ImprovMusicGen

Fine-tuning MusicGen for real-time and role-aware improvisation

Kevin Bretz ( )

July 5, 2026

Abstract

Music generation has seen significant advancements since the advent of Transformer-based architectures. However, the specific task of "improvisational" generation—where a model must generate a coherent musical continuation that complements an existing multi-track context—remains a challenging frontier. It would open the door to real-time AI accompaniment models for musical practice or even performance, and could be further fine-tuned using Reinforcement Learning.

This report presents **ImprovMusicGen**, a system designed to generate target instrumental stems conditioned on a musical and auto-regressive context. Leveraging the pre-trained **MusicGen** architecture and the **Slakh** dataset, we employ **Low-Rank Adaptation (LoRA)** to efficiently fine-tune the model. We demonstrate that a LoRA applied to the MusicGen small model can successfully generate a coherent musical improvisation, particularly when targeting an output of drums. However, it lacks consistency, being highly sensitive to the input, and tending to diverge, or "stray" noticeably after multiple auto-regressive steps. We attribute this to the small dataset size, as well as the complexity of the task and how different it is to what MusicGen was trained to do, but acknowledge room for improvement in our implementation.

1 Introduction

Standard music generation models often focus on unconditional generation or text-to-music tasks. While impressive, these approaches do not address the interactive nature of music creation, where musicians listen and react to one another.

This project addresses the problem of **Real-time Conditional Stem Generation**. Given a set of active instruments (the "Context"), as well as the previous output of the target instrument, the goal is to generate an ideally seamless continuation of the input music, or at the very least to generate a complementary stem that respects the target role and follows the input audio's musicality and rhythm. Since the base MusicGen small model can already be deployed in (pseudo-)real-time but is musically boring, a LoRA with the correct training should be equally real-time feasible.

We utilize **MusicGen** [1], a state-of-the-art auto-regressive Transformer model. To adapt this general-purpose model to our specific "jamming" task without the prohibitive cost of full fine-tuning, and without the risk of the model forgetting it's musical understanding, we utilize **Low-Rank Adaptation (LoRA)** [2].

2 Methodology

2.1 Dataset: Slakh2100

We utilize the **Slakh2100** dataset [3], a standard benchmark for music source separation. Slakh consists of high-quality synthetic audio generated from MIDI files, providing perfectly isolated stems for various instruments (Drums, Bass, Guitar, Piano, etc.).
However, it does not provide us with some crucial information for our task, such as the BPM of each track, or which stem is playing which role during (which segment of) which track, especially important for the lead and harmony roles to be clearly distinguishable. For this, we first deploy a BPM tracker and filter out all tracks under 90 BPM (real-time feasibility threshhold determined during testing), and then we make an attempt at automatically classifying our stems into "Lead" and "Harmony" roles using simple heuristics. The results are documented in our metadata.

For our task, we process the dataset into "Jam" pairs:

- **Context:** A mix of $N - 1$ stems from a track.
- **Target:** The remaining stem that the model must generate.

The audio is further augmented and then tokenized using the **EnCodec** neural audio codec, converting continuous waveforms into discrete codebook indices that the Transformer can predict.

2.2 Data Augmentation

For our model to have generalizable improvisation skills it is crucial for us to augment our already quite small dataset as much as possible. First of all, for every single training step we take a random segment of a track as context, and include a random amount of stems in the input mix. Next, we make sure to randomly pitch-shift the context as well as the target stem by +/- 5 semitones. Finally, to effectively mimic and hopefully combat the error accumulation in auto-regressive sequences, we introduce a random token corruption rate, which should closer resemble the actual error than just adding regular noise.

2.3 Model Architecture: MusicGen + LoRA

MusicGen is a language model over quantized audio tokens. It utilizes a residual vector quantization (RVQ) strategy to model multiple streams of codebooks.

To fine-tune this model efficiently, we inject trainable rank decomposition matrices into the Transformer's attention layers (LoRA). This allows us to freeze the vast majority of the pre-trained weights (approx. 300M parameters for MusicGen-Small) and only update a small fraction (the adapters), significantly reducing memory requirements and training time.

2.4 Training Objective

The model is trained using a Cross-Entropy loss. The input sequence consists of the tokenized **Context** audio followed by the **Target** audio. The model is tasked with auto-regressively predicting the next token of the Target sequence given the Context and previous Target tokens.

