# Auto-Regressive vs Flow-Matching: a Comparative Study of Modeling Paradigms for Text-to-Music Generation

**Anonymous authors**
**Paper under double-blind review**

## Abstract

Recent progress in text-to-music generation has enabled models to synthesize high-quality musical segments, full compositions, and even respond to fine-grained control signals, e.g. chord progressions. State-of-the-art (SOTA) systems differ significantly across many dimensions, such as training datasets, modeling paradigms, and architectural choices. This diversity complicates efforts to evaluate models fairly and pinpoint which design choices most influence performance. While factors like data and architecture are important, in this study we focus exclusively on the modeling paradigm. We conduct a systematic empirical analysis to isolate its effects, offering insights into associated trade-offs and emergent behaviors that can guide future text-to-music generation systems. Specifically, we compare the two arguably most common modeling paradigms: Auto-Regressive decoding and Conditional Flow-Matching. We conduct a controlled comparison by training all models from scratch using identical datasets, training configurations, and similar backbone architectures. Performance is evaluated across multiple axes, including generation quality, robustness to inference configurations, scalability, adherence to both textual and temporally aligned conditioning, and editing capabilities in the form of audio inpainting. This comparative study sheds light on distinct strengths and limitations of each paradigm, providing actionable insights that can inform future architectural and training decisions in the evolving landscape of text-to-music generation. Audio sampled examples are available at: `https://huggingface.co/spaces/Unk-Uname/ARvsFM`

## 1 Introduction

Unlike text and vision domains, the audio domain, and music generation in particular, has not yet converged on a dominant modeling approach. While both Auto-Regressive (AR) and non-AR methods have shown strong results, the trade-offs between them remain under explored (Zhu et al., 2023; Pathariya et al., 2024; Li et al., 2025). In natural language processing, the dominant modeling paradigm is AR generation over discrete token sequences (Touvron et al., 2023; Liu et al., 2024). In computer vision, leading models are typically non-AR, relying on diffusion or flow-matching processes over continuous latent spaces (Podell et al., 2023; Liu et al., 2023). However, it is not clear what approach should we follow for music and audio generation. Copet et al. (2023); Agostinelli et al. (2023) demonstrate impressive performance following the AR approach using discrete audio representation. In contrast, Chen et al. (2024); Lan et al. (2024) follows the diffusion and flow matching approaches and also show impressive performance. Lastly, Li et al. (2024); Bai et al. (2024) proposed hybrid methods utilizing AR and non-AR methods.

While a growing number of systems have demonstrated compelling capabilities in text-conditioned music generation, it is unclear what fundamentally accounts for performance differences across models. Variations in training data, latent representations, architecture design, and optimization procedures often confound evaluation. As a result, there is little consensus on whether improvements arise from the modeling paradigm itself or from external factors, like the training data or architectural choices. These contrasts underscore the need for systematic comparison in audio modeling, where foundational choices are still in flux.

Table 1: Concise summary of our conclusions: AR vs FM.

| **Axis** | **Approach** | **Key difference** |
|---|---|---|
| Text-to-music fidelity *(Sec. 5.1)* | AR | AR shows slightly better objective scores; robust to changes in frame rate. FM showed less robustness to frame-rate changes, especially in the VAE-based case. |
| Temporal control adherence *(Sec. 5.2)* | AR | Better chord IoU and melody similarity; both paradigms lose fidelity using temporally aligned controls. |
| Music inpainting *(Sec. 5.3)* | FM (supervised) | FM has the highest human evaluation scores; AR has lowest FAD but often generate bad transitions; FM (ZS) is unstable. |
| Inference speed and batch scaling *(Sec. 5.4)* | Setting dependent | AR wins at high batch sizes by utilizing key-value cache; FM could be faster than AR by sacrificing fidelity. |
| Sensitivity to training configuration *(Sec. 5.5)* | FM | FM reaches near-topline (Sec. 5.1) quality with small batches; AR needs larger batches to recover its topline scores. AR kept improving between 500K and 1M training steps, whereas FM gains tapered off. |

To mitigate that, we present a controlled empirical study comparing the two prominent and commonly used approaches for generative modeling in text-to-music generation: AR and Conditional Flow Matching (FM) (non-AR). All models are trained from scratch using the same training data, latent representations, and similar transformer model backbone architectures. We evaluate each modeling paradigm across multiple axes including perceptual quality, inference efficiency, robustness to training configuration, adherence to temporal control, and editing capabilities in the form of audio inpainting. This design isolates the modeling approach as the primary experimental variable.

Our results highlight consistent differences between the two paradigms. AR models exhibit slightly higher perceptual quality and stronger temporally-aligned control adherence, while flow-matching models show a potential to offer faster inference and greater flexibility for editing tasks. Such findings and additional observations outlined in this work provide practical guidance for selecting modeling paradigms in future music generation systems and are broadly covered in the following sections. Table 1 draws a summarized overview of our main conclusions.

## 2 Related Work

The idea of generating music through artificial intelligence has evolved significantly, beginning with rule-based symbolic systems (Pinkerton, 1956; Papadopoulos & Wiggins, 1999; Donnelly & Sheppard, 2011) and progressing to deep-learning approaches capable of synthesizing high-fidelity audio (Agostinelli et al., 2023; Huang et al., 2023; Copet et al., 2023; Li et al., 2024; Evans et al., 2024). Early experiments in AI-driven music creation focused on MIDI-based outputs (Huang & Wu, 2016), with systems like Jukedeck[1] offering genre-specific composition based on user-defined prompts. However, these early models struggled to capture the richness of human-composed music due to their reliance on pre-structured symbolic representations.

The introduction of deep learning revolutionized this field, first introducing deep Recurrent Neural Network systems Simon & Oore (2017); Mao et al. (2018), which further evolved to transformer based architectures, such as MuseNet [2] and MusicTransformer (Huang & Guo, 2019), demonstrating that transformers could generate stylistically coherent compositions. Subsequent works that followed moved beyond MIDI-based

[1] https://techcrunch.com/2015/12/07/jukedeck
[2] https://openai.com/index/musenet/

approaches to generate raw audio waveforms mainly do so using one of three generative paradigms: AR decoding, diffusion, and flow-matching.

A major breakthrough was introduced by JukeBox(Dhariwal et al., 2020), which incorporated both instrumental and vocal elements using AR decoding. Following this line of work, Borsos et al. (2023) introduce AudioLM, which first compresses audio into "semantic" and "acoustic" tokens and uses an AR Transformer to predict them. This enable the model to extend a musical excerpt without relying on any symbolic representation. Building on this idea, MusicLM (Agostinelli et al., 2023) adds a text-to-semantic stage and a hierarchical AR decoder, generating 24kHz music with noticeably better coherence and fidelity than JukeBox. MusicGen (Copet et al., 2023) simplifies the pipeline, encoding a 32kHz audio into EnCodec (Défossez et al., 2022) tokens and trains a single-stage AR Transformer to predict all token streams jointly, reducing latency while maintaining high prompt adherence and audio quality.

In parallel, inspired by recent success in text-to-image synthesis, diffusion models have emerged as an alternative paradigm, generating music by iteratively refining noise through a learned denoising process. Noise2Music (Huang et al., 2023) introduces a cascaded architecture that enables high-fidelity synthesis through progressive upsampling. MusicLDM (Chen et al., 2024), extends this approach by incorporating beat-synchronous augmentation, improving musical structure alignment with text prompts. StableAudio (Evans et al., 2024) further refines diffusion-based synthesis by introducing low-latency inference and high-resolution output (44.1kHz) offering long generation of full-length songs.

A more recent development is the adoption of flow-based generative models, which learn a continuous transformation from a simple distribution to the target audio distribution while conditioning on text. AudioBox (Vyas et al., 2023) introduces a FM framework capable of handling multiple audio modalities, including music, speech, and environmental sounds. JASCO (Tal et al., 2024) refines FM techniques for music generation, conditioning on textual descriptions and symbolic music features to enhance both coherence and controllability. MelodyFlow (Lan et al., 2024) optimizes single-stage FM models for high-fidelity text-guided music generation, improving both efficiency and musical structure adherence.

An additional timely development is the appearance of approaches combining AR with non-AR (Lam et al., 2023; Li et al., 2024; Bai et al., 2024), yet this falls outside of the scope of this work and will not be considered. Together, these approaches represent a diverse and rapidly evolving landscape in text-to-music generation. AR models continue to set strong baselines for musical structure and coherence, while non-AR methods, including Diffusion and FM models, offer promising alternatives for efficient generation and flexible control. Understanding how these paradigms compare under matched conditions remains an open challenge, motivating further exploration.

