CS 757

Spring 2025

Assignment 2

Generative Modeling of Music Audio Signals using Birdie

Sean Steinle, Kiya Aminfar

1. Abstract

This report details the design and implementation of a generative system for music audio signals leveraging the Birdie framework, a SSM designed for flexible training objective management. The core technical contribution involves bridging the modality gap between continuous audio signals and discrete token sequences suitable for transformer-based language models. We developed a multi-stage pipeline encompassing audio preprocessing, mel-spectrogram conversion, vector quantization (VQ) via k-means clustering to create an audio codebook, and a specialized tokenizer for bidirectional conversion. This tokenized representation interfaces with a standard transformer architecture trained using Birdie's objective system, primarily focusing on Prefix Language Modeling (PLM). We analyze the technical challenges encountered, particularly concerning the audio-to-token conversion process, integration with Birdie's data handling expectations, and memory management constraints within multi-process environments, especially on Windows platforms. The resulting system demonstrates the feasibility of applying text-centric training frameworks like Birdie to the audio domain, generating audio, yet more efforts is required to obtain competitive results.

2. Introduction

Generative modeling of audio, particularly music, presents significant challenges due to the high dimensionality, temporal complexity, and long-range dependencies inherent in audio waveforms. While transformer architectures have excelled in sequence modeling for text, their direct application to raw audio is computationally intensive. An alternative approach involves representing audio through discrete tokens derived from intermediate representations like spectrograms. This project investigates such an approach, utilizing a transformer model trained within the Birdie framework.

Birdie offers a sophisticated system for managing and dynamically scheduling various unsupervised and self-supervised training objectives (e.g., language modeling, infilling, denoising). However, its native design targets text data. The central technical hurdle addressed in this work is the adaptation of this text-centric framework for audio generation. This required developing a robust pipeline to convert continuous audio signals into a sequence of discrete

tokens and back, enabling the transformer model to learn distributional properties of the audio data through objectives defined on these token sequences. We specifically explore music generation, leveraging Birdie's capabilities to navigate a reinforcement-based learning objective and calculate loss based on that.

3. Technical approach

Our methodology integrates audio signal processing, vector quantization, transformer modeling, and the Birdie training orchestration framework.

3.1 Audio-to-token pipeline

A multi-stage pipeline was designed for bidirectional conversion between raw audio and discrete token sequences:

Audio loading and standardization: Input wav files are loaded and standardized to a fixed sample rate (SAMPLE_RATE = 22050 Hz) and duration (DURATION = 3 seconds). This ensures consistent input dimensions for subsequent processing. Audio is truncated or padded as necessary. Normalization is applied to mitigate amplitude variations.

Mel-spectrogram conversion: The standardized audio waveform is transformed into a mel-spectrogram, a time-frequency representation emphasizing perceptually relevant frequencies. This reduces dimensionality while preserving essential acoustic features. Key parameters include N_FFT = 512, HOP_LENGTH = 32, and N_MELS = 128. The resulting power spectrogram is converted to a decibel scale.

Other hyperparameters were also explored, revealing that the optimal N_fft value equals the cluster count and significantly exceeds the hop length. The investigation also includes N mels.

VQ and codebook creation: Mel-spectrogram frames (vectors of size N_MELS) are treated as feature vectors. K-means clustering (N_CLUSTERS = 512) is employed offline on a representative dataset of spectrogram frames to learn a discrete codebook. Frames are normalized (zero mean, unit variance) prior to clustering. The resulting cluster centroids form the audio codebook.

Tokenization: During runtime, each frame of an input mel-spectrogram is normalized using the pre-computed statistics and assigned the index of the nearest cluster centroid (codebook vector) via the fitted k-means model. This maps the continuous spectrogram to a sequence of discrete integer token IDs (0 to N_CLUSTERS-1).

Special token integration: The vocabulary is augmented with special tokens required by the language model and Birdie objectives (for example we added [PAD], [UNK], <|START|>, <|END|>, as well as specific paradigm tokens like <|PREFIX LM|>). The [PAD] token is specifically mapped to IGNORE_INDEX = -100 for compatibility with loss computation as is used in the Birdie pipeline.

String representation for Birdie: Since Birdie's internal mechanisms operate on text and expects specific data flow, the sequence of integer token IDs (including special tokens) is converted into a string literal format (e.g., '[123, 45, -100, 512, 234, ...]'). This string representation serves as the primary data format passed to Birdie's data handling and objective processing components. A text_grabber_fn is defined to extract this string from data sample dictionaries.

Detokenization and reconstruction: For audio generation, a sequence of token IDs output by the model is processed. First special tokens and invalid IDs are filtered, then remaining audio token IDs are mapped back to their corresponding codebook vectors (cluster centroids). After it, these vectors are de-normalized using the stored mean and standard deviation. The sequence of de-normalized vectors is reshaped into a mel-spectrogram and then the Griffin-Lim algorithm is used to synthesize an audio waveform from the reconstructed mel-spectrogram.

