- [54] Lambda Labs, *Lambda Labs GPU Cloud Pricing*, <https://lambda.ai/pricing>, Cloud GPU infrastructure for AI and ML workloads. Accessed January 2025, 2025.
- [55] S. Böck, F. Korzeniowski, J. Schlüter, F. Krebs y G. Widmer, “madmom: a new Python library for music signal processing,” en *Proceedings of the 24th ACM International Conference on Multimedia*, Python audio and music signal processing library with focus on music information retrieval tasks, Amsterdam, The Netherlands, 2016, págs. 1174-1178. DOI: 10.1145/2964284.2973795. dirección: <https://github.com/CPJKU/madmom>.
- [56] C. L. Krumhansl, *Cognitive Foundations of Musical Pitch*. New York: Oxford University Press, 1990, ISBN: 978-0-19-505475-8.
- [57] K. Hevner, “The affective character of the major and minor modes in music,” *The American Journal of Psychology*, vol. 47, n.º 1, págs. 103-118, 1935. DOI: 10.2307/1416710.
- [58] F. Lerdahl y R. Jackendoff, *A Generative Theory of Tonal Music*. Cambridge, MA: MIT Press, 1983, ISBN: 978-0-262-12094-4.
- [59] A. Roberts, J. Engel, C. Raffel, C. Hawthorne y D. Eck, “A hierarchical latent vector model for learning long-term structure in music,” en *Proceedings of the 35th International Conference on Machine Learning*, ép. Proceedings of Machine Learning Research, vol. 80, PMLR, 2018, págs. 4364-4373. dirección: <http://proceedings.mlr.press/v80/roberts18a.html>.