## 3 Background

### 3.1 Problem Setup and Formulation

Given an audio waveform $\boldsymbol{x} \in \mathbb{R}^{f_s \cdot t}$ of duration $t$ seconds, sampled at $f_s$ Hz, we assume access to a pre-trained latent representation model that encodes $\boldsymbol{x}$ into either (i) a continuous latent representation $\boldsymbol{z} \in \mathbb{R}^{D \times f_r \cdot t}$ or (ii) a discrete representation $\boldsymbol{q} \in \mathcal{S}^{N_q \times f_r \cdot t}$ (also known as audio tokenization or audio codec). Here, $f_r$ is the latent frame rate, $D$ the latent dimension, $\mathcal{S}$ is the set of discrete code indices and $N_q$ is the number of parallel code streams. Each discrete code stream has its own embedding table, also referred to as codebook. Our EnCodec model encodes 32kHz music to 50Hz multi-token stream composed of 4 codebook streams. The goal is to train a generative model that operates in this latent space, conditioned on a target textual description, using AR for discrete modeling or FM for continuous modeling. The generation could also be conditioned on other temporally aligned controls in addition to the textual description, e.g. chord progressions.

### 3.2 Auto-Regressive (AR) Decoding

The AR approach models the discrete latent sequence distribution using a causal transformer trained to predict the next token given past context. Given an audio segment $\boldsymbol{x}$, its textual description $c_{\text{txt}}$, and a

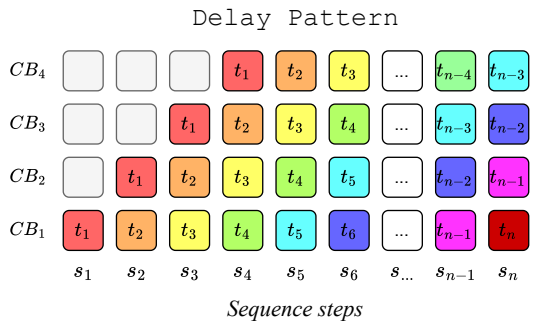


Figure 1: Multi-stream delay pattern modeling. Each row represents a single codebook stream, illustrating the applied delay pattern apperent in the shifting of $\text{CB}_j$ by $j-1$ sequence steps.

discrete latent representation of $\boldsymbol{x}$, $\boldsymbol{q} \in \mathcal{S}^{N_q \times f_r \cdot t}$, the model is trained to iteratively generate discrete tokens in an auto-regressive manner. Our discrete latent representation encodes the input waveform $\boldsymbol{x}$ to a sequence of $N_q > 1$ discrete streams that are obtained by utilizing Residual Vector Quantization (RVQ) (Zeghidour et al., 2021; Défossez et al., 2022). RVQ quantizes a continuous encoded latent $\boldsymbol{z} \in \mathbb{R}^{D \times f_r \cdot t}$ to $N_q$ streams of discrete tokens recursively quantizing the current residual. Formally, let $\hat{\boldsymbol{z}}_1$ be the quantized continuous latent representation of $\boldsymbol{z}$ for which each temporal entry was replaced with its closest, in terms of euclidean distance, vector in the $1^{\text{st}}$ codebook and let $\boldsymbol{q}_1$ be the corresponding indices of that sequence. Then, recursively applying quantization we can define $\forall j \in \{2, ..., N_q\} : \hat{\boldsymbol{z}}_j = \boldsymbol{z} - \sum_{l<j} \hat{\boldsymbol{z}}_l$ and obtain the corresponding indices stream $\boldsymbol{q}_j$. This iterative process then yields the discrete multi-stream representation $\boldsymbol{q} = \text{stack}([\boldsymbol{q}_1, ..., \boldsymbol{q}_{N_q}])$.

**Training with Delay Pattern.** Notice, at each timestep, one needs to predict $N_q$ different codes corresponding to a different codebooks. This begs the question, *how should we predict these codebooks?*. Naturally, this multi-stream representation dictates an inherent dependence between stream $j$ and all the ones proceeding it. Due to that, predicting $N_q$ codebooks in parallel at each timestep means independently sampling $N_q$ tokens, ignoring the inherently dependent structure. To mitigate that, we follow MusicGen (Copet et al., 2023) and train the model using a *delay pattern* to structure the multi-stream discrete representation. Instead of predicting all codebooks simultaneously, each stream is shifted by a predefined delay, enforcing a temporal offset that allows early predictions to condition later ones.

For a quantized sequence $\boldsymbol{q} \in \mathcal{S}^{N_q \times f_r \cdot t}$ with $N_q$ codebook streams, we impose a delay pattern structure on the multi-stream sequence to allow the $i^{\text{th}}$ codebook at timestep $j$ to be conditioned on all previous $< i$ codebook streams for all $\leq j$ timesteps, as depicted in Figure 1. Formally, denote $n = f_r \cdot t$, then $\forall i \in [N_q], j \in [n]$ we define a mapping $P : [N_q] \times [n] \to [N_q] \times [n + N_q - 1]$ s.t $P(i, j) = (i, j + i - 1)$. The model outputs a distribution over discrete tokens for each codebook stream across the temporal axis $\boldsymbol{p} \in \mathbb{R}^{N_q \times |\mathcal{S}| \times f_r \cdot t}$, and trained to minimize the cross-entropy objective:

$$\mathcal{L}_{\text{CE}}(\boldsymbol{q}, \boldsymbol{p}) = -\frac{1}{N_q \cdot |\mathcal{S}| \cdot f_r \cdot t} \sum_{k \in [N_q]} \sum_{j \in \mathcal{S}} \sum_{i \in [f_r \cdot t]} \mathbf{1}_{\{\text{q}_{\text{k,i}} = \text{j}\}} \cdot \log \boldsymbol{p}_{k,j,i}. \tag{1}$$

**Inference Process.** At inference time, the model generates tokens sequentially, following the same delay pattern used in training. Generation starts with an empty context, progressively sampling tokens while respecting the predefined temporal shifts between codebooks. Top-$k$ or top-$p$ sampling strategies are used during decoding, which are standard methods to control diversity in AR generation. The process continues iteratively until the full latent sequence is generated, after which it is decoded back into a waveform using the pretrained audio tokenizer.

### 3.3 Conditional Flow Matching (FM)

FM is an approach to training continuous normalizing flows by regressing a neural network onto a known vector field that generates a probability path (Lipman et al., 2022). Unlike diffusion models, which rely on

stochastic noise schedules and iterative denoising, FM learns a deterministic vector field to directly transform noise into data.

**Probability Path Definition.** We define a continuous transformation from a simple prior $p_0$ (e.g., $\mathcal{N}(0, I)$) to the data distribution $p_1$ using a time-dependent probability flow $\psi_\tau(y|y_1)$. The probability flow $\psi_\tau(y|y_1)$ defines a continuous path from $y_1$ to $y$ for which $\psi_\tau(y|y_1) = y \sim p_0$ at $\tau = 0$ and $\psi_\tau(y|y_1) = y_1 \sim p_1$ at $\tau = 1$.

Following the optimal transport setup, as defined by Lipman et al. (2022), we define a probability flow from $p_0$ to $p_1$ such that: (i) the mean and variance evolve linearly with time $\tau \in [0, 1]$; (ii) for $y_0 \sim p_0$ and $y_1 \sim p_1$, the probability path $\psi_\tau(y|y_1)$ and its corresponding vector field $\boldsymbol{v}_\tau(y|y_1)$ are given by:

$$\begin{aligned} \psi_\tau(y|y_1) &= (1 - (1 - \sigma_{\min})\tau)y + \tau y_1, \\ \boldsymbol{v}_\tau(y|y_1) &= \frac{y_1 - (1 - \sigma_{\min})y}{1 - (1 - \sigma_{\min})\tau}, \end{aligned} \tag{2}$$

where $\sigma_{\min}$ is some small constant to ensure numerical stability.

