Live Music Models

Lyria Team, Google DeepMind[1]

Abstract

We introduce a new class of generative models for music called *live music models* that produce a continuous stream of music in real-time with synchronized user control. We release Magenta RealTime, an open-weights live music model that can be steered using text or audio prompts to control acoustic style. On automatic metrics of music quality, Magenta RealTime outperforms other open-weights music generation models, despite using fewer parameters and offering first-of-its-kind live generation capabilities. We also release Lyria RealTime, an API-based model with extended controls, offering access to our most powerful model with wide prompt coverage. These models demonstrate a new paradigm for AI-assisted music creation that emphasizes human-in-the-loop interaction for live music performance.

Magenta RealTime (Open): `github.com/magenta/magenta-realtime`
Lyria RealTime (API): `g.co/magenta/lyria-realtime`

1 Introduction

Music exists in two complementary forms: as static recorded pieces ("music as a noun"), and as live performances collectively experienced in real time ("music as a verb") [1]. This second form of *live* music is particularly tied to the fundamental human experiences of creative flow [2, 3], embodied expression [4], and social connection [5]. Despite this, modern generative AI systems for musical audio have had an overwhelming emphasis on offline, turn-based generation [6–15].

Live music represents a new frontier for generative AI, one with numerous opportunities and technical challenges. In the conventional offline setting, users input control information, wait L seconds (offline *latency*), and receive T seconds of audio. In our proposed live setting, users continuously input control information, receiving T seconds of an uninterrupted audio stream from T seconds of interaction, with D seconds of *delay* between their control inputs and their influence on the audio stream. Placing users in a continuous perception-action loop promotes more active creation, creates higher-bandwidth interaction, fosters personalized expression and emphasize the process as much as the product. However, meeting these rigid synchronization requirements while maintaining high quality audio generation is a challenging task for machine learning models.

Some offline music generation systems [16] have a Real Time Factor (RTF) r$\geq 1\times$, i.e., they generate T seconds of audio with latency $L \leq T$.[2] However, most are not well-suited for live performance, as they lack other necessary attributes. Specifically, we differentiate *live music models* as those which have all three of the following attributes: (1) *real-time generation* with throughput RTF $\geq 1\times$, (2) *causal streaming* where audio generates continuously as a function of both user control inputs and past audio output, and (3) *responsive controls* (delay D is low, facilitating live interaction).

Open-weights models are particularly well-suited for live generative music because they can run locally on users' devices. On-device inference has numerous benefits in music [17], enabling (1) *lower latency* by eliminating network requests, (2) *higher reliability*, facilitating usage in real-world contexts, (3) *privacy* guarantees, and (4) *customization* by artists. Cloud-based APIs address complementary use cases, offering access to models running on specialized hardware that are more powerful than those that would run on edge devices, at the cost of some of the benefits of open-weights models.

[1] See Contributions and Acknowledgments section for full author list.

[2] RTF is commonly defined as both L/T and T/L. Here we use T/L, i.e., higher RTF means faster.

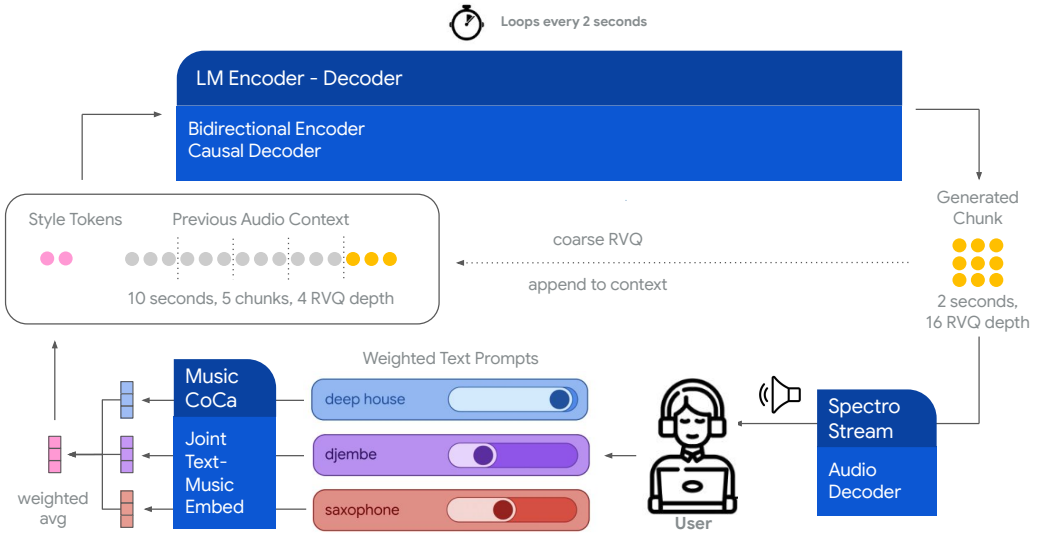

Figure 1: Magenta RealTime is a *live music model* that generates an uninterrupted stream of music and responds continuously to user input. It generates audio in two-second chunks using a pipeline with three components: (1) MusicCoCa, a *style embedding* model, (2) SpectroStream [20], an *audio codec* model, and (3) an encoder-decoder *language model*. For each chunk, a style embedding is computed via a weighted average of MusicCoCa embeddings of text and audio prompts from the user. Given this style embedding and 10 seconds (5 chunks) of past audio context, the language model decoder generates SpectroStream audio tokens for the new chunk, which is then decoded to audio.

To allow users to navigate these tradeoffs based on their application goals, we introduce a pair of systems that span both paradigms: Magenta RT (open-weights, on-device) and Lyria RT (API, cloud-based). Both use the same core methodological framework, which centers around codec language modeling [18, 19] (Figure 1). Specifically, we train a language model (LM) to generate audio tokens from SpectroStream [20], using a method similar to [21, 22] to achieve live streaming [23].

We focus our subsequent discussion primarily on Magenta RT, which may be of higher interest to the AI research community for its ability to be finetuned with new controls [24] or explored for transfer learning [25]. Magenta RT offers first-of-its-kind live generation among open-weights models. On automatic metrics of music quality, it outperforms existing offline open-weights music generation models like MusicGen (Large) [9] and Stable Audio Open [14]. Moreover, Magenta RT uses 38% fewer parameters (750M) than Stable Audio Open (1.2B), and 77% fewer than MusicGen Large (3.3B). Along with our codebase and model weights, we release a set of demos that run in real time on free-tier Colab TPUs (`v2-8`) and showcase three distinct use cases: live generation, finetuning, and a novel live audio input interaction that we call *audio injection* (Section 4.2). [3]

2 Method

Magenta RT is a codec language model [18, 19] designed to generate high-fidelity stereo audio in real time based on acoustic style conditioning provided by a user. To achieve this, we adopt a pipeline-based approach, using an encoder-decoder Transformer [26] to model audio tokens from SpectroStream [20] conditioned on style embeddings from our proposed MusicCoCa (Figure 1). See Appendix A for related work.

2.1 Audio tokenization via SpectroStream

Codec language modeling involves the use of a discrete audio codec to convert audio data into language-like *tokens*. A codec is a pair of functions, an encoder and decoder, that convert audio to and from a compressed space with minimal perceivable distortion. More formally, the encoder component `Enc` is a function mapping raw stereo audio waveforms $\mathbf{a} \in \mathbb{R}^{Tf_s \times 2}$ into sequences of

[3] Audio samples and links to code, weights and demos are provided in the online supplement: `https://storage.googleapis.com/live-music-models/index.html`

discrete tokens $\mathbb{V}_c^{Tf_k \times d_c}$, where T is the duration in seconds, f_s the audio sampling rate, $\mathbb{V}_c$ the codec vocabulary, f_k the token frame rate, and d_c the RVQ depth. The decoder module $\texttt{Dec} \approxeq \texttt{Enc}^{-1}$ then performs the approximate inverse operation to reconstruct the waveform given the compressed representation, ensuring that the process is as perceptually lossless as possible.