2.5 Per-Beat Generation

We define the length of the generated output to be exactly one beat, which varies based on the BPM of the track. However, as the dataset is midi-generated the beats are already correctly temporally aligned to the start of the audio. If we know the BPM of a track, we know exactly where in the audio the individual beats will be, and we can always pass the entire beat from start to finish to the model. Because of this, the model should be much better at recognizing patterns in beats. MusicGen inference duration is dominated by its output length, so we want to keep this as short as possible for the model to stay real-time feasible.
On the other hand, inference time is impacted much less by the input length. Because of this, and because musical structure and coherence are strongly dependent on the musical history, we propose to pass a total of 20 beats of musical context, as input to the model. With 5 bars of musical context, the model should hopefully be encouraged to go beyond just outputting generic 4-bar loops and hopefully create long-term progressions, while still learning that the immediate history contains the most important information for the next step. With an output length of one beat, this remains real-time feasible on our hardware down to a BPM of at lest 90.

3 Experiments

3.1 Experimental Setup

Training was conducted on a high-performance computing (HPC) cluster with the following specifications:

- **Hardware:** Single NVIDIA A100 GPU (80GB VRAM).
- **Base Model:** `facebook/musicgen-small`.
- **Optimization:** AdamW optimizer with a learning rate of 2.0×10^{-7} and a cosine annealing schedule.
- **Batch Size:** 32.
- **Context/Target Length:** 20 beats each.

3.2 Evaluation Protocol

The model is evaluated on a held-out validation set from Slakh. For each validation sample, we generate three outputs:

1. **Context:** The input audio fed to the model (the "backing track").
2. **Target:** The model's generated improvisation.
3. **Mix:** The Context and Target merged into one audio file to assess overall coherence.

Crucially, during evaluation we only generate one target beat at a time, but do this auto-regressively 20 times in a row, whereas during training we process all 20 beats in parallel. This speeds up training significantly and is made possible because we have the "ground truth" stem and can therefore pass this

as the auto-regressive input, which is a method known as "Teacher Forcing". A downside is that the model may overfit to this "perfect" input, so that when during actual evaluation a token containing errors gets passed, the model doesn't know how to deal with it and a cascading chain of errors occurs. We attempt to counteract this and improve our models generalization capabilities with our previously explained random token corruption.