**Training Objective.** Rather than modeling the full marginal probability path, the model estimates the conditional vector field $\hat{\boldsymbol{v}}_\tau(y|y_1)$. Given a text-conditioned latent $y_1$ drawn from $p_1$, we train the model to minimize the mean squared error between the estimated and reference vector fields:

$$\mathcal{L}_{\text{FM}}(\hat{\boldsymbol{v}}_\tau(y|y_1), \boldsymbol{v}_\tau(y|y_1)) = \mathbb{E}\left[||\hat{\boldsymbol{v}}_\tau(y|y_1) - \boldsymbol{v}_\tau(y|y_1)||^2\right]. \tag{3}$$

Following Tal et al. (2024) we slightly modify this training objective by applying a $\tau$ dependent loss scaling, assuming a batch size of $B$ samples:

$$\mathcal{L}_{\text{FM}}(\hat{\boldsymbol{v}}_\tau(y|y_1), \boldsymbol{v}_\tau(y|y_1)) = \frac{1}{B} \sum_{i \in [B]} (1 + \tau_i) \cdot ||\hat{\boldsymbol{v}}_{\tau_i}(y|y_1) - \boldsymbol{v}_{\tau_i}(y|y_1)||^2. \tag{4}$$

**Inference Process.** Generation follows a non-AR iterative process, iteratively refining the vector field estimation $\hat{\boldsymbol{v}}(y_\tau|y_1)$ using an ODE solver. Given an initial sample $y_0 \sim p_0$, we update the state iteratively:

$$y_\tau = y_{(\tau - \Delta\tau)} + \Delta\tau \cdot \hat{\boldsymbol{v}}(y_\tau|y_1). \tag{5}$$

While various ODE solvers and sampling schedules could be applied during inference, in this study we only consider two representative sampling methods: Euler's method for a fixed-grid sampling and Dopri5 (Dormand & Prince, 1980) for a dynamic (adaptive-step) sampling.

## 4 Experimental Setup

Throughout this paper we consider text-to-music generation tasks. We also evaluate AR and FM considering temporally aligned conditioning for music generation and music inpainting. As we perform multiple experiments in this work, this section serves to define the common ground. Deviations from the shared experimental setup defined in this section would be explicitly described in each of the relevant subsections.

### 4.1 Data

We train our models on a private proprietary dataset containing tracks from Shutterstock [3] and Pond5 [4] data collections, which sums to roughly 20k hours of Mono 32kHz mixtures paired with textual descriptions. We evaluate the trained models on a different private proprietary data containing 162 hours of high-quality

[3] https://shutterstock.com/music
[4] https://pond5.com

Mono 32kHz mixtures paired with high quality captioning textual descriptions. We refrained from evaluating the models over MusicCaps as it has much lower quality in both audio and paired captioning, and we believe that our chosen evaluation set serves as a much more reliable one.

### 4.2 Models

**Input Representation.** We use EnCodec's (Défossez et al., 2022) quantized discrete representation for AR modeling and its continuous, pre-quantizer, latent for FM modeling. In addition, we train a VAE-GAN autoencoder following StableAudio's (Evans et al., 2024) open source recipe[5] to compare FM continuous modeling on both representations.

**Backbone Model.** For the backbone transformer architecture we use the open source implementation of MusicGen Copet et al. (2023) using a 400M parameters configuration, and T5 (Raffel et al., 2020) to obtain text embeddings and pass them via cross-attention layers as text conditions. For the FM case we perform slight modifications similarly to Tal et al. (2024) and include U-Net-like skip connections. For further details regarding the latent representation models and the backbone transformer architecture refer to Appendix C.

### 4.3 Evaluation Metrics

**Perceptual Quality.** We employ Fréchet Audio Distance (Kilgour et al., 2018) (FAD) with the reference being a high-quality curated proprietary test set. We report FAD using the open source implementation *fadtk*[6] where a lower FAD score is associated with a higher perceptual quality.

**Audio Aesthetics.** We use Audiobox Aesthetics (Tjandra et al., 2025) estimators which serves as a proxy for subjective evaluation, evaluating different properties of the generated audio. Specifically, we consider:

- *Production Quality (PQ):* assesses the technical fidelity of the audio, the absence of distortions or other artifacts, well-balanced frequency range and smooth dynamics. It also reflects the skill in recording, mixing, and mastering.
- *Production Complexity (PC):* measures the number of audio elements and their interaction. Higher scores indicate layered compositions with multiple sounds, while lower scores reflect simpler, single-source recordings. It also considers how well elements blend together.
- *Content Enjoyment (CE):* captures the subjective appeal of audio, considering emotional impact, artistic skill, and creativity. Higher scores reflect engaging, expressive, and aesthetically pleasing content.

**Text Description Match.** To evaluate how well the generated audio matches the given textual description we compute the cosine similarity over a joint CLAP (Elizalde et al., 2023) text-audio representation. The similarity is computed between the track description and the generated audio, measuring audio-text alignment. We use the official pretrained CLAP model[7] in our evaluation.

**Temporally-Aligned Control Adherence.** We evaluate the adherence to different conditions: (i) chord progressions; (ii) melody; and (iii) drum beat conditioning. We extract pseudo-annotations for these conditions using pretrained classifiers and representation models. Detailed description of how these annotations were obtained can be found in Appendix A. We train a model for both AR and FM using all conditions together with condition-dropout.

We consider the following evaluation metrics:

- *Chords Intersection Over Union (IOU)*: We extract chord label and chord switch time in seconds sequence pairs for both the generated and reference audio waveforms and compute the IOU score between the two.

[5] https://github.com/Stability-AI/stable-audio-tools/tree/main
[6] https://github.com/microsoft/fadtk
[7] github.com/LAION-AI/CLAP

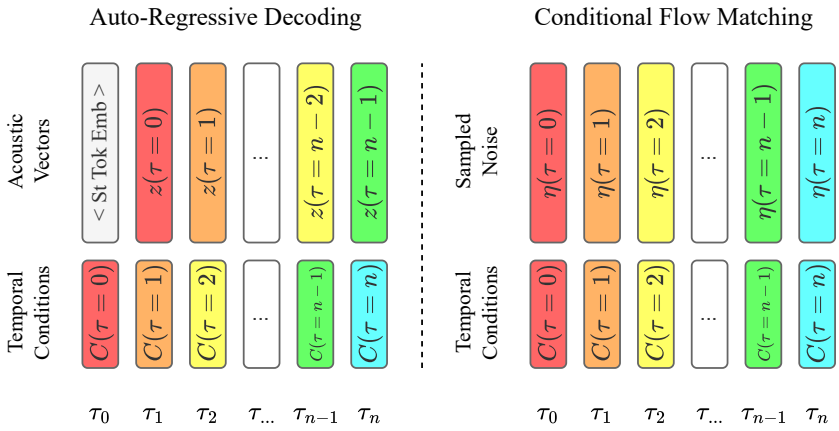


Figure 2: Temporal Conditioning Injection. $\tau_i$ denotes the temporal index in the sequence. In the auto-regressive case we apply a delayed concatenation where the conditions are stacked and concatenated over the channel axis one timestep prior to timestep they correspond to.

- *Down-beat F1 score (Beat F1)*: As Wu et al. (2024) suggested, we evaluate the down-beat F1 score using *mir eval*[8] (Raffel et al., 2014) considering a 50ms tolerance margin around classified downbeats in the reference signal.

- *Melody Chromagram Cosine Similarity (Melody Similarity)*: Similarly to Copet et al. (2023) and using the officially released implementation, we convert the resulting audio to a 12-bins chromagram representation (single octave) and compute the cosine similarity between the reference and the corresponding chromagrams.

### 4.4 Model Training

Unless stated otherwise, we train all models on 10 second segments for 500k update steps using a batch size of 256, AdamW optimizer with a learning rate of $1 \cdot 10^{-4}$, and a cosine learning rate scheduler with 4000 steps warmup followed by a cosine learning rate decay. We follow the training objectives as defined in Section 3.

### 4.5 Temporally Aligned Conditioning

Following the official release of Jasco (Tal et al., 2024), we explore conditioning using temporally aligned controls, i.e. time-dependent conditions. Controls are resampled to the latent frame-rate $f_r$ and concatenated to the input signal over the channel axis, then passed through a linear layer prior to the transformer module.

As depicted in Figure 2 the conditioning injection is done by concatenating the condition vectors (notated as $C$) to the expected transformer input, which is a shifted input in the AR case (with start token added as the first token) and a sampled standard gaussian noise $\eta$ in the FM case. The concatenated signals are then projected to the expected transformer dimension using a simple linear projection, and fed as input to the transformer module.

The training objective targets for both modeling paradigms remains the same as the text only variants, i.e. AR learns next-token prediction, FM learns the vector field from $\eta$ to $\boldsymbol{z}$. Training is done using a learned new null token for dropout, and inference is done using standard classifier free guidance with unconditional and conditional states.