Here we adopt the recently proposed SpectroStream codec [20], a full-band ($f_s = 48$kHz) multi-channel neural audio codec based on residual vector quantization (RVQ) [27], with an overall bandwidth of 16kbps ($f_k = 25$Hz, $d_c = 64$, $|\mathbb{V}_c| = 1024$). To facilitate live streaming, we reduce the bandwidth to 4kbps by generating only the first 16 RVQ levels (coarse and medium from Figure 3), yielding a live throughput target of 400 tokens per second. See Appendix C.1 for more details.

2.2 Style embeddings via MusicCoCa

We use a joint audio-text representation as a control mechanism for overall audio style. This is achieved by training a joint audio-text embedding model on music annotated with diverse textual descriptions. The text encapsulates musical characteristics useful for high-level control, which we collectively refer to as *style* (Section 4.1).

Our embedding model, MusicCoCa, builds upon MuLan [28] and CoCa [29]. It is a contrastive captioner (CoCa) consisting of two embedding towers, mapping each modality to a shared 768-dimensional space. The audio embedding tower M_A is a 12-layer VisionTransformer (ViT) [30]. Its input is a log-mel spectrogram of a 10s slice of 16kHz audio (128 channels and length 992; split into patches of size 16×16). The text embedding tower M_T is a 12-layer Transformer, which operates on tokenized text with a maximum sequence length of 128 tokens. We use attention pooling to reduce the activations of each tower to a single 768d embedding, which can be subsequently quantized into 12 discrete tokens with codebook size $|\mathbb{V}_m| = 1024$. This tokenized representation (of audio or text) is then used as a conditioning signal in the LM encoder. Variable-length audio of duration T may be embedded by zero padding to a multiple of 10 seconds and then mean pooling across $\lceil \frac{T}{10} \rceil$ chunks.

In addition to the two embedding towers, MusicCoCa has a text decoder which can generate audio text captions. In our application this decoder is a shallow 3-layer Transformer which only serves a regularizing purpose. The MusicCoCa optimization objective, based on the CoCa framework [29], consists of contrastive and generative loss components which we weigh equally. We train the model using the Adafactor optimizer with a learning rate of 1×10^{-4} and $1{,}000$ warmup steps for a total of $16{,}000$ steps. The batch size is 1024.

2.3 Modeling Framework

Magenta RT operates on audio tokens from a discrete audio codec, following an established practice in audio modeling [8, 9, 18, 31]. Our model autoregressively predicts discrete audio tokens conditioned on both preceding audio and a shared audio-text embedding. This modeling mechanism partly mirrors the MusicLM architecture [8], itself based on AudioLM [31]. Similarly to MusicLM, we use a Transformer-based architecture [26] for the token prediction model and enable text conditioning by leveraging a joint audio-text embedding model, using the target audio signal at training time, and text inputs at inference time. To facilitate live generation, we propose two high-level changes relative to MusicLM: (1) we replace the hierarchical cascade of multiple LMs with a single LM, using a recent method [23] similar to [21, 22] for efficiency, and (2) we propose a chunk-based autogressive approach to allow for infinite streaming generation.

More formally, given audio $\mathbf{a}$, our goal is to model the probability $P_\theta(\texttt{Enc}(\mathbf{a}) \mid M_A(\mathbf{a}))$ of the corresponding sequence of acoustic tokens given its associated style embedding. Here, Enc denotes the audio codec encoder and M_A the MusicCoCa audio encoder. At inference, we sample from the model $\mathbf{e\prime} \sim P_\theta(\cdot \mid \mathbf{c})$, conditioning on a style embedding $\mathbf{c}$ obtained from an arbitrary mixture of audio and text prompts. Finally, we use the codec decoder to produce output audio, i.e., $\mathbf{a\prime} = \texttt{Dec}(\mathbf{e\prime})$.

2.3.1 Chunk-based autoregression with coarse context

In the live generation setting, we require our model to generate an infinite and uninterrupted stream of audio with a RTF $\geq 1\times$. To achieve this, we propose two key techniques: chunk-based autoregression to enable infinite streaming, and the use of coarser RVQ tokens in the audio history.

In order to generate an infinite stream of audio, at inference time we must be able to predict an audio sequence with a length likely larger than what the model has seen during training. Such a mismatch in sequence length between training and inference is often found to result in unpredictable behavior and degraded performance [26, 32]. Previous work has addressed this issue via sliding attention windows with a relative positional encoding scheme [33, 34], informing the model about the relative distance between tokens instead of describing their absolute position within the sequence.

We instead propose *chunk-based autoregression*, where we operate on chunks of length $C = 2$ seconds, and, under a Markov assumption, predict each chunk based on a limited context of $H = 5$ previous chunks (10 seconds of history). This has several advantages: it reduces error accumulation and allows for stateless inference, eliminating the need to maintain a generation cache and simplifying model deployment. It also introduces flexibility during sampling, since conditioning is updated between calls without preserving information about controls beyond the context window.

To achieve RTF $\geq 1\times$, we also propose to use a *coarse representation of the audio context*. Due to the hierarchical structure of RVQ, lower quantization levels capture the most salient acoustic information. While generating target tokens at the full RVQ depth ($d_c = 16$) is crucial to maintaining high fidelity, a lower resolution may be sufficient to represent the audio history. Therefore, we use a coarser representation consisting of the first 4 RVQ tokens for conditioning on the previous chunks.

More formally, a *chunk* is a contiguous segment of audio tokens representing C seconds of audio. For audio $\mathbf{a}$, we define $\texttt{Chunk}_i \triangleq \texttt{Enc}(\mathbf{a})_{Cf_k i:Cf_k(i+1)}$, i.e., the span of tokens representing audio between Ci and $C(i+1)$ seconds and including the first 16 RVQ tokens for each frame. We also define $\texttt{Coarse}_i$ as the first 4 RVQ tokens over the same span of time. Our de facto modeling objective is thus $P_\theta(\texttt{Chunk}_i \mid \texttt{Coarse}_{i-H:i}, \mathbf{c}_i)$, where $\mathbf{c}_i = \texttt{Quantize}(M_A(\mathbf{a})_{\lfloor \frac{Ci}{10} \rfloor})$ is 12 tokens representing the most recent quantized MusicCoCa audio embedding for chunk i.

2.4 Encoder-Decoder Language Model

To model this distribution, we use an encoder-decoder Transformer [26] LM trained with T5X [35, 36]. We release pre-trained models using the T5 [35] Base and Large configurations.

Encoder The bidirectional encoder is responsible for processing the acoustic history and style control into an intermediate representation for generation. At chunk i, the encoder receives the concatenation of the acoustic history and style tokens $\mathbf{x}_i = \texttt{Coarse}_{i-H} \oplus \cdots \oplus \texttt{Coarse}_{i-1} \oplus \mathbf{c}_i$, with a total length of 1012 tokens ($4 \cdot C \cdot H \cdot f_k = 1000$ audio + 12 style tokens) and a vocabulary unified across the codec and quantized style tokens $\mathbb{V} = \{\texttt{<S>}, \texttt{<P>}\} \cup \mathbb{V}_c \cup \mathbb{V}_m$.

Decoder A key differentiating factor compared to prior work is our imposed constraint of achieving RTF $\geq 1\times$ generation to enable live interactive applications. Past work such as MusicLM [8] proposes a hierarchical cascade of language models to model tokens efficiently, while MusicGen [9] proposes a delay pattern—neither approach would achieve RTF $\geq 1\times$ for full bandwidth audio tokens. Instead, based on [23], our decoder comprises two connected Transformer modules. The "temporal" module constructs a temporal context by processing acoustic frames, where RVQ tokens within each frame are embedded and aggregated to yield a single frame-level embedding. The "depth" module then performs autoregressive prediction of the RVQ indices conditioned on the previous temporal context. With this setup, we achieve RTF=1.8 on H100 GPU with the T5 Large configuration.

3 Experiments

Magenta RT, and live music models more broadly, enable new types of interaction that are best assessed through direct human evaluation requiring extensive use. As such, many of the design choices in Magenta RT were evaluated through play testing by team members and partner musicians, optimizing for subjective feeling of how expressive and engaging the resulting instrument is to play.

