

# STREAMING GENERATION FOR MUSIC ACCOMPANIMENT

Yusong Wu<sup>1</sup>, Mason Wang<sup>2</sup>, Heidi Lei<sup>2</sup>, Stephen Brade<sup>2</sup>, Lancelot Blanchard<sup>2</sup>, Shih-Lun Wu<sup>2</sup>  
Aaron Courville<sup>1,3</sup>, Anna Huang<sup>2</sup>

<sup>1</sup>Mila, Quebec Artificial Intelligence Institute, Université de Montréal  
<sup>2</sup>MIT - Massachusetts Institute of Technology <sup>3</sup>Canada CIFAR AI Chair

## ABSTRACT

Music generation models can produce high-fidelity coherent accompaniment given complete audio input, but are limited to editing and loop-based workflows. We study real-time audio-to-audio accompaniment: as a model hears an input audio stream (e.g., a singer singing), it has to also simultaneously generate in real-time a coherent accompanying stream (e.g., a guitar accompaniment). In this work, we propose a model design considering inevitable system delays in practical deployment with two design variables: future visibility  $t_f$ , the offset between the output playback time and the latest input time used for conditioning, and output chunk duration  $k$ , the number of frames emitted per call. We train Transformer decoders across a grid of  $(t_f, k)$  and show two consistent trade-offs: increasing effective  $t_f$  improves coherence by reducing the recency gap, but requires faster inference to stay within the latency budget; increasing  $k$  improves throughput but results in degraded accompaniment due to a reduced update rate. Finally, we observe that naive maximum-likelihood streaming training is insufficient for coherent accompaniment where future context is not available, motivating advanced anticipatory and agentic objectives for live jamming.

**Index Terms**— Real-time Music Accompaniment, Music Generation, Audio Generation, LLMs

## 1. INTRODUCTION

Live music interaction, such as jamming or accompaniment, involves reciprocal coordination, anticipation, and collaborative creativity among players [1], resulting in an experience of creative flow [2] and social connection [3]. In live acoustic accompaniment, a model must listen to an incoming audio stream and emit an aligned coherent accompanying stream in real-time.

Recent multi-track generation systems leverage modern deep generative models to achieve high-fidelity accompaniment [4–6]. However, these systems are operated offline: they require the full input context before decoding. As they typically rely on large-scale generative models, generation of audio incurs nontrivial end-to-end latency and the systems are constrained to editing and loop-based workflows rather than live interaction. In offline use, the same latency merely lengthens the wait. In live accompaniment, excessive end-to-end latency or failure to maintain real-time alignment yields misaligned, inharmonic output, even when the generated accompaniment is coherent. Several recent works target real-time accompaniment but focus on small generative models with simple symbolic outputs such as note counterpoint or chord accompaniments [7–9].

In this paper, we study a framework for designing live acoustic-to-acoustic accompaniment with large-scale generative models that operate in real time. We target two deployment constraints: alignment and throughput. The accompaniment rendered and played at

wall-clock time  $t$  should musically align with the input audio at the same time, and generation must keep pace with playback. We formalize accompaniment as streaming chunked prediction with two design variables: the future visibility  $t_f$  and the output chunk duration  $k$ . At step  $t$ , the model predicts the future chunk  $y(t:t+k)$  given the input history  $x(1:t+t_f)$  and the previous output  $y(1:t)$ . Because end-to-end latency from audio I/O, model inference, and audio rendering is nontrivial, a negative  $t_f$  allows the model to produce audio slightly ahead to compensate, while predicting chunks of length  $k$  enables buffering and parallel generation. These choices introduce trade-offs: smaller  $t_f < 0$  increases tolerance to latency variability but widens the recency gap and reduces reactivity; larger  $k$  improves stability and throughput but lowers the update rate and delays incorporation of the latest input. This formulation separates model behavior from system constraints and exposes a latency-responsiveness design space that trade off quality.

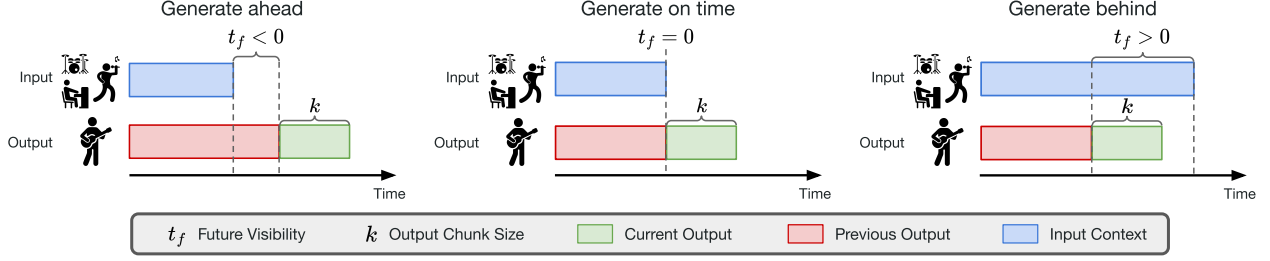
We investigate these trade-offs using Transformer-based decoders [10] trained on multi-track audio from the Slakh2100 dataset [11]. We tokenize audio with a residual vector-quantized (RVQ) representation [12] and condition the model on a random mixture of other tracks to generate one track. We sweep  $t_f$  to emulate systems with different system delay budgets, and sweep  $k$  to emulate different throughput and buffering regimes. We evaluate with three metrics for accompaniment quality under different system designs to show the frontier of the quality-latency-responsiveness trade-offs. We evaluate a learned coherence score and rhythm alignment between input mixture and output track, as well as audio quality on the generated track. Additionally, we conduct a listening study on selected systems, validating that our metrics align well with human perception.

Our contributions are as follows. First, we introduce a formal real-time accompaniment setup for streaming audio that identifies future visibility  $t_f$  and chunk size  $k$  as design axes. Second, we provide a systematic study on the Slakh2100 dataset that quantifies quality-latency-responsiveness trade-offs induced by different system designs across a grid of  $(t_f, k)$ . Third, we show that supervised streaming training alone is insufficient for coherent accompaniment, which motivates future work to develop training objectives that encode anticipation and coordination for live jamming. Finally, we release our codebase and pretrained checkpoints to support reproducibility and future research<sup>1</sup>.