[8] https://github.com/mir-evaluation/mir_eval

Table 2: Objective metrics for autoregressive (AR) and flow matching (FM) models after one million updates. Lower is better for FAD; higher is better for the other metrics.

| Hz | Modeling | FAD↓ | Clap↑ | PQ↑ | PC↑ | CE↑ |
|---|---|---|---|---|---|---|
| 25 | AR | 0.40 | 0.41 | 7.71 | 6.02 | 7.36 |
| | FM (EnC) | 0.42 | 0.39 | 7.78 | 5.42 | 7.13 |
| | FM (VAE) | 0.54 | 0.40 | 7.68 | 5.87 | 7.28 |
| 50 | AR | 0.47 | 0.40 | 7.69 | 5.78 | 7.24 |
| | FM (EnC) | 0.48 | 0.40 | 7.73 | 5.60 | 7.20 |
| | FM (VAE) | 0.87 | 0.40 | 7.41 | 5.87 | 7.07 |
| 100 | AR | 0.64 | 0.40 | 7.59 | 5.84 | 7.17 |
| | FM (EnC) | 0.68 | 0.38 | 7.47 | 5.63 | 6.92 |
| | FM (VAE) | 1.02 | 0.37 | 7.37 | 5.89 | 7.10 |

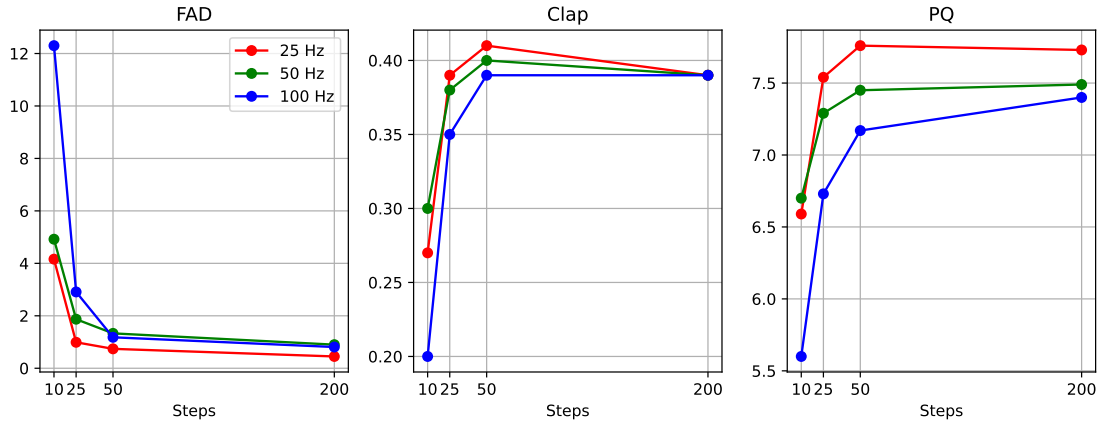


Figure 3: FM performance as a function of inference steps using Euler's method. Decreasing the number if inference steps show a steep degradation in score.

## 5 Comparative study: AR vs FM

### 5.1 Objective Comparison Under a Fixed Training Setup

We first examine AR and FM modeling paradigms under identical conditions. Each model trains for one million updates on the same dataset and backbone, and we report objective scores at latent frame rates of 25 Hz, 50 Hz, and 100 Hz in Table 2. Across the three frame rates, both modeling paradigms exhibit comparable performance, with a slight edge toward AR in terms of FAD, PC, and CE, while CLAP and PQ scores remain similar between the two. AR show less fluctuations as latent frame rate increases, overall and for FAD, PQ and CE in particular, compared to FM, especially in the VAE-based latent representation case. This observation implies that AR is more robust to the change in frame-rate. Interestingly, within the flow matching models, the EnCodec-based model consistently outperform VAE-based one on every metric.

The best performing FM configurations follows the Dopri5 (Dormand & Prince, 1980) ODE solver, a dynamic solver that iteratively estimates an error approximation and stops taking additional inference steps if the estimated error is less than a predefined tolerance factor. To further observe how performance is impacted by fixing the number of inference steps we repeat the evaluation using Euler's method on $\{10, 25, 50, 200\}$ inference steps, depicted in Figure 3. As the number of steps reduces below 50 we see a significant degradation

Table 3: Adherence to temporally aligned controls. "Single" rows report FAD / CLAP for chords, drums, and melody conditioning, respectively.

| Modeling | Conditioning | FAD↓ | Clap↑ | Chords IOU↑ | Beat F1↑ | Melody Sim.↑ |
|---|---|---|---|---|---|---|
| AR | All | **0.72** | **0.37** | **0.57** | 0.39 | **0.41** |
| FM | | 0.78 | 0.35 | 0.33 | 0.42 | 0.32 |
| AR | Single | 1.01 / 1.41 / 1.53 | 0.40 / 0.33 / 0.38 | **0.70** | 0.38 | **0.38** |
| FM | | 1.41 / 1.16 / 1.45 | 0.38 / 0.33 / 0.37 | 0.40 | 0.40 | 0.31 |

in performance. Taking more steps or using the adaptive Dopri5 solver limits this performance degradation, yet still needs a large number of steps to close the performance gap with AR. A full evaluation table is available in Table 6 on the Appendix.

**Take-away.** Both modeling paradigms (EnCodec-based latent) show comparable performance with a slight favor toward AR, which also prove to be more robust to the latent representation's sample rate. FM performance degrade as the number of inference steps decrease. In order to maintain comparable performance with AR, FM requires a large number of inference steps.

### 5.2 Temporally Aligned Control Adherence

Next, we compare AR and FM considering temporally aligned conditioning. Following JASCO (Tal et al., 2024), we train 50 Hz models conditioned on three temporally aligned conditions: chord progression, melody, and drum signals. As explained in Subsection 4.5 and visualized in Figure 2, the injection of the temporally aligned conditions is done by concatenating them over the channel axis prior to the transformer module. For each paradigm we test two scenarios: (i) all three controls provided; and (ii) one control provided while the others are set to a null token.

Despite lacking future context, Table 3 shows that the causal AR decoder tracks the controls more faithfully than FM. With all streams active, AR achieves higher Chord IoU (0.57 vs. 0.33) and melody similarity (0.41 vs. 0.32), while Beat F1 is comparable. The pattern holds in the single-control setting: AR leads on chords (0.70 vs. 0.40) and melody (0.38 vs. 0.31), and is on par for drums.

Interestingly, using temporally aligned conditioning reduces overall fidelity as apparent in FAD and CLAP scores in comparison to the text-to-music model in Section 5.1. Relative to the text-to-music model, FAD rises by 0.3–0.8 and CLAP falls by 0.02–0.05 for both paradigms. We hypothesise that the controls act as a strong bias: once top-$p$ sampling ventures onto a low-probability path that still satisfies the controls, the model continues down that trajectory, hurting realism. Appendix D expands this observation further.

**Take-away.** AR follows temporally-aligned conditioning more accurately than FM, but both paradigms lose perceptual quality under strict controls, illustrating a controllability–fidelity trade-off.

### 5.3 Inpainting

Music editing often requires replacing a flawed passage while preserving the surrounding context. We therefore compare inpainting capabilities: generating a masked span given past and future audio context. FM supports naïve zero-shot (ZS) inpainting via latent inversion, whereas AR does not (at least not for the observed vanilla setup). To enable AR we adopt the *fill-in-the-middle* strategy of Bavarian et al. (2022), where special tokens split each training example into $A\,|\,B\,|\,C$ segments; we present the model with $A\,|\,C$ and ask it to generate $B$ causally. For a fair comparison we train both AR and FM. We use a fixed 5 second masked span whose start time is chosen uniformly with at least 1 second margin on both sides. The algorithms used for FM inpainting (supervised and ZS) are available in Appendix F.

