CARNEGIE MELLON UNIVERSITY

MASTER'S THESIS

GrooveIQ: Sparse Representations for Controllable Drum Pattern Generation

---

Author:
Purusottam Samal

Thesis Committee:
Dr. Thomas Sullivan
Dr. Roger Dannenberg
Riccardo Schulz
-

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in

Music and Technology

August 4, 2025

Abstract

GrooveIQ: Sparse Representations for Controllable Drum Pattern Generation

by Purusottam Samal

Drum machines and digital audio workstations (DAWs) offer vast expressive potential, yet authoring natural, expressive drum patterns remains challenging, especially for non-drummers. Crafting compelling grooves often requires fine-grained control over velocity, timing, and articulation across multiple instruments, a task that typically demands both technical fluency and musical training. In contrast, rhythm is a universally intuitive facet of music: even untrained users frequently engage by tapping along to a beat. This observation motivates the design of drum generation systems that translate sparse, instinctive input, such as simple button presses, into rich, human-like drum patterns.

This thesis explores the problem of controllable drum sequence generation from sparse two-hit input sequences. First, we introduce a large-scale symbolic drum performance dataset designed to support expressive generation research. Next, we propose GrooveIQ, a novel autoregressive Transformer-based Variational Autoencoder (VAE) architecture that jointly learns sparse 2-hit control sequences and learns to map those control sequences to full nine-instrument drum performances. We investigate two complementary approaches to learning these sparse control representations: (1) a purely representation learning approach, where the model jointly learns a sequence of latent vectors and a sparse button sequence that minimize reconstruction loss, with sparsity encouraged via L1/group sparsity constraints; and (2) a self-supervised approach, where heuristic button sequences derived from the original drum patterns serve as inductive anchors for learning interpretable control representations.

Our findings demonstrate that the learned control sequences can act as intuitive, low-dimensional handles for expressive rhythm generation, potentially enabling musically aligned performance from minimal input.

Acknowledgments

The list of people I have to thank is long, and in no particular order.

I am deeply grateful to Professor Bhiksha Raj for the remarkable faith he placed in me—both as a TA for his Introduction to Deep Learning course and as a research assistant in the Machine Learning for Signal Processing Group. His intuitive teaching style and rigorous methodology have left a lasting imprint on my thinking and provided the theoretical foundation that made this thesis possible. I am especially thankful for his continued support in allowing me to access compute resources from his lab, even after my formal affiliation ended.

I would like to sincerely thank Professor Tom Sullivan and Professor Roger Dannenberg for their time, patience, and openness. Their willingness to engage with my musings—and my drumbeats—provided invaluable feedback, sharp insight, and a set of inductive biases that helped untangle conceptual knots and shaped much of the work presented in this thesis.

I am especially grateful to Professor Riccardo Schulz for stepping in at a time when the future of my thesis was most uncertain. His generosity and support at that critical moment will always stay with me.

Finally, I wish to express my heartfelt appreciation to my parents, my fiancée, and her parents, whose steadfast belief in me has given me the strength to weather some of the most difficult times in my life. Their support means everything.

Thank you all.

Contents

List of Figures

List of Tables

To my parents, Biswaranjan and Zarina Samal, for their unwavering love, strength, and sacrifices, and to my fiancée, Caitlin Valenzuela, for her endless support, patience, and belief in me…

Chapter 1

Introduction

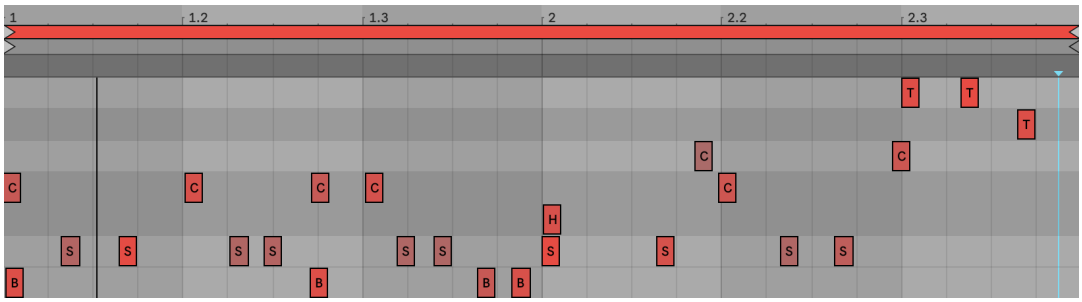

FIGURE 1.1: Piano roll visualization of a two-bar human-performed funk fill. Note the microtiming deviations, dynamic velocity changes, and intricate layering of drum voices—all of which contribute to its expressive character and are difficult to replicate through manual programming.