We complement this subjective measure with more constrained experiments to examine the effects of our controls and provide comparisons to existing models where possible. We focus these experiments on assessing core capabilities that are common to both live and offline music audio generation models, such as audio quality and adherence to text conditioning (Section 3.2.1), alongside others that are

Table 1: Instrumental music generation results on the Song Describer Dataset [40]. We compare to open models, using a fixed length of 47s for all samples, though our models are capable of arbitrary-length generation. For all prior models, we report results from [14].

Model	Live	Sample rate	Params	$\text{FD}_{openl3} \downarrow$	$\text{KL}_{passt} \downarrow$	$\text{CLAP}_{score} \uparrow$
Magenta RealTime	✓	48 kHz	760M	**72.14**	**0.47**	0.35
Stable Audio Open [14]	✗	44.1 kHz	1.1B	96.51	0.55	**0.41**
MusicGen-stereo-large [9]	✗	32 kHz	3.3B	190.47	0.52	0.31

unique to our live music models, such as the ability to generate musical transitions following changes in the conditioning signal (Section 3.2.2).

3.1 Experimental Set-up

Training We pretrain the LM at Base (220M parameters) and Large (770M parameters) size. The dataset comprises around 190,000 hours of primarily instrumental stock music sourced from various providers. Each training example consists of a 12-second audio clip randomly sampled from the raw data, structured as 10-second context tokens and 2-second target tokens. MusicCoCa tokens are derived from the target audio, of which the first 6 RVQ levels are used for training. To mitigate the cold-start issue in streaming, we replace early context tokens with variable-length padding tokens.

Each model is trained for 1.86 million steps using the Adafactor optimizer with batch size of 512 and an inverse square root learning rate schedule with 10,000 warmup steps. We use TPU-v6e (Trillium) hardware, with 128 chips for the Base model and 256 chips for the Large.

Sampling parameters For inference, prompts (text or audio) are embedded and tokenized by MusicCoca using the first 6 RVQ levels to match training. We sample with classifier-free guidance (CFG) [37–39], using a temperature of 1.3, a top-K of 40, and a CFG weight of 5.0.

3.2 Results

3.2.1 Audio quality and adherence to fixed text prompts

In this section, we compare Magenta RT to prior work under the offline text-to-music generation setting, where we keep the text conditioning fixed and sequentially generate chunks of audio up to a target length of 47 seconds. While our model can generate audio of arbitrary length, we choose this duration for a fair comparison between models. Following the evaluation protocol in [14], we then assess the quality and text adherence of the resulting generations using three established metrics, the Fréchet Distance based on OpenL3 embeddings [41] ($\text{FD}_{\text{openl3}}$), the Kullback–Leibler divergence (KL_{passt}) and $\text{CLAP}_{\text{score}}$.

We show the results in Table 1. Magenta RT has the lowest $\text{FD}_{\text{openl3}}$ and KL_{passt} scores, indicating that the generated audio is plausible and closely matches the eval reference audio, including at the level of semantic correspondence [14]. The $\text{CLAP}_{\text{score}}$ measures how well generated audio adheres to the specified text prompt. On this metric, Magenta RT scores between the other two models. The higher score of Stable Audio Open may be related to the fact that their model uses CLAP embeddings during training, as opposed to our model which trains using MusicCoCa.

3.2.2 Generating musical transitions

Live music models unlock the capability of responding dynamically to user inputs. To test this, we create a *prompt transition* evaluation. We pick pairs of text prompts and task the model with creating musical transitions between them. We linearly interpolate between MusicCoCa text embeddings of the start and end prompt over 60 seconds (6 steps, 10 seconds/step), and use this as style conditioning. We then measure cosine similarity between the audio embedding of the outputs and the conditioning embedding at that time. We perform transitions for 128 prompt pairs (Appendix F).

As seen in Figure 2 Magenta RT outputs maintain strong similarity to the target embedding throughout the transition (right panel), and effectively transitions from the initial to final prompt (left panel). The audio context conditioning lead to smooth transitions that blend styles by preserving elements

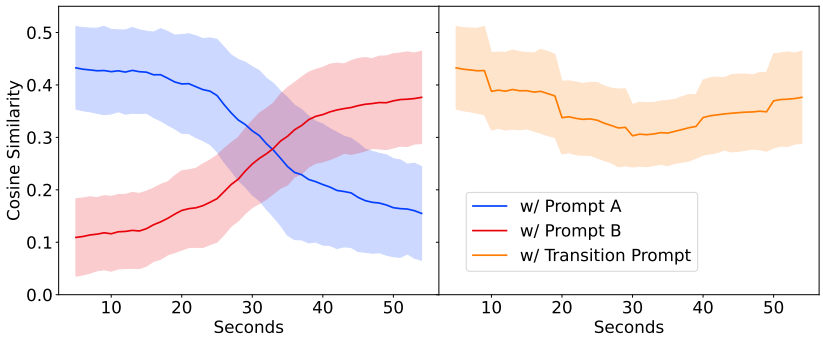

Figure 2: Prompt transition evaluation. Over 60s, we transition from embeddings of text prompt A to B by stepwise linear interpolation. Left: Cosine similarity compared to the initial (blue) and final (red) text embedding. Right: Cosine similarity to the interpolation between text embeddings provided to the model. In both plots, lines indicate the mean and shaded regions the standard deviation.

of the initial prompts. While this leads to lower similarity at end of transition and a slight dip in mid-transition similarity, it also lets the music continuously evolve in a smooth and coherent way, and makes the time history of prompts an important and expressive part of performance.

4 Controllable Generation

4.1 Style Conditioning via Text and Audio

During inference, we can create a target conditioning vector $\mathbf{c}$ by computing a weighted average of the MusicCoCa embeddings corresponding to N control prompts, provided as either text or audio $\mathbf{c} = \sum_{i=1}^{N} w_i M(\mathbf{c_i}) / \sum_i w_i$, where w_i controls the weight of each prompt. One advantage of using MusicCoCa embeddings instead of attending to text captions is the ability to perform embedding arithmetic to blend styles—e.g., a weighted sum of `embed`("techno") and `embed`("flute") gives a good approximation of `embed`("techno flute")—while also controlling the relative influence of each concept.

Beyond this, a shared audio-text embedding space for conditioning allows for the use of audio prompts as a more direct way of achieving a specific musical style or instrumentation that may be difficult to express via text. Since audio prompts more closely match the training setting, where style conditioning is obtained from the target audio itself, this type of prompting is also expected to be more effective. Furthermore, we are able to mix multiple audio prompts to achieve interpolations of different styles, or mix combinations of text and audio prompts. Overall, this type of conditioning offers control over high-level characteristics such as genre, style, instrumentation, mood.

4.2 Audio Injection

To allow users to continuously steer generation via a live audio stream, we propose an audio steering mechanism we call *audio injection*. At each generation step, we mix the user's input audio with the model's output, tokenize the resulting mix, and feed this as the context for the next generation. This is illustrated in Figure 7. Note, user audio is never played back directly. Rather, the model predicts the continuation of a past context that *includes* the user audio. Depending on the specific audio injected and the target style, the model may "choose" to repeat or transform the user audio, or may be influenced by its features (dynamics, melody, harmony). See Appendix E for details.

5 Conclusion

In this work, we introduced live music models, a new class of generative systems designed for real-time, continuous music creation with synchronized user control. We presented two such systems: Magenta RealTime, a fully open-weights model, and Lyria RealTime, an API-based model with extended controls. These models facilitate a novel paradigm for AI-assisted music, emphasizing

interactive, human-in-the-loop performance that prioritizes the creative process over just the end product. With future work, we aim to further decrease the control latency to unlock new interactive possibilities. Ultra-low latency could enable direct MIDI or audio control, akin to a new class of synthesizer or audio effect. Further, training on multi-stem audio would open the possibility for models to act as musical partners, jamming along with users and providing dynamic live accompaniment.

Contributions and Acknowledgments

Within each **Bolded Category** of contribution type, contributors are listed alphabetically.