## 2. RELATED WORK

**Real-time Music Accompaniment Systems** Building a computer partner that can perform with humans in real time has been a long-standing goal in computer music research [13]. Early systems are

<sup>1</sup>Audio samples: <https://lukewys.github.io/stream-music-gen>  
Code: <https://github.com/lukewys/stream-music-gen>



**Fig. 1.** We formalize real-time accompaniment as a streaming, chunked prediction. At each inference step, the model predicts a length- $k$  future chunk given its previous outputs and the input history, with the future visibility  $t_f$ . Models with  $t_f < 0$  predicts ahead to compensate for system latency, where the extreme case  $t_f = 0$  yields synchronous generation. Models with  $t_f > 0$  generate with input lookahead, which makes aligned playback to the performer infeasible.

predominantly non-neural. Score-following methods align a live performance to a notated score and trigger a pre-authored accompaniment in synchrony [13–15]. Other approaches generate accompaniment with hand-crafted heuristics [16] or by recombining phrases from a corpus that is either precompiled or built from recent input [17–19]. Recent work has shifted toward deep generative models for accompaniment. Most systems adopt small model backbones and target simplified symbolic tasks such as note counterpoint or chord sequence prediction [7–9, 20, 21]. Notably, the recently released Magenta RealTime [22] generates an uninterrupted stream of acoustic music and responds continuously to user input specified by weighted text prompts. In contrast, we study design choices for acoustic-to-acoustic accompaniment models that listen to a live audio stream and aim to emit music in real time.

**Real-time Control** Real-time control has long addressed fixed and time-varying delays by predicting and scheduling outputs in advance. Classic dead-time compensation, typified by the Smith predictor [23], uses an internal model to forecast the plant response over the delay and to adjust control so the commanded behavior remains time aligned. Model predictive control formalizes this lookahead through a finite prediction horizon and can incorporate known or bounded delays directly in the model or via horizon settings for robust tracking under latency [24]. Recent research on real-time robot execution adopts action chunking that predicts a short sequence of future actions per inference, which provides an execution buffer during computation [25].

### 3. STREAMING ACCOMPANIMENT MODELS

#### 3.1. Problem setup

We study streaming accompaniment with an input audio stream  $x(1, \dots, T)$  and an equal-length output accompaniment stream  $y(1, \dots, T)$ . At the current time  $t \in \{1, \dots, T\}$ , the model must make musical decisions given partially observable context, i.e., it can only use the input history and its own previous output. When deployed in a system, the rendered output at each moment should musically accompany the present input (e.g., in harmony and rhythm). During the interaction window, the model continuously generates audio while adapting to new inputs, and ideally can anticipate near-future input so that it generates slightly in advance. We refer to such models as *streaming accompaniment models*.

We formulate one streaming step as modeling a chunk prediction with *future visibility*  $t_f \in (-T, T)$ , the temporal offset between the start time of the prediction  $t$  and the latest input; *output chunk duration*  $k > 0$ , the length of audio emitted per generation call:

$$p_{\theta}(y(t:t+k) | y(1:t), x(1:t+t_f)). \quad (1)$$

As shown in Figure 1, the model with parameter  $\theta$  streams the accompaniment by iteratively calling the model: to predict the output starting from  $t$ , it conditions on previous output  $y(1:t)$  and input history  $x(1:t+t_f)$ , produces a length- $k$  chunk  $y(t:t+k)$ , then renders it. In this paper we focus on model design and training under this interface and defer interactive system engineering to future work.

#### 3.2. Future Visibility

The future visibility  $t_f$  intends to model the real delay from receiving the input to emitting audible output. By allowing  $t_f$  to be negative, zero, or positive, we cover three regimes:  $t_f < 0$  predicts ahead to compensate for system latency,  $t_f = 0$  is strictly synchronous, and  $t_f > 0$  grants lookahead to future input available in buffered or offline use. Smaller  $t_f < 0$  increases robustness to variable delays but contains less recent input information. When generating ahead, the quantity  $t_f < 0$  governs robustness when delays vary during operation, and it controls the *recency gap* between the latest available input  $x(t+t_f)$  and the first sample of the predicted chunk  $y(t)$ .

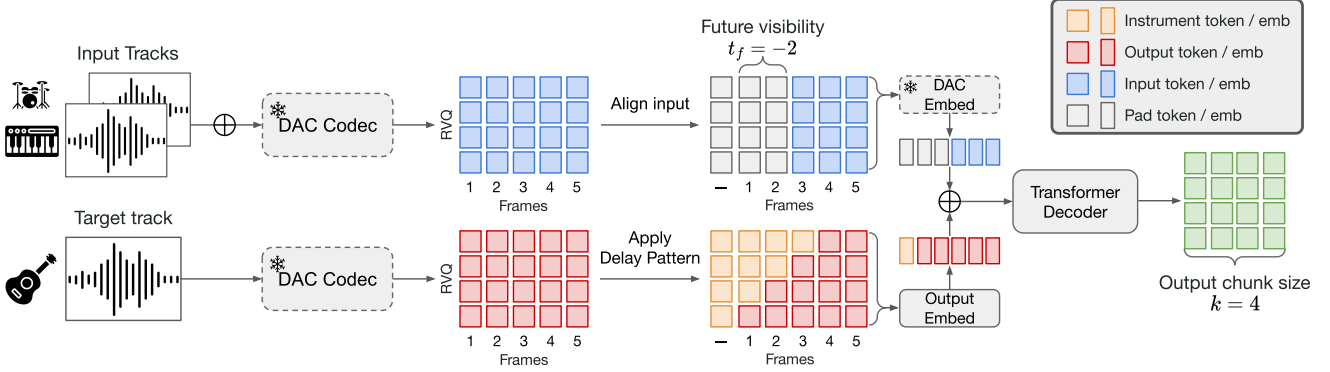
#### 3.3. Output Chunk Duration

The output chunk duration  $k$  specifies the length of each generated chunk. Setting the chunk duration  $k$  creates headroom between playback and inference. While the current chunk is rendered, the next chunk can be computed, so inference calls can be scheduled within a timing window rather than at a fixed cadence. This asynchronous scheduling improves tolerance to compute or network jitter, enables remote serving, and accommodates non-autoregressive generative models.  $k$  trades responsiveness for stability and efficiency. Smaller  $k$  increases responsiveness by allowing more frequent context updates while larger  $k$  increases stability and throughput.