Drum machines and digital audio workstations (DAWs) have greatly democratized music production, enabling users to craft rhythmically rich and expressive compositions. Yet, replicating the nuance and fluidity of human drumming remains a formidable task. As shown in Figure 1.1, even a brief two-bar funk fill showcases intricate dynamic variation, subtle timing shifts, and tight coordination across multiple drum voices, all of which contribute to its expressive feel. Expert producers often spend years developing the skill to manually sculpt such patterns through meticulous editing of velocity, timing, and articulation. For novices or non-drummers, this process can be daunting, often leading them to rely on pre-made loop libraries or rigid, quantized sequences that fall short of capturing realistic groove and variation.

However, rhythm is one of the most universally intuitive elements of music, shared across cultures and understood even by those without formal training [1]. Untrained listeners often instinctively tap along to complex rhythmic structures, suggesting that expressive rhythmic intent can be conveyed through minimal, intuitive input. This thesis is motivated by a central question: Can we design systems that translate sparse, low-dimensional rhythmic input—such as two-button sequences—into rich, expressive drum patterns that sound natural, human, and musically coherent? If successful, such systems could significantly lower the barrier to rhythm composition by removing the need for precise timing or multi-limb coordination, empowering users to shape expressive grooves through simple and instinctive interactions.

The field of symbolic music generation has advanced considerably with the rise of deep generative models. Sequence-to-sequence Variational Autoencoders (VAEs),

using recurrent networks or Transformers, have shown strong capabilities in modeling expressive musical performances. However, most existing approaches focus primarily on improving generation quality, with control mechanisms largely limited to disentangling specific dimensions of the latent space. There is limited exploration of interpretable, sparse control signals that can directly steer generation. Few methods attempt to jointly learn such sparse controls alongside the generative model, and even fewer examine how these controls might align with semantically meaningful components of the input—such as kick and snare patterns within a full drum sequence.

1.1 Key Contributions

This thesis makes the following contributions to deepen our understanding of how minimal, low-dimensional rhythmic input can be harnessed to guide high-dimensional symbolic music generation in expressive and musically coherent ways:

- **Dataset:** We introduce the GrooveIQ MIDI Dataset—a large-scale, open-source symbolic drum corpus tailored for research in expressive drum patten generation.
- **Model:** We propose GrooveIQ, a lightweight, autoregressive Transformer-VAE designed for real-time generation from sparse controls. The decoder conditions on a latent vector sequence $z_{1:T}$, current and past learned control inputs, and prior outputs. During training, a bidirectional encoder infers both $z_{1:T}$ and optionally the optimal sparse control sequence from the full drum sequence. At inference time, the encoder is discarded, $z_{1:T}$ is sampled from a learned prior distribution, and user-provided control sequences steer generation.
- **Sparse Control Learning:** We explore two paradigms for learning sparse control:
  - Unsupervised representation learning with sparsity constraints.
  - Self-supervised learning via alignment with heuristically derived controls.

  We evaluate both in terms of fidelity, interpretability, and controllability.
- **Application:** We demonstrate that expressive and musically coherent drum patterns can be produced from sparse two-hit control inputs, making rhythm composition more accessible to non-experts.

1.2 Thesis Organization

The remainder of this thesis is structured as follows:

- **Chapter 2 – Related Work:** Reviews literature on symbolic music generation, drum synthesis, and low-dimensional control.
- **Chapter 3 – Dataset:** Describes the creation and structure of the GrooveIQ dataset.
- **Chapter 4 – Methodology:** Details the architecture and training objectives for sparse control learning.

- **Chapter 5 – Experiments and Results:** Presents evaluations, ablations, quantitative and qualitative analyses.
- **Chapter 6 – Conclusion and Future Work:** Summarizes findings and outlines avenues for further research.

Chapter 2

Related Work

Research on symbolic drum pattern generation spans rule-based systems, statistical models, traditional machine learning approaches, and modern deep learning techniques. This section reviews the evolution of these approaches, with an emphasis on methods that generate expressive, human-like drum sequences. We focus in particular on recent deep learning models for symbolic music generation, explore how controllability is introduced in generative systems, and survey relevant drum-specific datasets and evaluation frameworks. Notable contributions such as PianoGenie, GrooVAE, Tap2Drum, MusicVAE, Pop Music Transformer, and Notochord are discussed in detail.