Tech Leads

Adam Roberts
Chris Donahue
Kehang Han

Core Contributors

Antoine Caillon
Brian McWilliams
Cassie Tarakajian
Ian Simon
Ilaria Manco
Jesse Engel
Noah Constant
Yunpeng Li
Timo I. Denk

Contributors

Alberto Lalama
Andrea Agostinelli
Cheng-Zhi Anna Huang
Ethan Manilow
George Brower
Hakan Erdogan
Heidi Lei
Itai Rolnick
Ivan Grishchenko
Manu Orsini
Matej Kastelic
Mauricio Zuluaga
Mauro Verzetti
Michael Dooley
Ondrej Skopek
Rafael Ferrer
Zalán Borsos

Lyria RealTime API

Doug Fritz
Ivan Solovyev
Joyce (JingJing) Xie
Matthew Tang
Olivier Lacombe
Peter Morgan

Magenta RealTime Release

Gus Martins
Paige Bailey
Omar Sanseviero
Tris Warkentin

Product Management

Jeff Chang
Hema Manickavasagam
Myriam Hamed Torres

Legal

Austin Tarango
Phoebe Kirk

Program Management

DY Kim
Mahyar Bordbar
Moon Park

Executive Sponsors

Aäron van den Oord
Douglas Eck
Eli Collins
Jason Baldridge
Tom Hume

Acknowledgements

Many thanks to the entire Lyria team for their support and feedback.

Special thanks and acknowledgment to Alex Tudor, Arathi Sethumadhavan, Arturas Lapinskas, Ashu Desai, Beat Gfeller, Cătălina Cangea, Chris Deaner, Christian Frank, Colin McArdell, Damien Vincent, Eleni Shaw, Julian Salazar, Kazuya Kawakami, Marco Tagliasacchi, Matt Sharifi, Michael Chang, Reed Enger, RJ Skerry-Ryan, Ron Weiss, Sander Dieleman, Sertan Girgin, Tancred Lindholm, Tobenna Peter Igwe, Victor Ungureanu, and Yotam Mann.

Additional thanks to Chris Reardon, Mira Lane, Koray Kavukcuoglu, and Demis Hassabis for their insightful guidance and support throughout the research process.

MusicFX DJ was the first deployment of Lyria RealTime and developed in collaboration with our partners from Google Labs including Obed Appiah-Agyeman, Tahj Atkinson, Carlie de Boer, Phillip Campion, Sai Kiran Gorthi, Kelly Lau-Kee, Elias Roman, Noah Semus, Trond Wuellner, Kristin Yim, and Jamie Zyskowski. We give our deepest thanks to Jacob Collier, Ben Bloomberg, and Fran Haincourt for their valuable feedback throughout the development process.

We also acknowledge the many other individuals who contributed across Google DeepMind and Alphabet, including our partners at Envisioning Studio and YouTube.

References

[1] Christopher Small. *Musicking: The Meanings of Performing and Listening*. Wesleyan University Press, 1998.

[2] William J. Wrigley and Stephen B. Emmerson. The experience of the flow state in live music performance. *Psychology of Music*, 2013.

[3] Alice Chirico, Silvia Serino, Pietro Cipresso, Andrea Gaggioli, and Giuseppe Riva. When music "flows". State and trait in musical performance, composition and listening: a systematic review. *Frontiers in Psychology*, 2015.

[4] Alexander Refsum Jensenius. *Sound Actions: Conceptualizing Musical Instruments*. The MIT Press, 2022.

[5] Wiebke Trost, Caitlyn Trevor, Natalia Fernandez, Florence Steiner, and Sascha Frühholz. Live music stimulates the affective brain and emotionally entrains listeners in real time. *Proceedings of the National Academy of Sciences*, 2024.

[6] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music. *arXiv:2005.00341*, 2020. URL `https://arxiv.org/abs/2005.00341`.

[7] Seth Forsgren and Hayk Martiros. Riffusion - Stable diffusion for real-time music generation. 2022. URL `https://web.archive.org/web/20221215132646/https://riffusion.com/about`.

[8] 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:2301.11325*, 2023. URL `https://arxiv.org/abs/2301.11325`.

[9] Jade Copet, Felix Kreuk, Itai Gat, Tal Remez, David Kant, Gabriel Synnaeve, Yossi Adi, and Alexandre Défossez. Simple and controllable music generation, 2023.

[10] Chris Donahue, Antoine Caillon, Adam Roberts, Ethan Manilow, Philippe Esling, Andrea Agostinelli, Mauro Verzetti, Ian Simon, Olivier Pietquin, Neil Zeghidour, et al. SingSong: Generating musical accompaniments from singing. *arXiv:2301.12662*, 2023. URL `https://arxiv.org/abs/2301.12662`.

[11] 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 *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, 2024.

[12] Zach Evans, Julian D Parker, CJ Carr, Zack Zukowski, Josiah Taylor, and Jordi Pons. Long-form music generation with latent diffusion. *arXiv:2404.10301*, 2024. URL `https://arxiv.org/abs/2404.10301`.

[13] Zach Evans, CJ Carr, Josiah Taylor, Scott H Hawley, and Jordi Pons. Fast timing-conditioned latent audio diffusion. In *ICML*, 2024.

[14] Zach Evans, Julian D Parker, CJ Carr, Zack Zukowski, Josiah Taylor, and Jordi Pons. Stable audio open. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2025.

[15] Ruibin Yuan, Hanfeng Lin, Shuyue Guo, Ge Zhang, Jiahao Pan, Yongyi Zang, Haohe Liu, Yiming Liang, Wenye Ma, Xingjian Du, et al. Yue: Scaling open foundation models for long-form music generation. *arXiv:2503.08638*, 2025. URL `https://arxiv.org/abs/2503.08638`.

[16] Zachary Novack, Zach Evans, Zack Zukowski, Josiah Taylor, CJ Carr, Julian Parker, Adnan Al-Sinan, Gian Marco Iodice, Julian McAuley, Taylor Berg-Kirkpatrick, et al. Fast text-to-audio generation with adversarial post-training. *arXiv:2505.08175*, 2025. URL `https://arxiv.org/abs/2505.08175`.

[17] Xun Zhou, Charlie Ruan, Zihe Zhao, Tianqi Chen, and Chris Donahue. Local deployment of large-scale music AI models on commodity hardware. In *ISMIR Late Breaking Demos*, 2024.

[18] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. In *NeurIPS*, 2017.

[19] Haibin Wu, Xuanjun Chen, Yi-Cheng Lin, Kai-wei Chang, Ho-Lam Chung, Alexander H Liu, and Hung-yi Lee. Towards audio language modeling–an overview. *arXiv:2402.13236*, 2024. URL https://arxiv.org/abs/2402.13236.

[20] Yunpeng Li, Kehang Han, Brian McWilliams, Zalan Borsos, and Marco Tagliasacchi. SpectroStream: A versatile neural codec for general audio. *arXiv:2508.05207*, 2025. URL https://arxiv.org/abs/2508.05207.

[21] Dongchao Yang, Jinchuan Tian, Xu Tan, Rongjie Huang, Songxiang Liu, Xuankai Chang, Jiatong Shi, Sheng Zhao, Jiang Bian, Xixin Wu, et al. UniAudio: An audio foundation model toward universal audio generation. *arXiv:2310.00704*, 2023. URL https://arxiv.org/abs/2310.00704.

[22] Alexandre Défossez, Laurent Mazaré, Manu Orsini, Amélie Royer, Patrick Pérez, Hervé Jégou, Edouard Grave, and Neil Zeghidour. Moshi: a speech-text foundation model for real-time dialogue. *arXiv:2410.00037*, 2024. URL https://arxiv.org/abs/2410.00037.

[23] Brian Victor McWilliams, Emanuel René Jacques Orsini, Zalán Borsos, Alexandru Tudor, Damien Vincent, and Marco Tagliasacchi. Efficient generation of multimodal sequences using between-frame and within-frame machine-learned models, Jun 2025. URL https://patentscope.wipo.int/search/en/WO2025128464.