#### 3.4. Practical System Design

The pair  $(t_f, k)$  defines a model space that balances latency and reactivity. However, for real-time accompaniment models, not all  $(t_f, k)$  pairs are deployable. Small  $t_f$  with small  $k$  yields highly reactive behavior with frequent updates and minimal recency gap, but little tolerance to variable delays. Moderate  $k$  with a  $t_f$  small enough to cover expected delays enables regular opportunities to adapt while remaining stable.

Let  $\tau_{\text{sys}} > 0$  denote the end-to-end system latency from sensing input to rendering output, and let  $\tau_{\text{jitter}} > 0$  denote a safety margin for variability. A latency condition for a practical system is  $t_f \geq \tau_{\text{sys}} + \tau_{\text{jitter}}$ . Let  $t_{\text{gen}}(k)$  be the average wall time required to generate a



**Fig. 2.** We implement streamed generation models as decoder-only Transformer conditioned on two streams. (Left) Audio of input tracks is mixed and both the mixture and the target stem are tokenized into RVQ codes by a DAC codec. (Middle) We apply the delay pattern to the output stream and prepend instrument token, then align the input and the output with padding to realize  $t_f$ . (Right) We embed input via frozen DAC codebook and output tokens with a learned embedding, before added and fed to the model, which predicts the next  $k$  frames.

chunk of duration  $k$ , resulting in a minimum update rate of  $r_{\text{update}} = 1/k$ . A throughput condition is  $t_{\text{gen}}(k) \leq k$ , so that generation keeps pace with playback. These conditions delimit a feasible region in the  $(t_f, k)$  plane.

### 3.5. Connecting to Offline Models in Unified Design Space

While practical real-time accompaniment occupies only a subspace of the  $(t_f, k)$  design plane, Eq. (1) provides a unified formulation that also covers offline regimes with future context. Ignoring  $x$  yields a decoder-only unconditional generative model on  $y$ . Allowing lookahead by taking  $t_f > 0$  grants access to future input; with full input  $x(1:T)$  available (that is,  $t_f = T$ ), setting  $k = 1$  recovers encoder-decoder autoregressive decoding, whereas setting  $k = T$  recovers non-autoregressive parallel generation models such as diffusion [26] or masked language modeling (MLM) [4, 27]. Prior streaming or real-time systems instantiate specific points in the  $(t_f, k)$  plane defined by Eq. (1). Anticipatory Music Transformer [28] uses a fixed lookahead and corresponds to  $t_f = \delta > 0$ , while ReaLchords [7] corresponds to  $t_f = 0$  and  $k = 1$ . Thus, by sweeping  $t_f$  and  $k$  we interpolate smoothly from online to offline, and from single-step prediction to chunked prediction.

## 4. EXPERIMENT SETUP

We study latency-responsiveness trade-offs by training a grid of  $(t_f, k)$  streaming models on the Slakh2100 dataset. We implement decoder-only Transformers operating on RVQ tokens, and we include two offline models for reference. We simulate streamed generation on the test set and evaluate 1024 examples using three metrics for overall coherence, rhythmic alignment and audio quality.

### 4.1. Dataset and Audio Representation

We use Slakh2100 dataset [11], which contains 145 hours of multi-track audio rendered from MIDI with professional virtual instruments. Each song includes a variable number of instrument stems and aligned MIDI. Following [4], for each training example we randomly select one target stem from the  $N$  available stems, then form the input mixture by sampling  $N_{\text{in}} \sim \text{Uniform}\{1, \dots, N-1\}$  stems from the remaining set and mixing them. We condition the model on the 18 General MIDI instrument categories [29], similar to [4].

We tokenize audio with a residual vector-quantized (RVQ) neural codec. Concretely, we train a Describe Audio Codec (DAC) model and use it to represent both the input mixture and the target

stem as sequences of  $N_q$  discrete codebook indices per frame [12]. The DAC we train encodes and reconstructs audio at 32kHz, and produces audio codes at 50 Hz. We train a DAC with 10 quantizer levels but use the first four levels for modeling. To control causal dependencies for real-time use, we implement a causal variant of the codec that only encodes past context for each frame of tokens [30].

### 4.2. Model Architecture

We implement the streaming distribution in Eq. (1) with a decoder-only Transformer [10] (Fig. 2). The backbone is a LLaMA-style [31] Transformer with 16 layers, 16 heads, and hidden size 1024. For output tokenization across RVQ levels, we adopt the “delay pattern” from MusicGen [32], which delays higher RVQ levels by one step relative to the coarse level. We choose the delay pattern rather than flattening RVQ levels into the sequence and predicting them serially, in order to minimize latency. With  $N_q = 4$  at 50 Hz, the delay pattern adds only  $((N_q - 1)/50) \text{ s} = 0.08 \text{ s}$  of computation delay. By contrast, a flattened scheme requires  $N_q$  serial predictions per frame and thus inflates per-frame computation by a factor of  $N_q$ .

Let  $\tau \in \{1, \dots, T_{\text{dac}}\}$  index codec frames. Let  $\mathbf{e}_\tau^x \in \mathbb{R}^{d_{\text{dac}}}$  and  $\mathbf{e}_\tau^y \in \mathbb{R}^d$  denote the per-frame embeddings for the input mixture and the target output, respectively. We obtain  $\mathbf{e}_\tau^x$  by summing the frozen DAC codebook embeddings across RVQ levels for frame  $\tau$ , and we obtain  $\mathbf{e}_\tau^y$  by summing a learned output codebook’s embeddings across RVQ levels using the delay interleaving pattern:

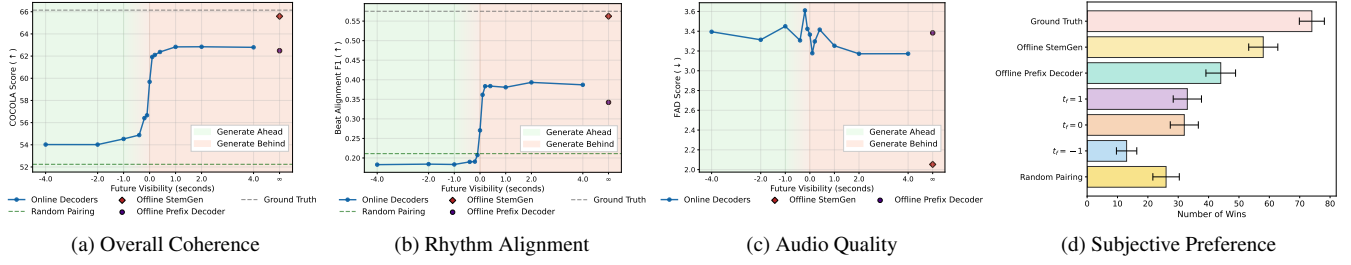
$$\mathbf{e}_\tau^x = \sum_{\ell=1}^{N_q} \text{Emb}_\ell^{\text{dac}}(c_{\tau,\ell}^x), \quad \mathbf{e}_\tau^y = \sum_{\ell=1}^{N_q} \text{Emb}_\ell^{\text{out}}(c_{\tau,\ell}^y). \quad (2)$$

Here  $c_{\tau,\ell}^x$  and  $c_{\tau,\ell}^y$  are the RVQ code indices at level  $\ell$  and frame  $\tau$  for the input and output streams,  $\text{Emb}_\ell^{\text{dac}}(\cdot) \in \mathbb{R}^{d_{\text{dac}}}$  are frozen embeddings from the trained DAC, and  $\text{Emb}_\ell^{\text{out}}(\cdot) \in \mathbb{R}^d$  are learned embeddings used only for the output stream.

We project the input embedding to the model dimension and fuse the streams with an additive gate that encourages the model to use the input rather than relying solely on its own history:

$$\mathbf{z}_\tau = \text{LN}(\mathbf{W} \text{LN}(\mathbf{e}_\tau^x)) + \mathbf{g} \odot \text{LN}(\mathbf{e}_\tau^y), \quad (3)$$

where  $\text{LN}(\cdot)$  is layer normalization,  $\mathbf{W} \in \mathbb{R}^{d \times d_{\text{dac}}}$  projects  $\mathbf{e}_\tau^x$  to the model dimension  $d$ , and  $\odot$  denotes elementwise product. The vector  $\mathbf{g} \in \mathbb{R}^d$  is a learned elementwise gate initialized to zeros to bias the model toward conditioning on the input stream.



**Fig. 3.** Overall coherence, rhythm alignment, audio quality and subjective preference of generated accompaniment for streaming models with  $k = 1$  across different  $t_f$ .

We implement instrument conditioning with a learned lookup embedding placed at the beginning of the output sequence. For  $t_f < 0$  we pad the output stream with a PAD token, and for  $t_f \geq 0$  we pad the input stream with a PAD embedding so that Eq. (1) is respected.

For  $k = 1$  we train an autoregressive decoder with a causal mask. For  $k > 1$  we train a prefix decoder where the fused input stream serves as the prefix and the model predicts only the next  $k$  output frames. At inference,  $k = 1$  uses standard frame-wise autoregressive decoding that increments the fused input by one frame per step. For  $k > 1$  we use non-overlapping chunked autoregressive decoding: generate  $k$  frames, update the prefix, then generate the next chunk. Overlapped or fully parallel chunked prediction is left for future work.

#### 4.3. Model Training and Offline Baseline

We train two offline models for reference. A prefix decoder baseline concatenates the input and output in time and predicts only output tokens; under our abstraction this corresponds to  $t_f = -T$  and  $k = 1$ . A masked language model baseline follows StemGen over RVQ codes [4] corresponds to  $t_f = -T$  and  $k = T$ . The prefix decoder matches the streaming model size, while the masked model uses 20 layers with the same heads and hidden size.

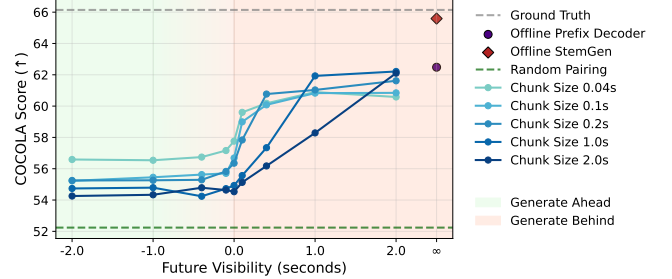
We train all models using cross entropy loss with the Adam [33] optimizer and the learning rate follows linear warmup for 10,000 steps then cosine decay, with a peak  $1 \times 10^{-4}$  and a floor  $1 \times 10^{-5}$  [34, 35]. Streaming models and the offline prefix decoder baseline use a batch size 64; the StemGen model uses a batch size 96.

#### 4.4. Evaluation

We assess accompaniment with three complementary objective metrics computed over the 1024 generated test clips: overall coherence, rhythmic alignment, and audio quality. For overall coherence, we use the pretrained COCOLA encoder [36] to compute a coherence score between the input mixture and the generated stem. COCOLA is trained in a self-supervised contrastive manner to capture harmonic and rhythmic coherence across musical audio. For rhythmic alignment, we estimate beats on both signals with a beat estimation model [37] and report the  $F_1$  score using Madmom [38] with default settings. For audio quality, we report Fréchet Audio Distance (FAD) computed on VGGish embeddings using a reference distribution built from the test split [39]. To validate the effectiveness of the objective metrics, we additionally conduct a listening study where participants blindly rate their preference between pairs of accompaniments from two randomly selected models for a given input.

### 5. RESULTS

We first fix  $k=1$  frame and sweep  $t_f \in \{\pm 4, \pm 2, \pm 1, \pm 0.4, \pm 0.2, 0\}$  seconds. Coherence (COCOLA, Fig. 3a) and rhythmic alignment



**Fig. 4.** Overall coherence for different  $k > 1$  frame and  $t_f$ .

(Beat  $F_1$ , Fig. 3b) both show a sharp improvement when moving from  $t_f \leq 0$  to small positive  $t_f$ , with the largest gain near  $t_f \in [-0.2, 0.2]$ . For  $t_f \geq 1$  s or  $t_f \leq -1$  s the curves largely saturate. The offline StemGen baseline at  $t_f = \infty$  outperforms all streaming models, while the offline prefix decoder is weaker but competitive. Streaming models with small positive  $t_f$  approach, and sometimes slightly exceed, the offline prefix decoder, whereas  $t_f \leq 0$  lies near the random-pairing floor on both metrics. The preference ordering on a selection of models (Fig. 3d) mirrors the objective metrics: offline models are preferred over streaming models, with larger  $t_f$  preferred more. Notably, all sources of accompaniment except the random pairing are statistically significantly outperform  $t_f = -1$ , showing that a practical accompaniment model with a reasonable delay results in poor subjective evaluations under supervised training. Audio quality (FAD, Fig. 3c) remains broadly stable with a mild improvement for positive  $t_f$  and small fluctuations around  $t_f = 0$ .