2.1 Rule-based and Statistical Approaches

Early approaches to drum pattern generation relied on explicit rules or patterns defined by musicians and simple statistical models. Rule-based systems typically used pre-programmed drum beats or groove templates (fixed timing and velocity offsets derived from real performances) to add human-like feel to computer-generated drums. For example, many Digital Audio Workstations include "humanization" functions that randomly jitter note timings and dynamics (often by adding Gaussian noise) to mimic a human drummer. However, such naive randomization has minimal effect on listener preferences [2], highlighting the limits of purely rule-based humanization.

Statistical approaches like Markov chains and probabilistic models have also been explored. Markov Drum systems model drum sequences as Markov processes, chaining drum hits based on transition probabilities learned from examples. One interactive system combined Markov chains with user studies on rhythmic similarity, allowing real-time control over pattern complexity (e.g. density) while generating drum beats [3]. These statistical models can capture style to an extent, but often lack the expressive nuance of human drumming.

2.2 Traditional Machine Learning Methods

Before the deep learning era, researchers applied various machine learning techniques to generate more expressive drum performances. Wright and Berdahl [4], for instance, used k-Nearest Neighbors and linear regression to learn micro-timing deviations in Brazilian drumming. Their system could adjust the timing of quantized drum beats by referencing similar phrases in a small dataset, producing a more natural "swing" or groove. Similarly, Echo State Networks (a type of reservoir computing RNN) were used by Tidemann and Demiris [5] to stochastically generate drum rhythms after learning from human drum patterns. These early ML models were

typically trained on limited data (a handful of drummers or genres) and evaluated via listening tests. While not as powerful as later deep models, they did outperform simple heuristics like groove templates in producing convincing variations. Listeners generally found the machine-learned outputs more realistic than those created by fixed rule-based perturbations [4]. These efforts demonstrated the promise of data-driven drum generation, even though the models were relatively simple by modern standards.

2.3 Deep Learning Approaches

With the rise of deep learning, generative models began to yield much more realistic and stylistically rich drum patterns and music sequences. Two broad paradigms emerged for symbolic music representation: piano-roll (image-like) and event sequence (language-like) representations [6]. Piano-roll methods treat time–pitch grids like images; for example, MuseGAN [7] and MidiNet [8] used convolutional and generative adversarial networks on multi-track piano-rolls (including a drum track) to produce multi-instrument music. Such image-based models naturally learn rhythmic structure since beats align with matrix columns. On the other hand, sequence models represent music as a stream of events (notes with timings) and draw on NLP techniques. The Music Transformer [9] treated MIDI events like language tokens and achieved impressive generation of classical piano with expressive timing. However, using a basic MIDI-like event encoding (with Time-Shift tokens for timing) posed challenges for rhythmic coherence – the model sometimes struggled to learn a stable beat, causing tempo drift.

2.3.1 Recurrent Neural Networks and Autoencoders

Earlier deep models for symbolic music often employed recurrent neural networks (RNNs) [10]. For example, Magenta's Drums RNN (an LSTM-based language model) learned to continue drum sequences in various styles, although without explicit learning of human timing nuance. Piano Genie [11] demonstrated an interactive RNN autoencoder: it compresses full piano performances into a discrete 8-button space, enabling a user with eight buttons to improvise a plausible piano melody in real time. Piano Genie's encoder-decoder RNN learns a mapping that translates simple trigger inputs into complex musical sequences, effectively allowing novices to create music through a constrained interface. This idea of an intelligent performance interface illustrates how RNN-based models can enforce musical validity while offering user control. Another example, PerformanceRNN [12], modeled expressive solo performances (e.g. piano) by outputting note events along with dynamics and timing, demonstrating that an LSTM can capture expressive timing when trained on performance data. These RNN approaches laid the groundwork, but they often faced difficulties with longer-term structure and sometimes collapsed to repetitive outputs for longer sequences.