4 Results

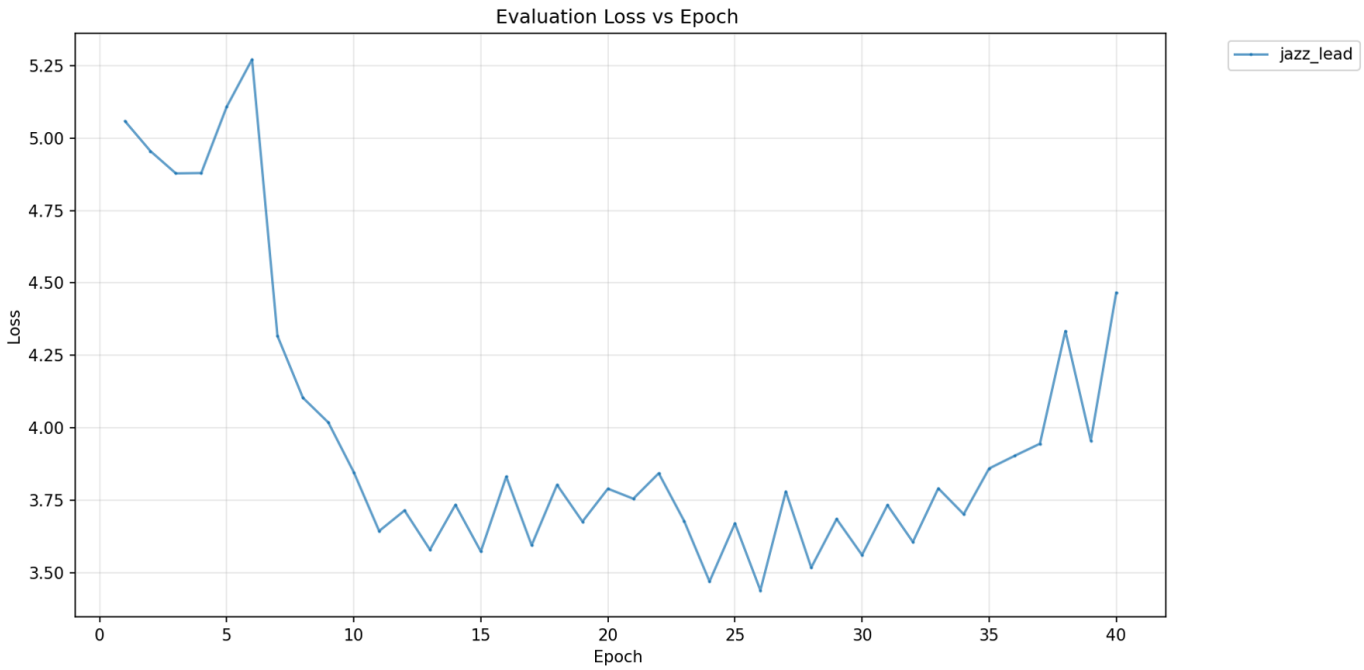

Figure 1: Training progression for a model specialized in playing the lead. Progression of Eval Loss over the course of the training epochs. Sharp drop over first 11 epochs from 5.25 to 3.7. Minimum of 3.45 reached at epoch 26, after which model apparently begins to overfit.

The training progression graph presented in 1 shows expected behavior. The model has a moment around epoch 6 where it learns how to lower the training loss while simultaneously improving the evaluation loss, after which it slowly reaches a minimum at epoch 26. After this, the model appears to be overfitting to the training data, as the evaluation loss slowly starts to creep back up again. Following this rational, our best epoch should be epoch 26.

However, when listening back to the generated audio on evaluation, it becomes apparent that the model did not learn nearly as well as 1 would have you believe. Indeed, these losses are computed as the cross-entropy (similarity) between the generated output and it's "ground-truth" stem, which are given in the form of EnCodec codebooks and do a bad job of representing the change in perceptual quality. When listening back to the output from models produced with our training pipeline, one can often hear the model suppressing all sounds and creating silence. Sometimes the model seems like it is trying to create sounds that are similar to the target but loses all musicality, and other times the

exact opposite can be observed, where the model completely switches up the sounds but with respect to the context's key and rhythm.

5 Discussion and Future Work

Although the models we produced did exhibit promising behavior, they don't come close to what we set out to achieve. To finish off, we would like to reflect on the evolution of this project, as well as critique our approach and make some proposals for future work.

5.1 Project Evolution

Here are some notable changes the project underwent over the course of its run:

1. **Removed Genres** After switching to the Slakh dataset due to the huge benefit of providing isolated stems, we had to make a compromise because the Slakh dataset is quite small by removing genre-specific training.
2. **LoRA, to Full Fine-Tuning, back to LoRA** Initial tests with LoRA were performing poorly, so we ran some tests of standard full fine-tuning instead. As expected, the model was extremely volatile, but we did have a stroke of luck when we evaluated a checkpoint that was taken during the early scheduler warm-up phase, so while the learning rate was still very low, and the results were far better than anything before. However, full fine-tuning is not the solution as the model is far too prone to catastrophic forgetting, so we reverted back to using a LoRA but with a greatly reduced learning rate, where we were able to reproduce the previous minor success.

Ideally, all of these idea's should be revisited once the core problems of data sparsity and training setup have been solved.

5.2 Future Work

1. **MusicGen distribution** We strongly assume that one of the main challenges we faced when attempting to fine-tune MusicGen is that we are trying to train it to do something it was fundamentally trained not to do during pre-training. Specifically, we think that MusicGen "wants" to output a full mix. It hasn't been trained to output only a certain role, it has been trained to receive a full mix and output a full mix. When we perform our evaluation using just the base MusicGen small model, we find that after multiple consecutive auto-regressive steps it starts to reintroduce more instruments into the mix, which seems to confirm our theory.
   The LoRA should technically be able to work around this, but is inconsistent in our experiments. More research should be done to identify if this is a fundamental issue with the MusicGen approach, or if there is a workaround.
2. **Role Ambiguity** Specifically for the "Lead" and "Harmony" roles, we would ideally need a dataset that comes with these already labeled. If the model is being confused by ambiguous training data it will never perform well. To confirm this, drum evaluations using our models are the most consistently good sounding. However, although slightly better than lead and harmony,

bass does not suffer from this ambiguity problem yet also under-performs when compared to drums. More research needs to be done to explain this behaviour.

3. **Training Design** Although we tried to account for as much as we could, the results show that there is still much room for improvement in our approach. We strongly encourage the continuation of this work, as we believe it could open the door for countless new applications, and possibly even more excitingly could be further expanded upon through reinforcement learning.

References

[1] Copet, J., et al. (2023). Simple and Controllable Music Generation. *arXiv preprint arXiv:2306.05284*.

[2] Hu, E. J., et al. (2021). LoRA: Low-Rank Adaptation of Large Language Models. *arXiv preprint arXiv:2106.09685*.

[3] Manilow, E., et al. (2019). Cutting the Slakh: A Dataset for Music Source Separation. *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*.