We next vary  $k \in \{0.04, 0.1, 0.2, 1, 2\}$  seconds across  $t_f \in \{\pm 2, \pm 1, \pm 0.4, \pm 0.2, 0\}$  seconds and evaluate overall coherence (Fig. 4). Within each  $k$ , the  $t_f$  trend mirrors the  $k=1$  case. Across  $k$ , larger chunks help when  $t_f > 0$  (more buffering and stable decoding) and hurt when  $t_f < 0$  because a larger fraction of each chunk lies beyond the visible input. In particular, for  $k > t_f > 0$ , the trailing portion  $k - t_f$  is effectively unconditioned on recent input, which degrades coherence. Very small  $k$  can also underperform, likely due to inefficient training where each example provides only  $k$  frames of supervised targets.

### 6. CONCLUSIONS AND FUTURE IMPLICATIONS

We formalize real-time acoustic accompaniment as streaming chunked prediction with the design variables future visibility  $t_f$  and chunk duration  $k$ , and we evaluate a grid of supervised models. Across objective metrics and the listening study, accompaniment quality improves with more input context and more frequent up-

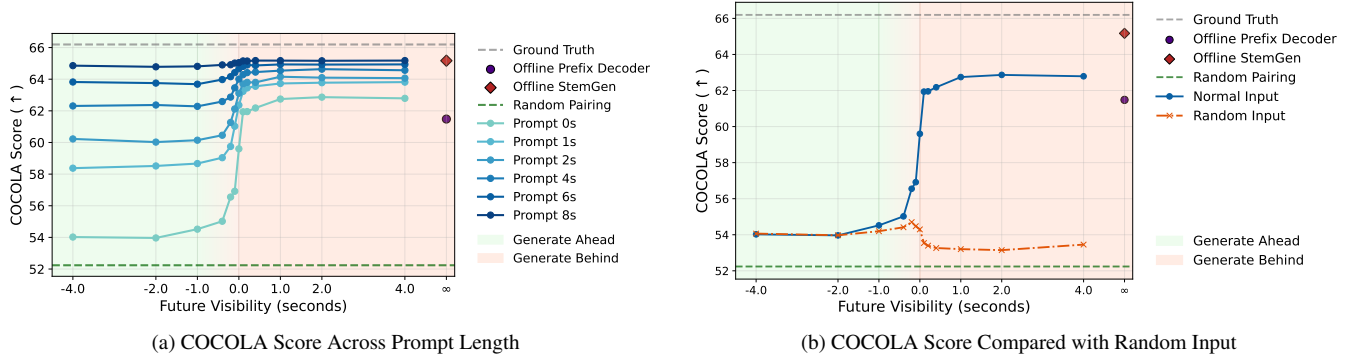
dates; however, practical deployment requires  $t_f < 0$  to offset latency, which drastically degrades quality. Consistent with prior work, training on well-composed datasets is insufficient, since such datasets rarely contain errors, corrective maneuvers, or co-adaptive behavior. These results motivate developing training objectives that explicitly encode anticipation and coordination, laying a foundation for future research on real-time audio accompaniment models.