Table 4 shows that all three approaches achieve similar objective scores. However, while listening to the generated audio, we notice artifacts, audible glitches or mismatched timbre, that the objective metrics fail

Table 4: Objective scores for inpainting with a 5 seconds mask. Lower is better for FAD; higher is better for all other metrics.

| Model | FAD↓ | CLAP↑ | PQ↑ | PC↑ | CE↑ |
|---|---|---|---|---|---|
| AR | **0.23** | 0.36 | 7.75 | 5.65 | 7.31 |
| FM | 0.32 | 0.36 | 7.80 | 5.48 | 7.31 |
| FM (ZS) | 0.30 | **0.39** | **7.89** | **5.73** | **7.40** |

Table 5: Human ratings (1−10; mean ± 95 % confidence interval, ∼250 judgments per cell).

| Criterion | GT | AR | FM | FM (ZS) |
|---|---|---|---|---|
| Transition smoothness | 8.78±0.10 | 7.57±0.19 | **8.11±0.15** | 7.09±0.26 |
| Audio match | 8.81±0.12 | 7.22±0.29 | **7.93±0.21** | 6.78±0.37 |

to capture. To support our subjective observations, we conducted a human study, in which the raters were requested to rank each of the observed methods on a scale of 1 to 10, where higher is better. Each audio segment was accompanied with its corresponding waveform figure and a horizontal red line indicator in correspondence to the current temporal position. The inpainted segment was visibly marked with a **yellow** background. For a visualization example see our Sample page. The raters were required to evaluate the following criteria:

- **Transition smoothness**: How smooth the transitions between the yellow (inpainted) and the white (reference) segments is? please refer only to the transitions between the segments.
- **Audio match**: How well does the audio in the yellow segments match the audio in the white segments? A good match should maintain instrumentation, dynamics (volume), tempo, and feel like a part of the same musical piece. Ignore the smoothness of the transition between segments.

The results of the human study, presented in Table 5, confirms preliminary observation. Supervised FM receives the highest scores for both *transition smoothness* and *audio match*, indicating that it generates missing segment with better alignment to the context. AR ranks second: it produces segments with good fidelity (lowest FAD) but often leaves a discernible seam at the boundaries. Zero-shot FM delivers the best CLAP, PQ, and CE but exhibits high variance: some samples perfectly fits the context while others drift into unrelated content. This suggests that the sampling configuration could be updated per-sample and a more complex sampling strategy could be used to improve ZS capabilities.

**Take-away.** Supervised flow matching is the most robust inpainting method: it yields the smoothest and most coherent edits; zero-shot flow matching is attractive for rapid, prompt-driven edits but needs a small hyper-parameter search per-sample or a better sampling strategy to provide more stable outputs.

### 5.4 Runtime Analysis and Model Scaling

In subsection 5.1 we show that FM can approach AR quality only when it runs a large number of inference steps and still lags slightly in FAD, PC, and CE. This raises two practical questions: (i) *Is it worthwhile to cut the step count to gain inference speed?*; (ii) *Does such speed-up scales to batch processing?* AR re-uses hidden states through key–value (KV) caching, so its cost per token falls as the batch grows; FM has no comparable mechanism.

To answer these questions we record throughput (samples / sec) and per-sample latency on a single A100 GPU for batch sizes 2–256. AR is evaluated with KV caching enabled. FM is evaluated with Euler's method fixed at 200, 50, 25, and 10 steps. Objective scores for these settings appear in Appendix E (Table 6); runtime curves are plotted in Figure 4.

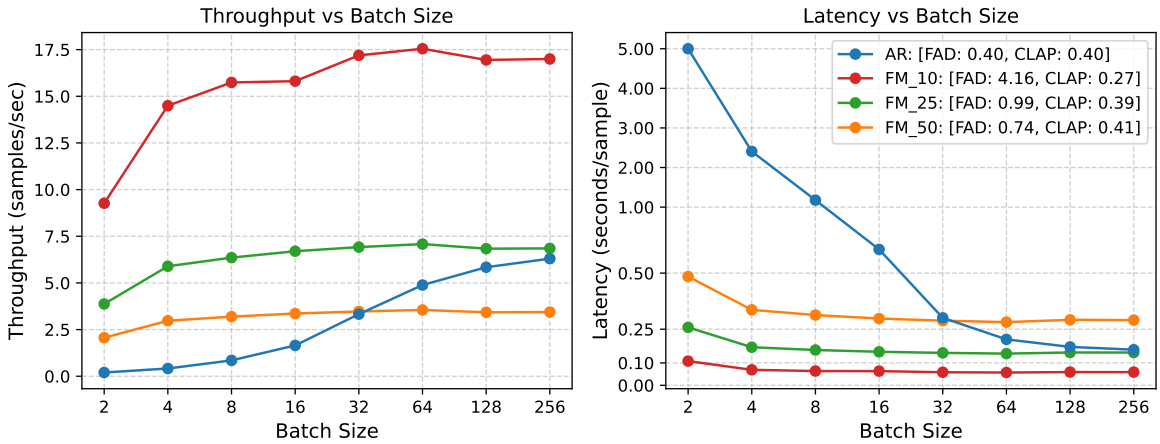


Figure 4: Inference speed versus batch size. Left: throughput; right: latency. AR gains steadily from KV caching, whereas FM plateaus after batch size 8. Euler using 10 steps is the fastest configuration but has the worst FAD (4.16).

Throughput for AR rises almost linearly, reaching 6.5 samples per second at batch 256, while latency drops below 0.15 sec per sample. FM plateaus: using 50 steps following Euler's solver, tops out near 3.5 samples per second. The gap closes only when the solver uses few steps; Euler 10 is faster than AR at every batch size, but its audio quality is significantly lower (FAD 4.16 versus 0.40).

**Take-away.** AR scales better with batch size thanks to KV caching; FM may becomes faster while reducing the number of inference steps, however this comes at the cost of degraded generation quality. Selecting a modeling paradigm therefore hinges on how much quality one is willing to trade for latency.

### 5.5 Sensitivity to Training Configuration

In Section 5.1 we consider models that were trained using a one million update steps and a batch size of 256. Real-world projects often operate under lower-resource constraints. In this experiment we maintain the number of update steps fixed and vary the *tokens seen per update* by changing batch size and segment duration. The goal is to observe how each modeling paradigm's performance change as a function of batch size and segment duration. For each configuration defined by batch size $\in \{8, 16, 32, 64, 128, 256\}$ and segment duration $\in \{10, 30\}$[sec] we train two generative models using AR decoding and FM.

Figure 5 shows that FAD decreases for both paradigms as batch size increases, where the reduction is steeper for the AR model. At the largest setting (256 batch-size of 10-second clips) the AR FAD almost matches the value reached after one million updates, as seen in Section 5.1, while FM levels off earlier. When considering CLAP similarity, FM benefits steadily from larger batches, whereas the AR shows some fluctuations and relatively flattens above batch size 16. Aesthetic metrics present the opposite pattern: PQ and CE are roughly constant for FM setup, while for the AR setup they consistently improve. Notice, the AR results are still below the ones obtained after one million update steps, implying there it still benefit from longer training.

**Take-away.** When the number of update steps is capped, FM reaches almost the same FAD, PQ, and CE as in the one-million-step topline using much smaller batches, though its CLAP score keeps improving with scale. The AR model needs a larger token budget per step to match its topline performance and benefits more from large scale training.

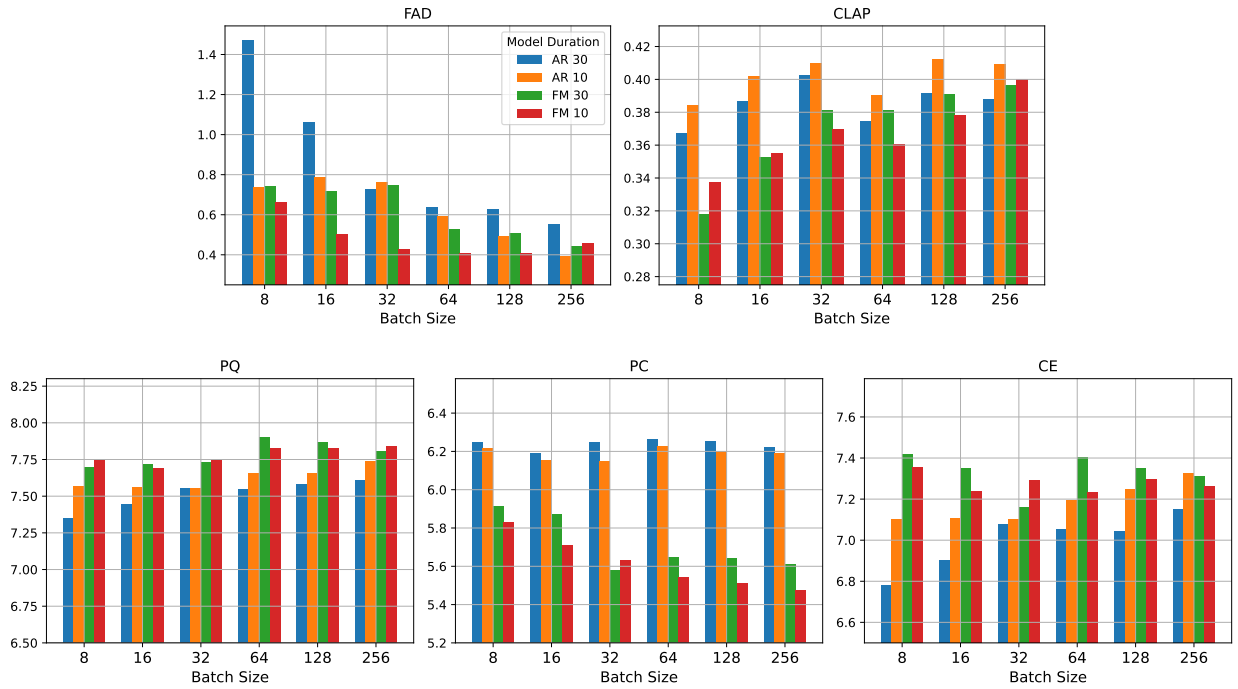


Figure 5: Objective scores after 500k updates as a function of batch size and segment duration. Both paradigms improve with more tokens per update step, but AR is more sensitive to the change.