3.2 Transformer model architecture

A standard transformer decoder architecture, adapted for sequence generation, was implemented (SimpleTransformer):

```
class SimpleTransformer(nn.Module):
  def __init__(self, config):
    super().__init__()
    # Vocab size includes audio tokens + special tokens
    self.embedding = nn.Embedding(config['vocab_size'], config['hidden_size'])
    # Standard positional encodings for sequence order
    self.position_embeddings = nn.Embedding(config['sequence_length'], config['hidden_size'])
    self.layers = nn.ModuleList([
      TransformerLayer(...) for _ in range(config['num_layers'])
    ])
    self.layer_norm = nn.LayerNorm(config['hidden_size'])
    self.dropout = nn.Dropout(config['dropout_prob'])
    # Output projection layer
    self.lm_head = nn.Linear(config['hidden_size'], config['vocab_size'], bias=False)
    # Weight tying for parameter efficiency
    self.lm_head.weight = self.embedding.weight
```

Key configuration parameters used:

```
"model_config": {
  "num_layers": 2,      # Shallow model for faster iteration
  "hidden_size": 128,   # Embedding and hidden state dimension
  "num_heads": 4,       # Number of attention heads
  "ffn_hidden_size": 256, # Inner dimension of feed-forward layers
  "dropout_prob": 0.1,  # Regularization
  "vocab_size": 516,    # N_CLUSTERS + num_special_tokens (*varied)
```

```
    "ignore_index": -100,  # For loss calculation
    "sequence_length": 64  # Maximum context length
}
```

Weight tying between the input embedding layer and the output language modeling head (lm_head) was employed to reduce the model's parameter count and potentially improve generalization.

3.3 Birdie integration and training objectives

Birdie orchestrates the training by managing objective application, and dynamic scheduling based on validation performance. The framework utilizes a registry of objective classes. Our configuration focused primarily on prefix language modeling, Infilling and next token prediction (standard objectives for sequence generation)

```
    # { "name": " prefix_language_modeling ", "prob": 0.2, ... }
    # { "name": "infilling", "prob": 0.3, ... },
    # { "name": "next_token_prediction", "prob": 0.2, ... }
```

We configured the system to utilize three key sequence modeling tasks: Prefix Language Modeling, Infilling, and Next Token Prediction (NTP). In the context of our audio tokens, PLM trained the transformer to predict the latter part of a token sequence given an initial prefix, directly aligning with the autoregressive generation task. Infilling required the model to reconstruct masked sections of the token sequence based on the surrounding context, encouraging the learning of local dependencies and reconstruction capabilities beneficial for audio coherence. NTP, the standard next-token prediction task, provided a dense gradient signal at each step, focusing the model on predicting the immediate successor token.

Birdie's framework allowed us to assign probabilities to each objective, determining how frequently each task was presented to the model during training. The data_generator provided the raw token sequences (as string literals derived from audio), and Birdie's objective handlers then applied the specific transformations for each task to create the final batched input and label tensors.

While NTP might lead to lower training loss values early on due to its simpler prediction task, achieving high-quality, structured audio generation often necessitates incorporating objectives like PLM and Infilling, which train the model on more challenging, context-aware tasks crucial for capturing musical structure. The performance and relative effectiveness of each objective are also influenced by the quality of the VQ tokenization and the configured sequence length.

4. Training process

4.1 Data flow

Birdie manages the data pipeline using multiple worker processes (configurable via num_workers). The data_generator function, equipped with the fitted tokenizer and config, is invoked by each Birdie worker. It loads audio files, performs the audio-to-token conversion, generates the string representation, assigns an objective based on the configured probabilities, and yields/returns sample dictionaries. For training, it provides an infinite generator while for validation, a finite list.

Birdie's internal mechanisms consume these samples, apply the objective transformations (using the paradigm_str and objective-specific logic), batch them, and deliver the final tensors (input_ids, label_ids) to the main training loop via get_next_training_sample. PyTorch Accelerate is used to manage the distributed setup, preparing the model, optimizer, and the Birdie orchestrator itself for multi-process execution.

4.2 Training loop

The training loop iterates for a fixed number of steps (num_steps), fetching batches from Birdie, performing forward/backward passes, and updating model parameters.

It was found out it needs many-many-many steps to get to some results.

Loss is computed using CrossEntropyLoss, ignoring the [PAD] token index (-100). Validation is performed periodically using run_validation, which fetches batches via measure_validation_losses() and computes loss without gradient updates.

5. Generation process

Audio generation is performed autoregressively using the trained SimpleTransformer model and the SpectrogramTokenizer.