## 7. REFERENCES

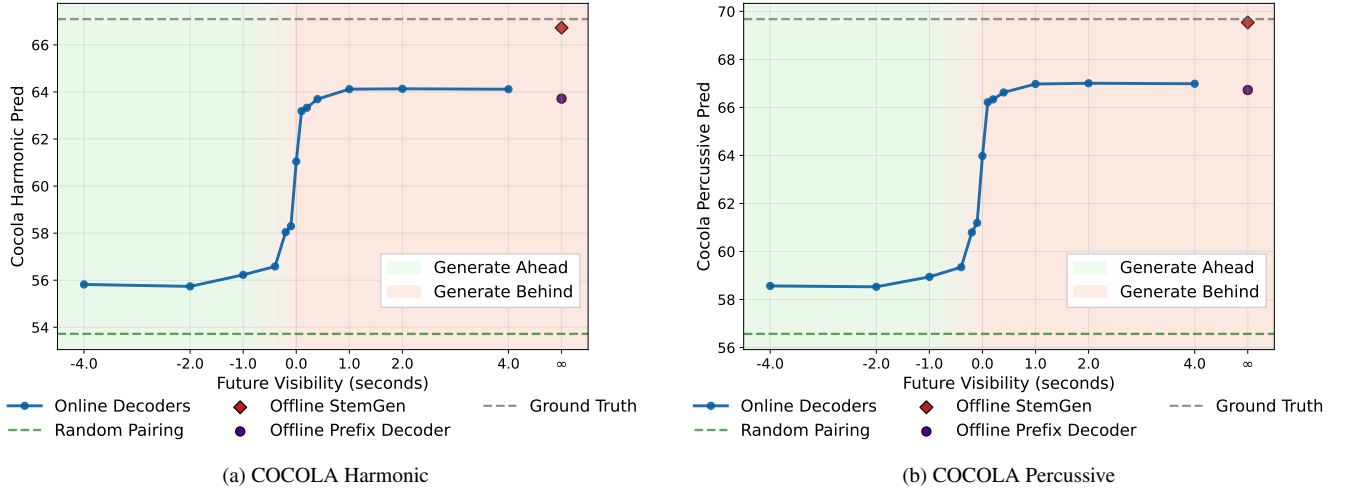
- [1] Peter Keller, “Joint action in music performance,” in *Enacting Intersubjectivity: A Cognitive and Social Perspective to the Study of Interactions*. 2008.
- [2] William J Wrigley and Stephen B Emmerson, “The experience of the flow state in live music performance,” *Psychology of Music*, 2013.
- [3] Wiebke Trost, Caitlyn Trevor, Natalia Fernandez, et al., “Live music stimulates the affective brain and emotionally entrains listeners in real time,” *Proceedings of the National Academy of Sciences*, 2024.
- [4] Julian D Parker, Janne Spijkervet, Katerina Kosta, et al., “Stemgen: A music generation model that listens,” in *ICASSP*. IEEE, 2024.
- [5] Chris Donahue, Antoine Caillon, Adam Roberts, et al., “Singsong: Generating musical accompaniments from singing,” *ArXiv*, 2023.
- [6] Simon Rouard, Robin San Roman, Yossi Adi, et al., “Musicgen-stem: Multi-stem music generation and edition through autoregressive modeling,” in *ICASSP*, 2025.
- [7] Yusong Wu, Tim Cooijmans, Kyle Kastner, et al., “Adaptive accompaniment with realchords,” in *ICML*, 2024.
- [8] Christodoulos Benetatos, Joseph VanderStel, and Zhiyao Duan, “Bachduet: A deep learning system for human-machine counterpoint improvisation,” in *NIME*, 2020.
- [9] Zihao Wang, Kejun Zhang, Yuxing Wang, et al., “Songdriver: Real-time music accompaniment generation without logical latency nor exposure bias,” in *ACM MM*, 2022.
- [10] Ashish Vaswani, Noam Shazeer, Niki Parmar, et al., “Attention is all you need,” *NeurIPS*, 2017.
- [11] Ethan Manilow, Gordon Wichern, Prem Seetharaman, et al., “Cutting music source separation some slakh: A dataset to study the impact of training data quality and quantity,” in *WASPAA*. IEEE, 2019.
- [12] Rithesh Kumar, Prem Seetharaman, Alejandro Luebs, et al., “High-fidelity audio compression with improved rvqgan,” *NeurIPS*, 2023.
- [13] Roger B Dannenberg, “An on-line algorithm for real-time accompaniment,” in *ICMC*, 1984.
- [14] Christopher Raphael, “Music plus one and machine learning,” in *ICML*, 2010.
- [15] Arshia Cont, “Antescofo: Anticipatory synchronization and control of interactive parameters in computer music,” in *ICMC*, 2008.
- [16] George E Lewis, “Too many notes: Computers, complexity, and culture in voyager,” in *New Media*. Routledge, 2003.
- [17] Gérard Assayag, Georges Bloch, Marc Chemillier, et al., “Omax brothers: a dynamic yopology of agents for improvisation learning,” in *ACM workshop on Audio and music computation multimedia*, 2006.
- [18] Jérôme Nika and Marc Chemillier, “Improtek: integrating harmonic controls into improvisation in the filiation of omax,” in *ICMC*, 2012.
- [19] Jérôme Nika, Marc Chemillier, and Gérard Assayag, “Improtek: introducing scenarios into human-computer music improvisation,” *Computers in Entertainment (CIE)*, 2017.
- [20] Nan Jiang, Sheng Jin, Zhiyao Duan, et al., “RL-duet: On-line music accompaniment generation using deep reinforcement learning,” in *AAAI*, 2020.
- [21] Alexander Scarlatos, Yusong Wu, Ian Simon, et al., “Realjam: Real-time human-ai music jamming with reinforcement learning-tuned transformers,” in *CHI EA*, 2025.
- [22] Lyria Team, Antoine Caillon, Brian McWilliams, et al., “Live music models,” *ArXiv*, 2025.
- [23] O. J. M. Smith, “A controller to overcome dead time,” *ISA Journal*, 1959.
- [24] Maximilian Schwenzer, Muzaffer Ay, Thomas Bergs, et al., “Review on model predictive control: an engineering perspective,” *The International Journal of Advanced Manufacturing Technology*, 2021.
- [25] Kevin Black, Manuel Y Galliker, and Sergey Levine, “Real-time execution of action chunking flow policies,” *ArXiv*, 2025.
- [26] Ke Chen, Yusong Wu, Haohe Liu, et al., “Musicldm: Enhancing novelty in text-to-music generation using beat-synchronous mixup strategies,” in *ICASSP*. IEEE, 2024.
- [27] Hugo Flores Garcia, Prem Seetharaman, Rithesh Kumar, et al., “Vampnet: Music generation via masked acoustic token modeling,” *arXiv preprint arXiv:2307.04686*, 2023.
- [28] John Thickstun, David Leo Wright Hall, Chris Donahue, and Percy Liang, “Anticipatory music transformer,” *Transactions on Machine Learning Research*, 2024.
- [29] MIDI Manufacturers Association, “Complete MIDI 1.0 detailed specification,” <http://www.midi.org/techspecs/gm.php>, 1999, Updated 2008.
- [30] Shih-Lun Wu, Aakash Lahoti, Arjun D Desai, et al., “Towards codec-LM co-design for neural codec language models,” in *NAACL SRW*, 2025.
- [31] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al., “The llama 3 herd of models,” *ArXiv*, 2024.
- [32] Jade Copet, Felix Kreuk, Itai Gat, et al., “Simple and controllable music generation,” *NeurIPS*, 2024.
- [33] Diederik P. Kingma and Jimmy Ba, “Adam: A method for stochastic optimization,” in *ICLR*, 2015.
- [34] Priya Goyal, Piotr Dollár, Ross Girshick, et al., “Accurate, large minibatch sgd: Training imagenet in 1 hour,” *ArXiv*, 2017.
- [35] Ilya Loshchilov and Frank Hutter, “Sgdr: Stochastic gradient descent with warm restarts,” in *ICLR*, 2017.
- [36] Ruben Ciranni, Giorgio Mariani, Michele Mancusi, et al., “Cola: Coherence-oriented contrastive learning of musical audio representations,” in *ICASSP*. IEEE, 2025.
- [37] Francesco Foscari, Jan Schlüter, and Gerhard Widmer, “Beat this! accurate beat tracking without dbn postprocessing,” in *ISMIR*, 2024.