## 6 Conclusions and Future Work

We compared AR and FM under a unified framework that fixed data, representation model, backbone architecture and evaluation code. Five tasks were considered: text-to-music generation at different frame rates, conditioning temporally-aligned controls, inpainting, throughput and latency inference scalability, and sensitivity to training configuration considering a fixed number of training steps.

AR achieved the lowest FAD and slightly higher PQ and CE, showing higher robustness to latent frame rate changes; whereas FM degraded at the higher rate, especially considering a VAE-based latent. AR also followed temporally-aligned controls more closely. General fidelity under this conditional setup decreased for both modeling paradigms, confirming a tension between controllability and generation quality. For inpainting, the supervised FM model received the highest human ratings for transition smoothness and audio match, while AR retained the best FAD but often suffered from audible seams; zero-shot FM showed inconsistent behavior - from seamless natural transitions to generating unrelated segments. On the compute side AR throughput rose almost linearly with batch thanks to key–value caching, while FM flattened quickly and surpassed AR only when considered few steps with an apparent reduction in quality. With a fixed update budget FM reached near-topline quality with smaller batch sizes, whereas AR required larger batches and continues to improve from 500K to 1M steps.

This study centered on a single 400M-parameter transformer model and maintained a controlled experimental setup across all evaluations. While our findings reveal clear distinctions between AR and FM under these constraints, we acknowledge that alternative sampling strategies, efficient training methods, architectural innovations, and model scaling could yield different results. Future work would explore these axes to more comprehensively assess the strengths and limitations of each paradigm. We encourage the community to further investigate AR and FM models under fair, unified settings to advance our collective understanding of controllable and efficient music generation.

## References


Andrea Agostinelli, Timo I Denk, Zalán Borsos, Jesse Engel, Mauro Verzetti, Antoine Caillon, Qingqing Huang, Aren Jansen, Adam Roberts, Marco Tagliasacchi, et al. Musiclm: Generating music from text. *arXiv preprint arXiv:2301.11325*, 2023.

Ye Bai, Haonan Chen, Jitong Chen, Zhuo Chen, Yi Deng, Xiaohong Dong, Lamtharn Hantrakul, Weituo Hao, Qingqing Huang, Zhongyi Huang, et al. Seed-music: A unified framework for high quality and controlled music generation. *arXiv preprint arXiv:2409.09214*, 2024.

Mohammad Bavarian, Heewoo Jun, Nikolas Tezak, John Schulman, Christine McLeavey, Jerry Tworek, and Mark Chen. Efficient training of language models to fill in the middle. *arXiv preprint arXiv:2207.14255*, 2022.

R.M. Bittner, B. McFee, J. Salamon, P. Li, and J.P. Bello. Deep salience representations for $f_0$ estimation in polyphonic music. In *18th Int. Soc. for Music Info. Retrieval Conf.*, Suzhou, China, Oct. 2017.

Zalán Borsos, Raphaël Marinier, Damien Vincent, Eugene Kharitonov, Olivier Pietquin, Matt Sharifi, Dominik Roblek, Olivier Teboul, David Grangier, Marco Tagliasacchi, et al. Audiolm: a language modeling approach to audio generation. *IEEE/ACM transactions on audio, speech, and language processing*, 31: 2523–2533, 2023.

Ke Chen, Yusong Wu, Haohe Liu, Marianna Nezhurina, Taylor Berg-Kirkpatrick, and Shlomo Dubnov. Musicldm: Enhancing novelty in text-to-music generation using beat-synchronous mixup strategies. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1206–1210. IEEE, 2024.

Jade Copet, Felix Kreuk, Itai Gat, Tal Remez, David Kant, Gabriel Synnaeve, Yossi Adi, and Alexandre Défossez. Simple and controllable music generation. *Advances in Neural Information Processing Systems*, 36:47704–47720, 2023.

Alexandre Défossez, Jade Copet, Gabriel Synnaeve, and Yossi Adi. High fidelity neural audio compression. *arXiv preprint arXiv:2210.13438*, 2022.

Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*, 2020.

Patrick Donnelly and John Sheppard. Evolving four-part harmony using genetic algorithms. In *Applications of Evolutionary Computation: EvoApplications 2011: EvoCOMNET, EvoFIN, EvoHOT, EvoMUSART, EvoSTIM, and EvoTRANSLOG, Torino, Italy, April 27-29, 2011, Proceedings, Part II*, pp. 273–282. Springer, 2011.

John R Dormand and Peter J Prince. A family of embedded runge-kutta formulae. *Journal of computational and applied mathematics*, 6(1):19–26, 1980.

Benjamin Elizalde, Soham Deshmukh, Mahmoud Al Ismail, and Huaming Wang. Clap learning audio concepts from natural language supervision. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5. IEEE, 2023.

Zach Evans, Julian D Parker, CJ Carr, Zack Zukowski, Josiah Taylor, and Jordi Pons. Stable audio open. *arXiv preprint arXiv:2407.14358*, 2024.

Allen Huang and Raymond Wu. Deep learning for music. *arXiv preprint arXiv:1606.04930*, 2016.

Delong Huang and Fei Guo. Multiplicity of periodic bouncing solutions for generalized impact hamiltonian systems. *Boundary Value Problems*, 2019(1):57, 2019.

Qingqing Huang, Daniel S Park, Tao Wang, Timo I Denk, Andy Ly, Nanxin Chen, Zhengdong Zhang, Zhishuai Zhang, Jiahui Yu, Christian Frank, et al. Noise2music: Text-conditioned music generation with diffusion models. *arXiv preprint arXiv:2302.03917*, 2023.

Kevin Kilgour, Mauricio Zuluaga, Dominik Roblek, and Matthew Sharifi. Fr\'echet audio distance: A metric for evaluating music enhancement algorithms. *arXiv preprint arXiv:1812.08466*, 2018.

Max WY Lam, Qiao Tian, Tang Li, Zongyu Yin, Siyuan Feng, Ming Tu, Yuliang Ji, Rui Xia, Mingbo Ma, Xuchen Song, et al. Efficient neural music generation. *Advances in Neural Information Processing Systems*, 36:17450–17463, 2023.

Gael Le Lan, Bowen Shi, Zhaoheng Ni, Sidd Srinivasan, Anurag Kumar, Brian Ellis, David Kant, Varun Nagaraja, Ernie Chang, Wei-Ning Hsu, et al. High fidelity text-guided music editing via single-stage flow matching. *arXiv preprint arXiv:2407.03648*, 2024.

Peike Patrick Li, Boyu Chen, Yao Yao, Yikai Wang, Allen Wang, and Alex Wang. Jen-1: Text-guided universal music generation with omnidirectional diffusion models. In *2024 IEEE Conference on Artificial Intelligence (CAI)*, pp. 762–769. IEEE, 2024.

Sifei Li, Mining Tan, Feier Shen, Minyan Luo, Zijiao Yin, Fan Tang, Weiming Dong, and Changsheng Xu. A survey on cross-modal interaction between music and multimodal data. *arXiv preprint arXiv:2504.12796*, 2025.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Xingchao Liu, Xiwen Zhang, Jianzhu Ma, Jian Peng, et al. Instaflow: One step is enough for high-quality diffusion-based text-to-image generation. In *The Twelfth International Conference on Learning Representations*, 2023.

Huanru Henry Mao, Taylor Shin, and Garrison Cottrell. Deepj: Style-specific music generation. In *2018 IEEE 12th international conference on semantic computing (ICSC)*, pp. 377–382. IEEE, 2018.