[24] Liwei Lin, Gus Xia, Junyan Jiang, and Yixiao Zhang. Content-based controls for music large language modeling. *arXiv:2310.17162*, 2023. URL https://arxiv.org/abs/2310.17162.

[25] Rodrigo Castellon, Chris Donahue, and Percy Liang. Codified audio language modeling learns useful representations for music information retrieval. In *ISMIR*, 2021.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

[27] 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*, 2021.

[28] Qingqing Huang, Aren Jansen, Joonseok Lee, Ravi Ganti, Judith Yue Li, and Daniel PW Ellis. Mulan: A joint embedding of music audio and natural language. *arXiv:2208.12415*, 2022. URL https://arxiv.org/abs/2208.12415.

[29] Jiahui Yu, Zirui Wang, Vijay Vasudevan, Legg Yeung, Mojtaba Seyedhosseini, and Yonghui Wu. CoCa: Contrastive captioners are image-text foundation models. *TMLR*, 2022.

[30] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

[31] 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*, 2023.

[32] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M Dai, Matthew D Hoffman, Monica Dinculescu, and Douglas Eck. Music Transformer. In *ICLR*, 2018.

[33] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 2024.

[34] Ofir Press, Noah A Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv:2108.12409*, 2021. URL https://arxiv.org/abs/2108.12409.

[35] 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. *JMLR*, 2020.

[36] Adam Roberts, Hyung Won Chung, Gaurav Mishra, Anselm Levskaya, James Bradbury, Daniel Andor, Sharan Narang, Brian Lester, Colin Gaffney, Afroz Mohiuddin, et al. Scaling up models and data with t5x and seqio. *Journal of Machine Learning Research*, 2023.

[37] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021.

[38] Felix Kreuk, Gabriel Synnaeve, Adam Polyak, Uriel Singer, Alexandre Défossez, Jade Copet, Devi Parikh, Yaniv Taigman, and Yossi Adi. Audiogen: Textually guided audio generation, 2023. URL https://arxiv.org/abs/2209.15352.

[39] Guillaume Sanchez, Honglu Fan, Alexander Spangher, Elad Levi, Pawan Sasanka Ammanamanchi, and Stella Biderman. Stay on topic with classifier-free guidance, 2023. URL https://arxiv.org/abs/2306.17806.

[40] Ilaria Manco, Benno Weck, Seungheon Doh, Minz Won, Yixiao Zhang, Dmitry Bogdanov, Yusong Wu, Ke Chen, Philip Tovstogan, Emmanouil Benetos, Elio Quinton, György Fazekas, and Juhan Nam. The Song Describer dataset: a corpus of audio captions for music-and-language evaluation. In *Machine Learning for Audio Workshop at NeurIPS 2023*, 2023.

[41] Aurora Linh Cramer, Ho-Hsiang Wu, Justin Salamon, and Juan Pablo Bello. Look, listen, and learn more: Design choices for deep audio embeddings. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019.

[42] Roger B Dannenberg. An on-line algorithm for real-time accompaniment. In *International Computer Music Conference (ICMC)*, 1984.

[43] Barry Vercoe. The synthetic performer in the context of live performance. In *International Computer Music Conference (ICMC)*, 1984.

[44] Nicola Orio, Serge Lemouton, and Diemo Schwarz. Score following: State of the art and new developments. *New Interfaces for Musical Expression (NIME)*, 2003.

[45] George E Lewis. Too many notes: Computers, complexity and culture in voyager. *Leonardo music journal*, 2000.

[46] Belinda Thom. BoB: an interactive improvisational music companion. In *International Conference on Autonomous Agents*, 2000.

[47] Francois Pachet. The Continuator: Musical interaction with style. *Journal of New Music Research*, 2003.

[48] Zachary Kondak, Mikayla Konst, Carli Lessard, David Siah, and Robert M Keller. Active trading with Impro-Visor. In *International Workshop on Musical Metacreation*, 2016.

[49] Chris Donahue, Ian Simon, and Sander Dieleman. Piano Genie. In *International Conference on Intelligent User Interfaces (IUI)*, 2019.

[50] Nan Jiang, Sheng Jin, Zhiyao Duan, and Changshui Zhang. RL-Duet: Online music accompaniment generation using deep reinforcement learning. In *AAAI*, 2020.

[51] Christodoulos Benetatos, Joseph VanderStel, and Zhiyao Duan. BachDuet: A deep learning system for human-machine counterpoint improvisation. In *New Interfaces for Musical Expression (NIME)*, 2020.

[52] Lancelot Blanchard, Perry Naseck, Eran Egozy, and Joseph A Paradiso. Developing symbiotic virtuosity: AI-augmented musical instruments and their use in live music performances. 2024.

[53] Yusong Wu, Tim Cooijmans, Kyle Kastner, Adam Roberts, Ian Simon, Alexander Scarlatos, Chris Donahue, Cassie Tarakajian, Shayegan Omidshafiei, Aaron Courville, et al. Adaptive accompaniment with ReaLchords. *arXiv:2506.14723*, 2025. URL https://arxiv.org/abs/2506.14723.

[54] Jesse Engel, Lamtharn (Hanoi) Hantrakul, Chenjie Gu, and Adam Roberts. DDSP: Differentiable digital signal processing. In *ICLR*, 2020.

[55] Antoine Caillon and Philippe Esling. RAVE: A variational autoencoder for fast and high-quality neural audio synthesis. *arXiv:2111.05011*, 2021. URL https://arxiv.org/abs/2111.05011.

[56] Sander Dieleman, Aaron Van Den Oord, and Karen Simonyan. The challenge of realistic music generation: modelling raw audio at scale. In *NeurIPS*, 2018.

[57] Zihan Liu, Shuangrui Ding, Zhixiong Zhang, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Yuhang Cao, Dahua Lin, and Jiaqi Wang. SongGen: A single stage auto-regressive transformer for text-to-song generation. *arXiv:2502.13128*, 2025.

[58] Junmin Gong, Sean Zhao, Sen Wang, Shengyuan Xu, and Joe Guo. ACE-Step: A step towards music generation foundation model. *arXiv:2506.00045*, 2025.

[59] Mark Levy, Bruno Di Giorgi, Floris Weers, Angelos Katharopoulos, and Tom Nickson. Controllable music production with diffusion models and guidance gradients. *arXiv:2311.00613*, 2023.

[60] Zachary Novack, Julian McAuley, Taylor Berg-Kirkpatrick, and Nicholas J Bryan. DITTO: Diffusion inference-time t-optimization for music generation. In *ICML*, 2024.

[61] Rithesh Kumar, Prem Seetharaman, Alejandro Luebs, Ishaan Kumar, and Kundan Kumar. High-fidelity audio compression with improved RVQGAN, 2023.

[62] Shih-Lun Wu, Aakash Lahoti, Arjun D Desai, Karan Goel, Chris Donahue, and Albert Gu. Towards codec-LM co-design for neural codec language models. In *Student Research Workshop at the Nations of the Americas Chapter of the Association for Computational Linguistics (NAACL-SRW)*, 2025.

[63] Jingwei Zhao, Gus Xia, and Ye Wang. Beat transformer: Demixed beat and downbeat tracking with dilated self-attention. *arXiv:2209.07140*, 2022. URL https://arxiv.org/abs/2209.07140.

[64] Minseok Kim, Woosung Choi, Jaehwa Chung, Daewon Lee, and Soonyoung Jung. KUIELab-MDX-Net: A two-stream neural network for music demixing. *arXiv:2111.12203*, 2021. URL https://arxiv.org/abs/2111.12203.

[65] Jesse Engel, Matthew Hoffman, and Adam Roberts. Latent constraints: Learning to generate conditionally from unconditional generative models. In *ICLR*, 2018.

A Related Work

Our work on live music models connects to longstanding goals in computer music of live, computer-aided performance. Earlier work centered around score following: tracking live performances against known musical material to enable computer accompaniment [42–44], or systems that could both track input from human musicians and generate new symbolic material using simple probabilistic models [45–48]. More recently, numerous systems have proposed live interaction with generative AI models of symbolic music [49–53], or small models trained on narrow music audio distributions [54, 55]. Here we aim to bridge the gap between the live interaction paradigm and the broad audio generation and control capabilities of large-scale generative AI models of music audio.