- [38] Sebastian Böck, Filip Korzeniowski, Jan Schlüter, et al., “madmom: a new Python Audio and Music Signal Processing Library,” in *ACM MM*, 2016.
- [39] Kevin Kilgour, Mauricio Zuluaga, Dominik Roblek, et al., “Fr\’echet audio distance: A metric for evaluating music enhancement algorithms,” *ArXiv*, 2018.
- [40] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu, “On layer normalization in the transformer architecture,” in *Proceedings of the 37th International Conference on Machine Learning*, 2020, arXiv:2002.04745.
- [41] Biao Zhang and Rico Sennrich, “Root mean square layer normalization,” *arXiv preprint arXiv:1910.07467*, 2019.
- [42] Zhen Qin, Dong Li, Weigao Sun, Weixuan Sun, Xuyang Shen, Xiaodong Han, Yunshen Wei, Baohong Lv, Xiao Luo, Yu Qiao, and Yiran Zhong, “Transnormerllm: A faster and better large language model with improved transnormer,” *arXiv preprint arXiv:2307.14995*, 2023.
- [43] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu, “Roformer: Enhanced transformer with rotary position embedding,” *arXiv preprint arXiv:2104.09864*, 2021.
- [44] Alex Henry, Prudhvi Raj Dachapally, Shubham Pawar, and Yuxuan Chen, “Query-key normalization for transformers,” in *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020, arXiv:2010.04245.
- [45] Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, Rodolphe Jenatton, Lucas Beyer, Michael Tschannen, Anurag Arnab, Xiao Wang, Carlos Riquelme, Matthias Minderer, Joan Puigcerver, Utku Evci, Manoj Kumar, Sjoerd van Steenkiste, Gamaleldin F. Elsayed, Aravindh Mahendran, Fisher Yu, Avital Oliver, Fantine Huot, Jasmijn Bastings, Mark Patrick Collier, Alexey Gritsenko, Vighnesh Birodkar, Cristina Vasconcelos, Yi Tay, Thomas Mensink, Alexander Kolesnikov, Filip Paveti’c, Dustin Tran, Thomas Kipf, Mario Lu’ci’c, Xiaohua Zhai, Daniel Keysers, Jeremiah Harmsen, and Neil Houlsby, “Scaling vision transformers to 22 billion parameters,” *arXiv preprint arXiv:2302.05442*, 2023.
- [46] Noam Shazeer, “Glu variants improve transformer,” *arXiv preprint arXiv:2002.05202*, 2020.
- [47] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai, “Gqa: Training generalized multi-query transformer models from multi-head checkpoints,” in *Proceedings of EMNLP 2023*, 2023, arXiv:2305.13245.





**Fig. 5.** Accompaniment performance for streaming models with  $k = 1$  evaluated across different future visibility  $t_f$ . Left: performance under different prompt length. Right: performance when conditioning on the paired input compared with conditioning on a random input.



**Fig. 6.** COCOLA harmonic score and COCOLA percussive score of generated accompaniment for streaming models with  $k = 1$  across different  $t_f$ .

## A. APPENDIX

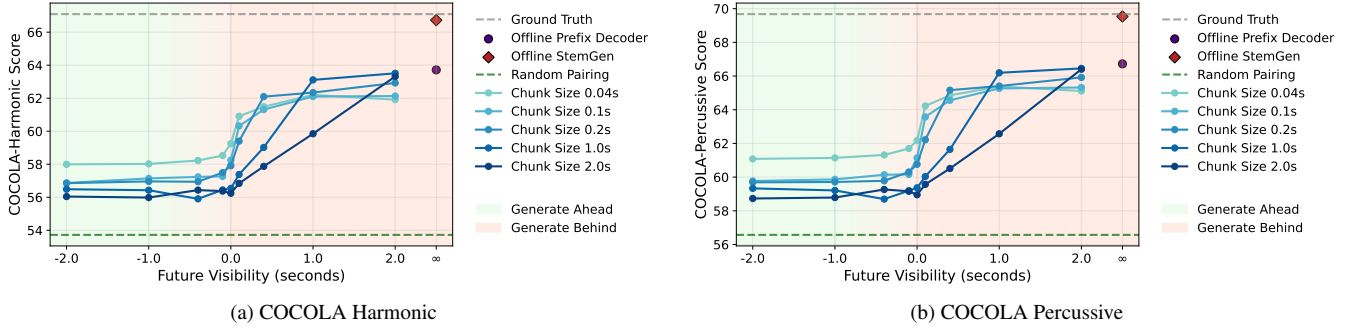
### A.1. Acknowledgment

We would like to thank Ke Chen for the discussion on this project.

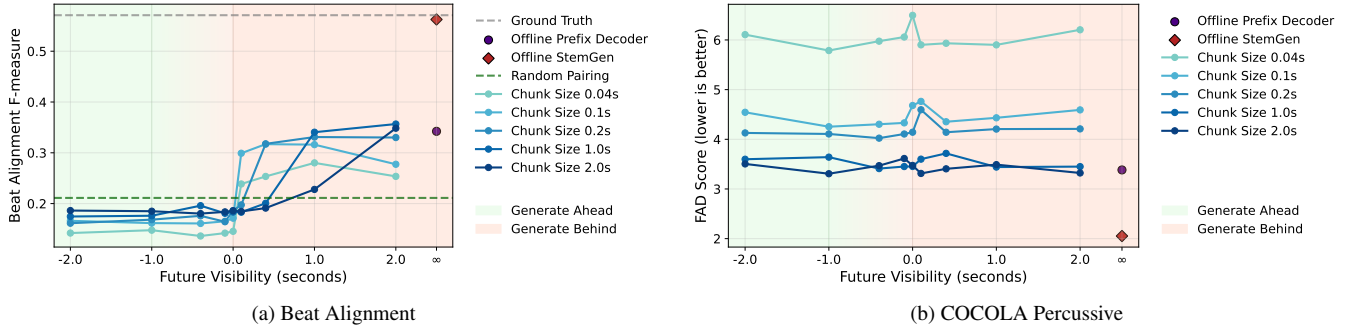
### A.2. Additional Results

We further investigate how each model configuration attends to the input and output. We run streamed generation while pairing each target stem with a randomly chosen input from the test set, and we compare COCOLA against the true paired input. As shown in Fig. 5b, the scores with true and random inputs are almost indistinguishable when  $t_f \leq 2$ , which indicates that the decoder relies mainly on its own history and the instrument token under low or negative visibility. For small positive  $t_f$ , the gap increases, which implies that the model begins to exploit the input stream for both harmonic and rhythmic cues. This is consistent with the main results where coherence and beat alignment improve as lookahead increases. We warm start decoding by providing a ground truth prefix of both input and output with duration  $L$  seconds, then start streaming. Fig. 5a shows that gains from prompting are largest when  $t_f \leq 0$ , and decrease as  $t_f$  grows. This suggests that, in low future-visibility regimes, a short history reduces exposure bias and stabilizes local decisions, whereas with more lookahead the model already observes sufficient recent context and benefits less from a longer prefix.