George Papadopoulos and Geraint Wiggins. Ai methods for algorithmic composition: A survey, a critical view and future prospects. In *AISB symposium on musical creativity*, volume 124, pp. 110–117. Edinburgh, UK, 1999.

Mohammed Johar Pathariya, Pratyush Basavraj Jalkote, Aniket Maharudra Patil, Abhishek Ashok Sutar, and Rajashree L Ghule. Tunes by technology: A comprehensive survey of music generation models. In *2024 International Conference on Cognitive Robotics and Intelligent Systems (ICC-ROBINS)*, pp. 506–512. IEEE, 2024.

Richard C Pinkerton. Information theory and melody. *Scientific American*, 194(2):77–87, 1956.

Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023.

Colin Raffel, Brian McFee, Eric J Humphrey, Justin Salamon, Oriol Nieto, Dawen Liang, Daniel PW Ellis, and C Colin Raffel. Mir_eval: A transparent implementation of common mir metrics. In *ISMIR*, volume 10, pp. 2014, 2014.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Ian Simon and Sageev Oore. Performance rnn: Generating music with expressive timing and dynamics. `https://magenta.tensorflow.org/performance-rnn`, 2017.

Or Tal, Alon Ziv, Itai Gat, Felix Kreuk, and Yossi Adi. Joint audio and symbolic conditioning for temporally controlled text-to-music generation. *arXiv preprint arXiv:2406.10970*, 2024.

Andros Tjandra, Yi-Chiao Wu, Baishan Guo, John Hoffman, Brian Ellis, Apoorv Vyas, Bowen Shi, Sanyuan Chen, Matt Le, Nick Zacharov, et al. Meta audiobox aesthetics: Unified automatic quality assessment for speech, music, and sound. *arXiv preprint arXiv:2502.05139*, 2025.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Apoorv Vyas, Bowen Shi, Matthew Le, Andros Tjandra, Yi-Chiao Wu, Baishan Guo, Jiemin Zhang, Xinyue Zhang, Robert Adkins, William Ngan, et al. Audiobox: Unified audio generation with natural language prompts. *arXiv preprint arXiv:2312.15821*, 2023.

Shih-Lun Wu, Chris Donahue, Shinji Watanabe, and Nicholas J Bryan. Music controlnet: Multiple time-varying controls for music generation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 32:2692–2703, 2024.

Neil Zeghidour, Alejandro Luebs, Ahmed Omran, Jan Skoglund, and Marco Tagliasacchi. Soundstream: An end-to-end neural audio codec. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 30: 495–507, 2021.

Yueyue Zhu, Jared Baca, Banafsheh Rekabdar, and Reza Rawassizadeh. A survey of ai music generation tools and models. *arXiv preprint arXiv:2308.12982*, 2023.

# A Temporal Controls Preprocessing

In this work we consider drum beat, melody, and chord progression conditioning. We perform an offline preprocessing stage for each observed latent representation frame rate, and save the preprocessed conditioning signals to memory.

**Drum Beat** To obtain the drum beat supervision we follow Tal et al. (2024), utilizing a pretrained EnCodec (Défossez et al., 2022) model. For each data sample, we first encode it to it's corresponding pre-quantization continuous representation using a pretrained EnCodec model that operates in the expected latent representation frame rate. We then perform temporal blurring (Tal et al., 2024), averaging every 5 sequential latent vectors and broadcast them back to their original frame rate. Finally, we pass the blurred latent vector through the first vector quantization layer of the pretrained EnCodec model, and save the resulted integer sequence to memory.

**Melody** To obtain the melody condition we use the pretrained deep salience multi-F0 detector [9] (Bittner et al., 2017). The pretrained multi-F0 detector outputs a confidence score over a predetermined range of 53 notes (G2 to B7) spanning over $\sim$ 86Hz confidence vector sequence. Given the expected latent representation frame rate, we perform a simple linear interpolation to stretch / shrink the confidence vector stream to match it. We then pass a threshold of 0.5 confidence score, zeroing out all values below threshold. Finally, we create an integer sequence for each data sample, replacing each entry with its corresponding argmax or 54 in case the column contains only zeros.

**Chord Progression** To obtain chord progressions, we use the Chordino [10] chord extraction model and create a (<chord label>, <switch time in sec>) pairs sequence for each data sample in our dataset. Chordino has a vocabulary size of 193 different chords, hence we create a chord to index mapping and save it to memory to be further used for tokenization. Given an expected latent frame rate, we can then convert the extracted chord sequences to integer sequences using the precomputed chord to index mapping, quantizing the <switch time in sec> timestamps to match the expected frame rate, repeating the same chord index until the next switch.

# B Sampling Hyperparameter Search

For both modeling paradigms we experiment with classifier free guidance coefficients $\in \{1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0\}$.

For FM we do inference with Dopri (dynammic number of steps) or Euler (considering number of steps: $\in \{10, 25, 50, 150.250\}$).

For AR we explore temperature $\in \{1.2, 1.4, 1.6, 2.0, 2.4, 2.8\}$, top p $\in \{0.6, 0.8\}$ or top k $\in \{250, 500\}$.

# C Latent representation and transformer model specifications.

## C.1 Latent Representation

We follow the approach taken in Copet et al. (2023); Vyas et al. (2023); Tal et al. (2024) and use EnCodec's (Défossez et al., 2022) quantized discrete representation for AR modeling and its continuous, pre-quantizer, latent for FM modeling. Using un-normalized representations shows a notable performance gap. In addition, we train a VAE-GAN autoencoder following StableAudio's (Evans et al., 2024) open source recipe[11], training with a latent KL divergence constraint w.r.t $\mathcal{N}(0, I)$, without any quantization performed during training. We train $\{25, 50, 100\}$[Hz] latent frequency variants for each model, training for $400k$ steps using AdamW optimizer with a learning rate of $3 \cdot 10^{-4}$ considering $\sim$ 1[Sec] segments and a batch size of 64 samples. The specific configurations used for each model variant can be found in subsection C.3.

[9] https://github.com/rabitt/ismir2017-deepsalience
[10] https://github.com/ohollo/chord-extractor
[11] https://github.com/Stability-AI/stable-audio-tools/tree/main

### C.2 Normalizing EnCodec Latent Representation For FM modeling.

We sample $N = 2048$ 10 second random segments from our train set and encode them to a latent representation matrix $M$ of shape $[N, T, D]$ where $D$ is the latent dimension and $T$ is the corresponding temporal dimension $T = 10 \cdot f_r$. We compute a single scalar for the empirical mean and empirical mean std as follows:

```
mean = M.mean()
mean_std = M.std(dim=1).mean()
---
def normalize(z: Tensor):
    return (z - mean) / mean_std

def unnormalize(z: Tensor):
    return z * mean_std + mean
```

### C.3 Latent Representation Models Configurations

Both of our observed latent representation models are symmetric auto-encoder models. We use the open-source implementation in Audiocraft to train EnCodec, and the implementation in Stable-Audio-Tools to train the VAE. The table below specifies the critical hyper-parameters required to train each of the representation model configurations. In the Discrete EnCodec case we use 4 residual codebooks, each containing 2048 bins.

| Model | Frame Rate | Strides | Channels | Activation | Latent Dimension |
|---|---|---|---|---|---|
| EnCodec | 25 | [8, 8, 5, 4] | [64, 128, 256, 512] | GELU | 128 |
| | 50 | [8, 5, 4, 4] | | | |
| | 100 | [8, 5, 4, 2] | | | |
| VAE $\alpha_{\text{KL}} = 10^{-3}$ | 25 | [2, 4, 4, 5, 8] | [128, 256, 512, 1024, 2048] | Snake | 64 |
| | 50 | [2, 4, 4, 4, 5] | | | |
| | 100 | [2, 2, 4, 4, 5] | | | |

#### C.3.1 Backbone Transformer

For the backbone transformer decoder model we follow the implementation of Copet et al. (2023) using a 400M parameters configuration containing 24 transformer decoder blocks with a hidden dimension of 1024, 16 multi-head attention layers and a feed-forward dimension of 4096. We use T5 (Raffel et al., 2020) to obtain text embeddings and pass them via cross-attention layers as text conditions. For the FM case we perform slight modifications similarly to Tal et al. (2024) and include U-Net-like skip connections. With $2N$ being the number of transformer decoder blocks, we add skip-connections, connecting the input of the $i$'th block with the $2N - i$'th block output, for $i \geq N + 1$. Each skip connection follows a simple concatenation and linear projection: Linear(Concat($x$, skip)).