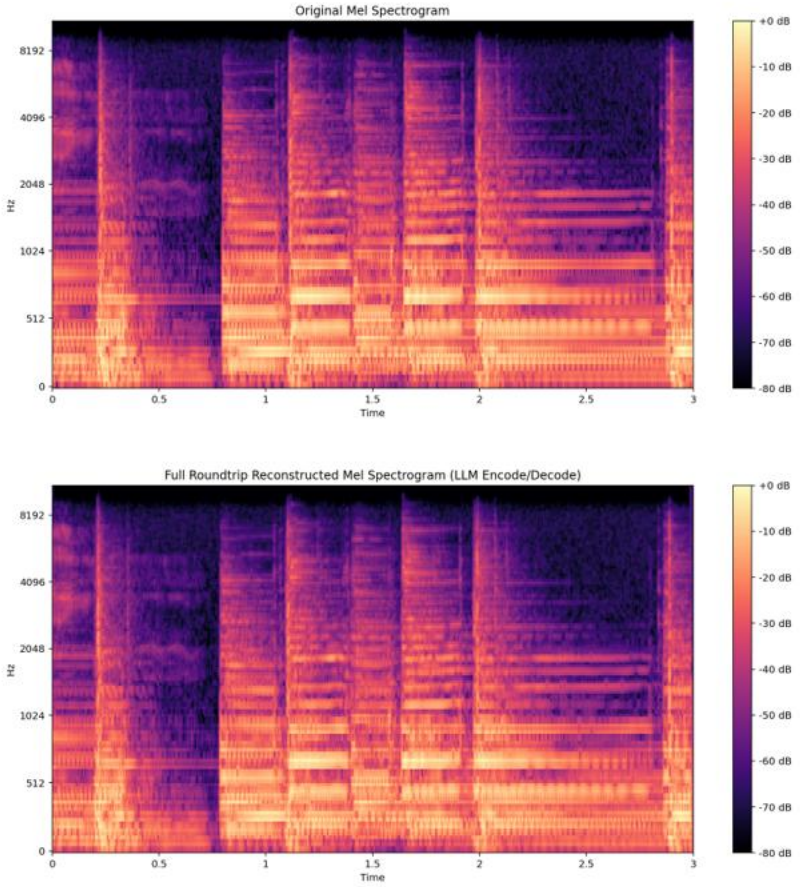
The process starts with an initial sequence (e.g., [PAD, PAD, <|START|>] or tokens from a seed audio clip) represented as initial_text. The model predicts subsequent tokens one by one. Sampling strategies like temperature scaling and top-k filtering control the randomness and quality of the generated sequence. The final sequence of token IDs is decoded back into an audio waveform using the tokenizer's decode method.

6. Results and discussion

Training dynamics: The model demonstrated stable training convergence, with the cross-entropy loss decreasing from an initial value around 6.25 (close to -log(1/516)) to approximately 5.97 over

the configured training steps after 10 steps. Validation loss exhibited a similar trend, suggesting the model was learning meaningful patterns in the audio token sequences rather than simply overfitting. The PLM objective effectively guided the model to capture sequential dependencies.

Spectrogram reconstruction: Visual inspection and quantitative analysis of the spectrograms reconstructed from the VQ codebook indicated acceptable fidelity. While VQ inevitably introduces quantization error, the use of 512 clusters provided sufficient granularity to preserve major spectral structures, although some high-frequency or transient details might be smoothed.



Generated audio quality: Qualitative analysis of generated audio samples revealed segments exhibiting characteristics consistent with noise.

7. Technical Challenges and Solutions

Modality gap: The primary challenge was interfacing the continuous audio domain with the discrete, text-oriented Birdie framework.

Solution: The multi-stage audio-to-token pipeline, particularly the VQ step for discretization and the string-literal representation for Birdie compatibility, successfully bridged this gap. The text_grabber_fn was crucial for extracting this string representation.

Memory management and multi processing issues: Multi-process data loading within Birdie, combined with PyTorch's memory allocation, frequently triggered errors related to insufficient paging file size (DLL load failed... The paging file is too small...) on Windows.

Solution: A combination of strategies was employed: (1) Manually increasing the Windows virtual memory (paging file) size. (2) Implementing explicit tensor deletion (del tensor) followed by gc.collect() within the training and validation loops, although Python's garbage collection guarantees are limited. (3) Utilizing torch.cuda.empty_cache() periodically, though its effectiveness varies. (4) Limiting thread parallelism in numerical libraries (OMP_NUM_THREADS=1, MKL_NUM_THREADS=1) to reduce contention in multi-process settings. Pre-tokenizing the entire dataset and saving tokens could be an alternative but was not implemented here due to the focus on integrating the full pipeline with Birdie.

Data generator design: Ensuring the data_generator provided data in the exact format Birdie expected (infinite generator for train, list for validation, correct dictionary structure, string-literal text field) required careful implementation and debugging.

Solution: Adhering strictly to Birdie's expected function signature and return types, and implementing robust error handling within the generator to skip problematic audio files. Attaching config and tokenizer to the generator function object provided necessary context within Birdie's worker processes.

In addition, continuous debugging (print) shed light on the path forward.

8. Conclusion and future work

This project demonstrated a method for generative audio modeling by adapting the text-focused Birdie framework through a VQ-based audio-to-token pipeline. The system, utilizing a transformer architecture trained primarily with Prefix Language Modeling, generated coherent audio segments reflecting characteristics of the training data. Key technical challenges related to the modality gap and memory management were addressed through specific pipeline design choices and mitigation strategies.

The quality of generated audio is inherently limited by the fidelity of the VQ representation. The Griffin-Lim algorithm for phase reconstruction can introduce artifacts. The model size and training duration were limited, restricting the complexity of learned patterns.