Many offline music audio generation models are based off of codec LMs [18, 19]. Codec LMs are theoretically capable of streaming provided they meet two criteria: (1) the language model and codec are *causal* (outputs never depend on future inputs), and (2) they generate with RTF $\geq 1\times$ for some chunk size C (which lower bounds the control delay D). To the best of our knowledge, no existing codec LM music generation model [6, 8, 9, 15, 56–58] satisfies both criteria (most use non-causal codecs). Other music generation systems [7, 11–14] are based off of latent diffusion—many achieve a throughput RTF $\geq 1\times$ but generate in a non-causal fashion. Some "outpainting" methods have been proposed for latent diffusion music models [59, 60] that are related to chunked autoregression—to the best of our knowledge, these approaches have not been explored for live generation.

B Limitations

Both Magenta RT and Lyria RT present known limitations. Since the models operate on two-second chunks, user inputs for the style prompt may take two or more seconds to influence the musical output. Due to the maximum audio context window of ten seconds, the models are also unable to directly reference music that has been output further into the past. While this is sufficient to create melodies, rhythms, and chord progressions, it does not allow to automatically create longer-term song structures.

C Additional Methodological and Training Details

C.1 SpectroStream

Here we adopt the recently proposed SpectroStream codec [20], a full-band multi-channel neural audio codec based on residual vector quantization (RVQ) [27]. Similarly to its predecessor SoundStream [27], SpectroStream is trained using a combination of adversarial and reconstruction losses. Unlike SoundStream, SpectroStream models audio in the time-frequency domain and adopts a delayed fusion mechanism, which together allow for high-fidelity audio. Specifically, SpectroStream operates on full-bandwidth stereo music at high sample rate ($f_s = 48$kHz). Relative to other discrete codecs like the Descript Audio Codec [61], we train SpectroStream with a relatively slow framerate ($f_k = 25Hz$) and deeper residual quantizers ($d_c = 64$), consistent with recommendations from [62]. Using 10-bit codebooks ($|\mathbb{V}_c| = 1024$), this induces an overall bandwidth of 16kbps. With the goal of facilitating streaming generation via an LM, we reduce the bitrate for generative modeling to 4kbps by generating only the first 16 RVQ levels (coarse and medium from Figure 3), inducing a throughput target for live generation of 400 tokens per second.

D Lyria RT and Advanced Controls

Lyria RT shares the same general architecture as Magenta RT, but with additional controls (see comparison in Table 2). Here we detail how those controls are provided as musical descriptors (D.1), how they are steered through self-conditioning and control priors (D.2), and lastly how the style embeddings are further guided through latent constraints (D.3).

D.1 Descriptor-based Conditioning

As seen in Section 4.1, contrastive audio-text embeddings obtained from models such as MusicCoCa and MuLan effectively enable control over the overall style of generated musical signals. They

Model	Access	Style Model	SS RVQ Levels	Refinement Model	Advanced Controls	Latent Constraints
Magenta RT	Open	MusicCoCa	16	✗	✗	✗
Lyria RT	API	MuLan [28]	64	✓	✓	✓

Table 2: Comparison of Magenta RT and Lyria RT model features.

Control	Feature Extractor
Brightness	Log-mel Spectrogram Spectral Centroid and Bandwidth
Density	Onset detection
Key	Chroma weighted average
Tempo	Beat prediction model [63]
Stems On/Off	Stem separation (Bass, Drums, Vocals, Other), threshold on stem loudness

Table 3: List of controls used in training Lyria RT and method of musical descriptor feature extraction.

are not designed, however, to control fine-grained musical attributes. This section investigates the incorporation of additional conditioning features extracted from audio signals using Music Information Retrieval (MIR) methods. A full list of advanced controls for Lyria RT is provided in Table 3.

Temporal conditioning We aim to provide precise control over the tempo of the generated stream, defined using *beats-per-minute* (BPM). Since we do not have access to BPM annotated audio examples, we use an off-the-shelf beat prediction model [63] to annotate our dataset, yielding an estimate for beat positions and an averaged BPM value for the entire track. We start by conditioning our model on the target BPM value rounded to the nearest integer using cross-attention.

Instrumentation, timbre and harmony In addition to style steering using MuLan embeddings, we provide finer grained controls for these features. We use a source separation model [64] to extract the vocals, bass, drums and other stems from our dataset, and use those stems to generate a set of acoustic features to further condition our model. We specifically extract peak loudness, spectral centroid and bandwidth, chromas and transients separately for every stem.

D.2 Self-conditioning

Predicting conditioning tokens Given an acoustic token sequence $\mathbf{x}$ and a conditioning token sequence $\mathbf{c}$, it is common practice to learn the conditional distribution $p(\mathbf{x}|\mathbf{c})$ in a supervised fashion, and expose the conditioning tokens to the user during inference to make the generation controllable. This approach has several drawbacks:

1. User defined conditioning might be out of distribution, especially when conditioning tokens are dependent on each other (e.g., natural temporal evolution of a conditioning signal).
2. Unconditional generation can be achieved through training-time conditioning dropout, however the resulting samples are empirically worse than those from a model trained unconditionally from scratch.

To address these issues, we introduce *self-conditioning* where we aim at learning the joint distribution $p(\mathbf{x}, \mathbf{c}) = p(\mathbf{x}|\mathbf{c})p(\mathbf{c})$. This can be achieved by prefixing the acoustic tokens with the conditioning tokens, and train the model using a causal mask, as seen in Figure 4.

Intuitively, this allows to use the resulting model either unconditionally, by first sampling the conditioning tokens and then sampling from the conditional distribution, or let the user override some or all of the conditioning tokens to gain some control over the generation. While this helps bridge the quality gap with unconditional models, this methods expects the user to provide conditioning tokens aligned with the underlying prior distribution $p(\mathbf{c})$. This is a reasonable assumption for scalar conditions like tempo, but is significantly harder for complex conditions such as stems loudness or chromas.

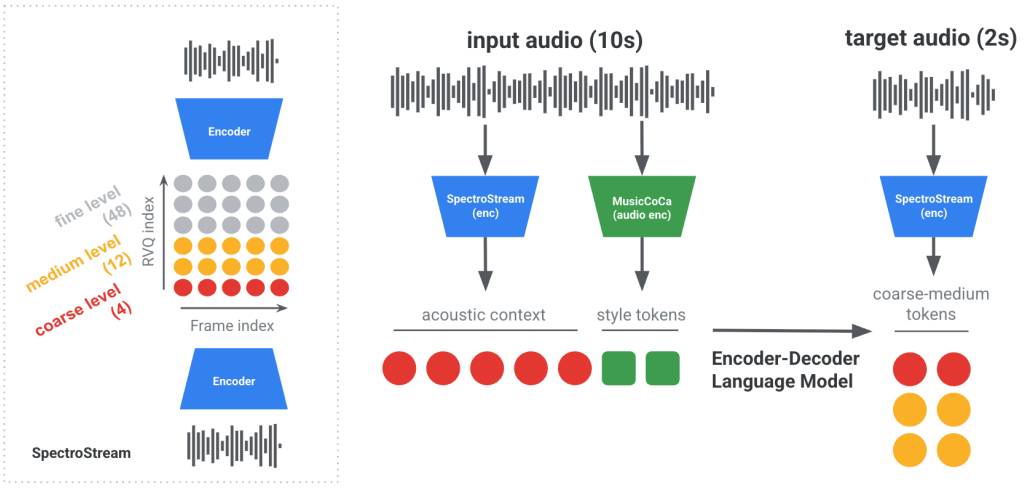

Figure 3: Overall architecture of Magenta RT. Coarse acoustic tokens and quantized style tokens corresponding to 10s of audio context are concatenated and fed to the encoder part of our model. The decoder then predicts coarse and medium acoustic tokens corresponding to the the following 2 seconds.