## D Entropy Analysis Under Strict Temporal Conditioning

To shed light on the quality drop observed in Table 3, we measure how strict conditioning affects the uncertainty of the AR decoder. For each test prompt we record the token probability over the first EnCodec codebook stream at every sampling step and compute its entropy. The curve is averaged over 100 random prompts; the text-to-music baseline is obtained by using the models trained in Section 5.1.

Figure 6 shows that conditioning sharpens the distribution early in the sequence, where the provided control tokens directly determine the next few audio tokens, but then the entropy plateaus around timestep 50. We hypothesize that the conditioning imposes a strong bias and therefore leads to a higher chance of sampling out-of-distribution token sequences (some steps doesn't match the conditions), and causes a higher entropy

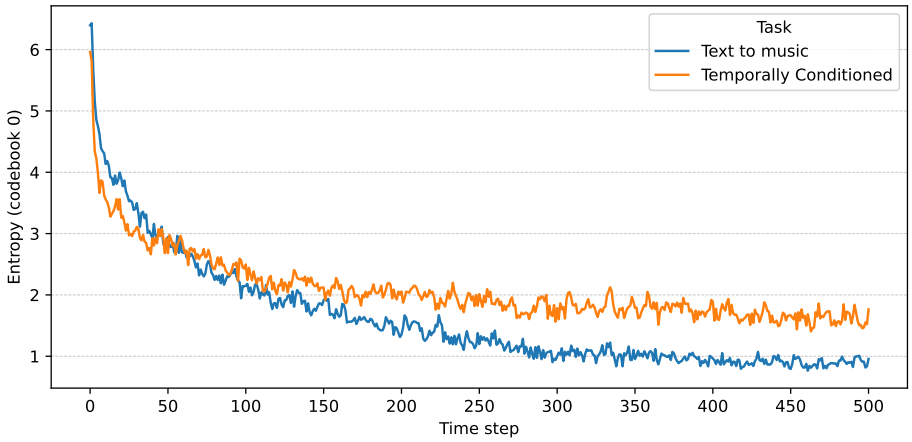


Figure 6: Average entropy of the AR decoder as a function of sampling step. Conditioning lowers entropy in the first $\sim$50 steps, indicating confident early predictions, but raises it later, suggesting increased sensitivity to low-probability paths.

as the sequence becomes longer due to that. This manifests as the FAD and CLAP degradation reported in Section-5.2. This also implies that using multi-source classifier-free guidance (Tal et al., 2024) may mitigate this effect, and we leave that for future work.

## E Constricting Sampling Steps

Table 6 depicts the full tradeoff w.r.t objective quality metrics when fixing the number of sampling steps made with Euler ODE solver during FM inference.

Table 6: FM sampling using fixed number of steps with Euler ODE solver.

| Hz | Steps | FAD↓ | Clap↑ | PQ↑ | PC↑ | CE↑ |
|---|---|---|---|---|---|---|
| 25 | 200 | 0.45 | 0.39 | 7.73 | 5.50 | 7.16 |
| | 50 | 0.74 | 0.41 | 7.76 | 5.63 | 7.24 |
| | 25 | 0.99 | 0.39 | 7.54 | 5.48 | 7.01 |
| | 10 | 4.16 | 0.27 | 6.59 | 5.07 | 5.94 |
| 50 | 200 | 0.90 | 0.39 | 7.49 | 5.70 | 6.97 |
| | 50 | 1.33 | 0.40 | 7.45 | 5.68 | 6.95 |
| | 25 | 1.87 | 0.38 | 7.29 | 5.64 | 6.78 |
| | 10 | 4.92 | 0.30 | 6.70 | 5.46 | 6.10 |
| 100 | 200 | 0.81 | 0.39 | 7.40 | 5.64 | 6.86 |
| | 50 | 1.18 | 0.39 | 7.17 | 5.74 | 6.72 |
| | 25 | 2.91 | 0.35 | 6.73 | 5.7 | 6.31 |
| | 10 | 12.3 | 0.20 | 5.60 | 5.02 | 4.67 |

**Algorithm 1** Zero-Shot Inpainting Evaluation via Flow Matching Inversion

**Require:** Source latent $\mathbf{z}_0$ which is a sequence of $T$ latent vectors, condition tensors $\mathcal{C}$, guidance terms $\mathcal{G}$, number of Euler steps $N$ and a fixed mask size $M = T/2$
**Ensure:** Inpainted latent $\mathbf{z}$

**Initialize:** $\Delta t \leftarrow 1/N$, $\mathbf{z} \leftarrow \mathbf{z}_0$, $t \leftarrow 1$
Create empty list `noises` $\leftarrow [\,]$ ▷ Perform inversion
**for** $i = 1$ to $N$ **do**
    Compute $\mathbf{v}_\theta \leftarrow \texttt{model}(\mathbf{z}, t, \mathcal{C}, \mathcal{G})$
    Update $\mathbf{z} \leftarrow \mathbf{z} - \Delta t \cdot \mathbf{v}_\theta$
    Append current $\mathbf{z}$ to `noises`
    Update $t \leftarrow t - \Delta t$
**end for**
Reverse the list `noises`
**Sample** random start index $s \sim \text{Uniform}(0.1 \cdot T, 0.9 \cdot T)$
Set $e \leftarrow s + M$
Initialize $\mathbf{z} \sim \mathcal{N}(0, I)$, $t \leftarrow 0$ ▷ Perform inpainting
**for** each noise $\mathbf{n}$ in `noises` **do**
    Replace $\mathbf{z}[:, : s] \leftarrow \mathbf{n}[:, : s]$, $\mathbf{z}[:, e :] \leftarrow \mathbf{n}[:, e :]$
    Compute $\mathbf{v}_\theta \leftarrow \texttt{model}(\mathbf{z}, t, \mathcal{C}, \mathcal{G})$
    Update $\mathbf{z} \leftarrow \mathbf{z} + \Delta t \cdot \mathbf{v}_\theta$
    Update $t \leftarrow t + \Delta t$
**end for**
Replace $\mathbf{z}[:, : s] \leftarrow \mathbf{z}_0[:, : s]$, $\mathbf{z}[:, e :] \leftarrow \mathbf{z}_0[:, e :]$
**return** $\mathbf{z}$

**Algorithm 2** Supervised Inpainting Flow Matching

**Require:** Source latent $\mathbf{z}_0$, condition tensors $\mathcal{C}$, guidance terms $\mathcal{G}$, number of Euler steps $N$ and a fixed mask size $M = T/2$
**Ensure:** Inpainted latent $\mathbf{z}$

**Sample** random start index $s \sim \text{Uniform}(0.1 \cdot T, 0.9 \cdot T)$
Set $e \leftarrow s + M$
**for** step in $\{1, ..., N\}$ **do**
    Replace $\mathbf{z}[:, : s] \leftarrow \mathbf{z_0}[:, : s]$, $\mathbf{z}[:, e :] \leftarrow \mathbf{z_0}[:, e :]$
    Compute $\mathbf{v}_\theta \leftarrow \texttt{model}(\mathbf{z}, t, \mathcal{C}, \mathcal{G})$
    Update $\mathbf{z} \leftarrow \mathbf{z} + \Delta t \cdot \mathbf{v}_\theta$
    Update $t \leftarrow t + \Delta t$
**end for**
Replace $\mathbf{z}[:, : s] \leftarrow \mathbf{z}_0[:, : s]$, $\mathbf{z}[:, e :] \leftarrow \mathbf{z}_0[:, e :]$
**return** $\mathbf{z}$

## F Inpainting algorithms

In the fine-tuning case of AR decoding, we need to introduce 3 new tokens (<a>, <b>, <c>) in order to partition the source latent representation to 3 segments: A,B,C. B is the segment to be inpainted. We place the special token prior to each segment, resulting in a $T + 3$ length segment, and reorganize the sequence as [<a>, A, <c>, C, <b>, B]. During inference we give [<a>, A, <c>, C, <b>] as a prompt to the model, which continues to generate until <eos> or max generation length is met. We then re-organize the segments to A, B, C and reconstruct the waveform.

For FM, we could perform inpainting either by using a pretrained model and perform zero-shot inpainting, or by training a model specifically for the task. Algorithm 1 describes the naive logic implemented to allow for zero-shot inpainting using a FM model and a fixed sampling schedule. In the supervised case of FM, we

do not perform inversion, and simply plug in the latent representation itself in the unmasked segments as depicted in algorithm 2.