We include the COCOLA harmonic score and COCOLA percussive score of generated accompaniment for streaming models with  $k = 1$  across different  $t_f$  in Fig. 6. For streaming models with combinations of  $k > 1$  and  $t_f$ , we include COCOLA harmonic score and COCOLA percussive score in Fig. 7, and beat alignment and FAD score in Fig. 8.



**Fig. 7.** COCOLA harmonic score and COCOLA percussive score of generated accompaniment for streaming models with combinations of  $k > 1$  and  $t_f$ .



**Fig. 8.** Beat alignment and FAD score of generated accompaniment for streaming models with combinations of  $k > 1$  and  $t_f$ .

### A.3. Dataset Details

When selecting input-output track pairs and a start time for each 10 s window, we filter by silence and instrument class. We compute A-weighted short-time RMS at 50 Hz and label frames with level below  $-60$  dB as silent. We then reject a window if the input mixture and the target stem do not have at least 50% overlap of non-silent frames. When choosing the target stem, we exclude vocal classes.

### A.4. Transformer Backbone

We use a transformer backbone similar to Llama 3. Following current large-model practice, the network uses pre layer normalization throughout [40], with Root Mean Square Normalization as the normalization operator [41]; we adopt the simplified RMSNorm variant reported to work well at large scale [42]. Positional information is injected by rotary position embedding in self-attention [43]. Inside attention, we apply query-key normalization by normalizing queries and keys along the head dimension before the similarity computation [44], together with a dimension-dependent scaling factor on the normalized scores as recommended by recent scaling results [45]. The feed-forward sublayers use the gated SwiGLU activation [46]. To improve decoding efficiency while preserving quality, key-value projections are shared across groups of query heads, i.e., grouped-query attention [47]. The decoder also supports cross-attention for conditioning on external context.

### A.5. Model Implementation Details

For all prefix-decoder models, we do not use a bidirectional mask on the prefix. Instead, we apply a single causal mask over the entire sequence so that every position attends only to past positions.

For the StemGen masked language model, we found a VampNet-style [27] confidence ranking to outperform the original StemGen ranking. At each sampling iteration and for each RVQ level, we compute a confidence score for token  $\hat{y}_t$  as

$$\text{conf}(\hat{y}_t) = \log p(\hat{y}_t) + \text{temp} \cdot g_t, \quad (4)$$

where  $p(\hat{y}_t)$  is the model probability of  $\hat{y}_t$ ,  $g_t \sim \text{Gumbel}(0, 1)$  is i.i.d., and  $\text{temp}$  is linearly annealed to 0 over the sampling iterations.

### A.6. Streaming Prefix Decoder Details

In the streaming setting with chunk size  $k > 1$ , we train a prefix decoder. For each minibatch, we sample a prefix length  $\ell$  uniformly from  $\{0, k, 2k, \dots, T - k\}$ . Given  $\ell$ , we construct each example by aligning inputs and outputs up to step  $\ell$ , then require the model to predict the



next  $k$  steps ( $\ell + 1, \dots, \ell + k$ ) under a causal attention mask. The loss is computed only on these  $k$  target steps, and gradients are applied exclusively to those positions, while earlier tokens serve as context without direct supervision. This variable-prefix sampling exposes the model to a range of prefix boundaries and supports chunked next-k prediction during streaming inference.

### A.7. Model Sampling Details

For all decoder-only Transformers, we sample with softmax temperature 1.0 and top- $k = 200$ . For the StemGen model, we use per-level maximum noise temperatures [8.0, 8.0, 4.0, 4.0] for RVQ levels  $\ell = 1, \dots, 4$ , and [128, 64, 32, 32] sampling steps per level. The StemGen model is trained with input dropout probability 20% (the input embedding is zeroed when dropped). At sampling time we use classifier-free guidance with scale 2.0.

For online streaming models, the total input it ever conditioned on is  $t_f + T$ . That is, if  $t_f > 0$ , the model see extra input stream, and vice versa.

### A.8. Listening Study Details

We ran a listening study in which 24 participants evaluated the models in this study as well as ground truth and random pairing examples. Participants blindly evaluated the models by indicating their preference between pairs of accompaniments for a given input. A Kruskal-Wallis H test and confirmed that there are statistically significant pairs among the permutations. We evaluate significance with a post-hoc analysis using the Wilcoxon signed-rank test with Bonferroni correction (with  $p/0.05/21$  as there are 7 models evaluated).

To ease the participant’s cognitive load, we select samples with six or fewer input tracks for inclusion in the listening test.

	Ground Truth	Offline StemGen	Offline Prefix Decoder	$t_f = 1$	$t_f = 0$	$t_f = -1$	Random Pairing
Ground Truth	N/A	!	*	*	*	*	*
Offline StemGen	!	N/A	!	!	!	*	*
Offline Prefix Decoder	*	!	N/A	!	!	*	!
$t_f = 1$	*	!	!	N/A	!	*	!
$t_f = 0$	*	!	!	!	N/A	*	!
$t_f = -1$	*	*	*	*	*	N/A	!
Random Pairing	*	*	!	!	!	!	N/A

**Table 1.** Pairwise statistical significance results for listening study. \* indicates significant difference ( $p < 0.05/21$ ), ! indicates non-significant ( $p > 0.05/21$ ).

### A.9. Audio Mixing

For all objective evaluations and the listening study, we use a fixed loudness pipeline. First, we loudness-normalize the predicted track and the target track to  $-18$  dB. When forming the mix, all tracks in the input mixture are summed at equal loudness, and the predicted (or target) track is mixed  $+5$  dB relative to each input track. Finally, we normalize the resulting mixture to  $-18$  dB.