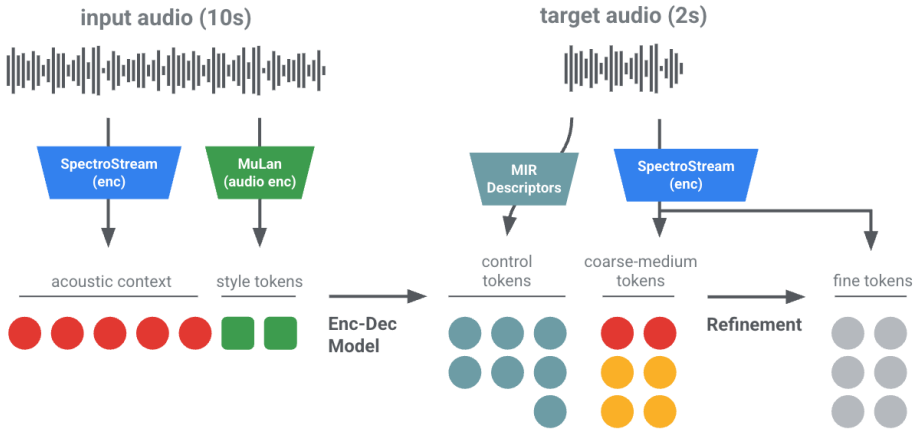

Figure 4: Diagram of Lyria RT training / predicting conditioning tokens. Coarse acoustic tokens and quantized MuLan tokens are concatenated and fed to the encoder part of our language model. Control tokens, including BPM, stem balance, brightness, density and chromas (see Section D.1) are predicted first by the decoder, followed by coarse and medium level acoustic tokens. Finally, a small refinement model predicts the fine-scale acoustic tokens as described in Section D.4.

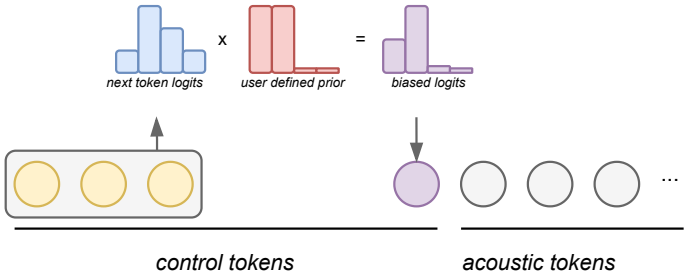

Figure 5: Control priors for self-conditioning. The predicted logits (likelihood) for the control tokens are shifted by a user defined prior dictated by the control values. These are combined to give the final posterior logits that are used for sampling, and steer the model outputs in the direction of the user controls.

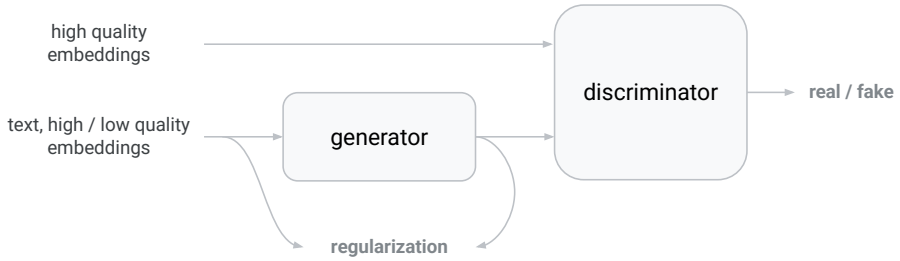

Figure 6: Training the latent constraint model

Control priors We leverage the fact that we already have an estimation of the distribution underlying the conditioning tokens $p(\mathbf{c})$ to implement *soft controls*. Soft controls are implemented through the definition of a prior distribution over the conditioning tokens that is used to steer the sampling process. In practice, we map simple controls to categorical priors that we combine with the next token predicted logits to steer the sampling process, as shown in Figure 5.

D.3 Constraining Style Embeddings

As mentioned in Section 2.2, we train the language models conditioned on style embeddings computed from the raw audio waveform. This is in contrast to the inference setup where we use text based embeddings. While similar, due to the contrastive training of MuLan, the audio and text embedding spaces are not completely overlapping. Indeed, early experiments showed that a two layer MLP is sufficient to classify embeddings as text-based or audio-based with $> 90\%$ accuracy. In practice, we notice a non-negligible drop in audio quality when predicting from text based embeddings compared to audio embeddings, which is likely due to the mismatch between both embedding spaces.

Furthermore, biases exist between specific text genres and audio recording quality. To address these issues, impose a *latent constraint* [65], by building a dataset comprised of MuLan embeddings from three sources (text, low-quality audio, and high-quality audio) and training a small GAN to transform text based embeddings into high quality audio embeddings using an adversarial setup.

Both the generator and discriminator are a 4-layer MLP. The discriminator is trained to separate ground truth high-quality audio embeddings (i.e. *real) embeddings* from the generator outputs (i.e. *fake* embeddings). The generator is trained to optimize the following two objectives: being classified as *real* by the classifier while staying close to the original base embedding. We use the Hinge GAN objective, and use cosine similarity between input and generated embeddings as a regularization strategy.

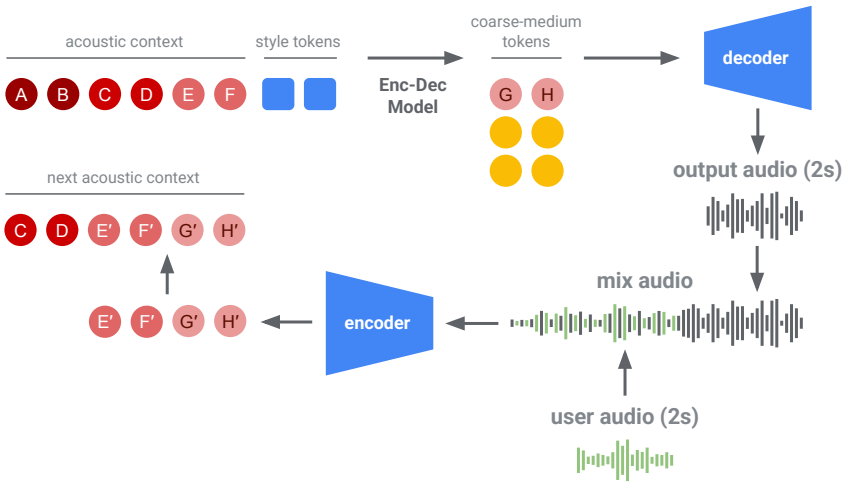

Figure 7: Steering with a live audio stream using *audio injection*. At each inference step, user input audio is mixed with model output audio, and the mix is encoded as coarse SpectroStream tokens. These context tokens are passed as the input for the next inference step, so the model predicts the continuation following its own output *plus* the user's input.

D.4 Refinement Model

As seen in Figure 4, Lyria RT has an additional refinement model to predict 48 additional fine-scale RVQ SpectroStream tokens per a frame. This model is trained separately from the core language model, and is a very small MLP that quickly autoregressively predicts the remaining tokens. This provides a slight bump in audio fidelity at the cost of longer inference. Because Magenta RT is focused towards real-time control on less powerful hardware, we reserve the refinement model for Lyria RT and find it to be a good compromise of audio fidelity and compute efficiency.

E Audio Injection

To allow the user to continuously steer generation via a live audio stream, we propose an audio steering mechanism that we call *audio injection*. At each generation step, we mix the user's input audio with the model's output, tokenize the resulting mix, and feed this as the model's context for the next generation. This is illustrated in Figure 7. Note, user audio is never played back directly. Rather, the model predicts the continuation of a past context that *includes* the user audio. Depending on the specific audio injected and the target style, the model may "choose" to repeat or transform the user audio, or may be influenced by its features (dynamics, melody, harmony).

To better control the influence of the user audio on the model output, we use classifier free guidance [37]. Specifically, we perform inference on both a positive example where the audio context *includes* the mixed-in user input, as well as a negative example where the audio context consists of only the model's output. The final logits used for sampling are calculated as a linear combination of the positive and negative logits, where higher guidance weight (w) pushes towards the positive conditioning: $(1 + w) \cdot \text{Logits}_{\text{pos}} - w \cdot \text{Logits}_{\text{neg}}$

Live Audio Prompt To further increase the effect of the user input, we add a "live" audio prompt that periodically updates based on the MusicCoCa audio embedding of the most recent user input audio. Increasing the weight of this prompt within overall prompt mixture has the effect of steering the model to output music in a style that is consistent with the user audio (e.g., containing guitar, if the user is playing guitar).

One challenge to audio injection is that there is unavoidable latency between the input and output streams. In our testing setup, we observed a delay of several seconds between the input and output. Thus, if the user is performing *along with* the model, we will only have around 7 seconds of input audio ready to mix with the model's most recent 10 seconds of output, to feed as the next context.

We consider two solutions to this issue, which are suited to two different interaction styles: "free" mode, and "looper" mode. In free mode, the user's input audio is mixed directly with the "concurrent" output (i.e. matching what the user heard while they were performing), but no input is mixed into the final stretch of context. In looper mode, we assume the music follows a "looping" structure at a known tempo, and mix input from the *previous* loop into the current context, filling the entire context window with input audio. We discuss these approaches and their tradeoffs below.

"Free" Mode We mix the user's input in at its original timing (aligning with what the user heard when they performed it), and mix in silence for that portion of the context for which the streamer has not yet received the inputs. Since the inputs are mixed at their logical location, this control mechanism is intuitive, and the streams can be mixed without knowledge of the tempo or song structure. However, one disadvantage is the user's contribution will suddenly drop to silence for the final portion of the model's context. This "cutting out" can weaken the effect of the conditioning, as the most recent context tends to have the largest effect on generation. It may also lead to unwanted artifacts, since the user's audio may be clipped to silence mid-phrase.[4]

"Looper" Mode For music with fixed tempo and a relatively short "looping" structure (e.g., a repeating 4-bar chord progression), we can overcome the latency issue by mixing in user audio from the *previous* loop, as opposed to the current one. This interaction is similar to a "looper" pedal (where audio is looped to build layered composition); however instead of playing the past audio back verbatim, we mix it into the model's context window, where it may influence the model's continuation. Compared to "free" mode, this has the advantage of allowing us to fill the *entire* context window with user audio, which we find increases the likelihood of the model being influenced by that audio.

However there are several downsides to "looper" mode. First, the control is less intuitive than "free" mode, as the effect of the user input is delayed by one loop. This requires the user to plan ahead, and also limits what musical forms can be expressed. Second, the user needs to specify the loop length (e.g., 8 beats at 120 BPM), and we need to guarantee the model output actually respects this chosen tempo and loop length. This can be achieved by tempo conditioning (Section D.1), or by adjusting the loop length in real time to match the model output. However if the model ever drifts from the set tempo, the mixed user input will be misaligned with the model output, which can lead to unexpected results.

Throughout development, we prototyped audio injection with musicians of different backgrounds, including instrumentalists (guitar, piano, drums), a producer, and a live electronic DJ. User responses were varied, with some finding it inspiring, and others finding it too unpredictable.

[4]We can soften these artifacts by fading the input to silence. However, the fade out may still be unexpected, and doesn't correspond to the user's true performance.

F Prompt Transitions

Prompts used for the "Prompt Transition" eval in Section 3.2.2. Transitions are linear interpolations of text embeddings every 10 seconds over 60 seconds (i.e., $[0.0, 1.0]$, $[0.2, 0.8]$, $[0.4, 0.6]$, $[0.6, 0.4]$, $[0.8, 0.2]$, $[1.0, 0.0]$).

Accordion → Ambient
Accordion → Dirty Synths
Accordion → Minimal Techno
Afrobeat → Synthpop
Alternative Country → Dirty Synths
Alto Saxophone → Dulcimer
Ambient → Gypsy Jazz
American Folk → Afrobeat
Balalaika Ensemble → Fuzz Guitar
Banjo → Jamaican Dub
Baroque → Djembe
Baroque → West Coast Hip Hop
Bass Clarinet → Orchestral Score
Bassoon → Classic Rock
Blues Rock → Balalaika Ensemble
Blues Rock → Glitch Hop
Blues Rock → Marching Band
Bongos → Lute
Bongos → Viola Ensemble
Bouzouki → Indie Folk
Breakbeat → Bongos
Breakbeat → Synthpop
Cavaquinho → Flamenco Guitar
Cello → Classic Rock
Charango → Guitar
Clavichord → Precision Bass
Congo Drums → Vaporwave
Contemporary R&B → Harpsichord
Contemporary R&B → Pop Punk
Country → Breakbeat
Country → Renaissance Music
Delta Blues → Smooth Pianos
Didgeridoo → Charango
Didgeridoo → Kalimba
Dirty Synths → Cavaquinho
Dirty Synths → Ukulele
Disco Funk → Hard Rock
Djembe → Dirty Synths
Djembe → Psychedelic
Doo Wop → Bossa Nova
Doo Wop → Industrial Rock
Doo Wop → Tango
Drum & Bass → Gypsy Jazz
Drum & Bass → Harp
Drum & Bass → Trance
Electro Swing → Lute
Electro Swing → Pipa
Erhu → Mandolin
Fiddle → Hard Rock
Fiddle → Surf Rock
Flamenco Guitar → Bass Clarinet
Funk Metal → Pipa
Funky → Koto
Fuzz Guitar → Soprano Saxophone
Fuzz Guitar → Synth Pads
Fuzz Guitar → TR-909 Drum Machine
Fuzz Guitar → Trance
Glitch Hop → Synthpop
Glockenspiel → Clavichord
Hang Drum → Jamaican Dub
Hang Drum → Steel Drum
Hard Bop Jazz → Tango
Hard Rock → Balalaika Ensemble
Hard Rock → Hurdy-gurdy
Harp → R&B (Rhythm and Blues)
Harpsichord → Congo Drums
Harpsichord → Koto
Harpsichord → Trance
Heavy Metal → Chiptune
Heavy Metal → Jamaican Dub
Hurdy-gurdy → Cavaquinho
Hurdy-gurdy → Classic Rock
Hurdy-gurdy → Gypsy Jazz
Hyperpop → Bongos
Indian Classical → Tuba
Indie Pop → Ska
Industrial Rock → Accordion
K-Pop → Soprano Saxophone
Klezmer → Glockenspiel
Klezmer → Hang Drum
Klezmer → Tuba
Latin Jazz → Erhu
LinnDrum → Synth Pads
Lo-Fi Hip Hop → Bagpipes
Lute → Balalaika Ensemble
Lyre → Latin Jazz
Lyre → Techno
Mandolin → Synth Pads
Marching Band → Chamber Music
Mbira → Hard Bop Jazz
Neo-Soul → Marimba
Neo-Soul → Psychedelic Rock
Orchestral Score → Mellotron
Piano Ballad → Garage Rock
Piano Ballad → LinnDrum
Pipa → Chiptune
Pop Punk → Ambient
Pop Punk → Baroque
Pop Punk → Hurdy-gurdy
Post-Punk → Chillout
Precision Bass → Accordion
Progressive House → Irish Folk

Progressive House → Mariachi
Psychedelic Rock → Sitar
Renaissance Music → Warm Acoustic Guitar
Sarod → Funk Drums
Sitar → Marimba
Ska → Fiddle
Steel Drum → Post Rock
Surf Rock → Gypsy Jazz
Thrash Metal → Tabla
Vaporwave → Ragtime Piano
Warm Acoustic Guitar → Klezmer
West Coast Hip Hop → Indian Classical
Woodwinds → Psychedelic Rock
Psychedelic → Moombahton
R&B (Rhythm and Blues) → Mariachi
Sarod → Afrobeat
Sarod → Smooth Pianos
Ska → Dubstep
Smooth Pianos → Garage Rock
Surf Rock → Fiddle
Synth Pads → Mellotron
Trance → Djembe
Warm Acoustic Guitar → Indie Electronic
Warm Acoustic Guitar → LinnDrum
Woodwinds → Accordion
Zither → Dreamy