2.3.2 Variational Autoencoders

To better handle musical structure and provide a latent space for interpolation, researchers turned to Variational Autoencoders (VAEs) [13]. Roberts et al. [14] introduced MusicVAE, a hierarchical latent-variable model for music sequences. MusicVAE uses a two-level decoder (e.g. for measures and notes) to overcome the "posterior collapse" issue of sequence VAEs and capture long-term structure. This hierarchical VAE significantly improved sample quality and interpolation of musical phrases compared to a flat baseline. MusicVAE's latent space enabled operations like blending two drum patterns or melodies by averaging their codes, yielding coherent hybrids. Focusing specifically on drum performances, GrooVAE [15] applied a VAE and seq2seq model to learn the nuances of human drumming. Gillick and colleagues created the Groove MIDI Dataset (GMD) of 13.6 hours of professional drummers, including aligned MIDI and audio with fine timing and velocity data. With this data, they trained GrooVAE models on tasks like Humanization (generating a realistic performance from a quantized drum score) and Drum Infilling or Tap2Drum (adding complementary drum hits or mapping a tapped rhythm to a full drum pattern). The GrooVAE models could generate expressive drum performances that convincingly mimic human "groove", outperforming prior approaches. Notably, their Tap2Drum application allows a user to input a simple tap pattern (e.g. a basic beat on one drum) and the model "imagines" a full drum arrangement with appropriate accents and fills. This provides a form of high-level controllability while retaining realistic microtiming. Subsequent work has built on these ideas; for example, PocketVAE [16] uses a two-step VQ-VAE to first generate a refined drum note pattern (adding ghost notes, etc.), then apply separate modules for velocity and micro-timing adjustments, effectively applying a "pocket" groove to a user's rough beat. PocketVAE also allows conditional generation, such as specifying genre or swapping in groove patterns from reference performances. These VAE-based models highlight how latent representations can capture drum feel and support structured transformations of rhythms.

2.3.3 Transformer-Based Models

Transformers [17] have pushed the state of the art in symbolic music generation, especially for longer compositions. Music Transformer [9] showed that self-attention can learn musical long-term dependencies, like repetition and variation, better than RNNs. Building on this, Pop Music Transformer [6] introduced a new event representation called REMI (REvamped MIDI) to address the rhythmic shortcomings of the naive MIDI encoding. REMI adds explicit Bar and Position (sub-beat) tokens, as well as Tempo and Chord events, to give the model a sense of metrical structure. Using this representation with a Transformer-XL [18], Huang and Yang's model was able to generate multi-minute pop piano compositions with much clearer downbeat consistency and phrase structure than the baseline Transformer with a standard encoding. In fact, by simply changing the input representation (and not the network architecture), they improved the musical coherence (especially rhythm) over the previous Music Transformer model. The Pop Music Transformer demonstrates that incorporating musical domain knowledge (like bars and beats) into deep models can significantly enhance generation quality. Other Transformer-based systems, such as OpenAI's MuseNet [19] and Microsoft's MMM [20], have extended sequence models to multiple instruments including drums. MuseNet uses token prefixes to condition generation on style or composer, and can produce drum accompaniments in

various genres. MMM (Multitrack Music Machine) likewise generates multi-track music with a single Transformer by interleaving instruments, using special tokens to handle instrument changes. These show the growing capacity of Transformers to handle drum pattern generation as part of full arrangements. Compared to RNNs, Transformers can capture longer repetitions (like a drum groove that persists yet evolves over dozens of bars) and handle parallel structure, though they require large training corpora.

2.3.4 Generative Adversarial Networks and Other Models

While less common than autoregressive models, Generative Adversarial Networks (GANs) [21] have also been applied to symbolic music. MuseGAN [7] is a notable GAN-based system that generates multi-track music (bass, drums, guitar, etc.) using a convolutional GAN on pianoroll representations. It can produce coherent drum grooves alongside other instruments, and even allows simple conditioning like fixing one track while generating others. GAN approaches can excel at producing realistic local texture (e.g. plausible drum hit patterns) but often struggle with global structure without additional constraints. Another recent approach, Notochord [22], does not use GANs but is worth noting as a non-autoregressive probabilistic model. Notochord introduces a flexible event-factorization and uses an RNN-based architecture designed for real-time interaction. It can generate multi-track MIDI (including drums) with latency under 10 ms per event, enabling live collaboration with human performers. Notochord's probabilistic framework permits fine-grained control during generation: for example, a user or external program can constrain the next note to be a snare hit at a certain minimum time, or fix a desired pitch or velocity, while the model fills in the rest of the context. This approach blends ideas from autoregressive models and interactive music systems, prioritizing responsiveness and user control over long-term planning. Overall, deep learning has given rise to a rich landscape of drum generation models, from VAEs that learn a latent groove space to Transformers that capture song-length drum structures – far surpassing the capabilities of earlier methods.

2.3.5 Controllability in Generation

An important theme in recent work is controllability – allowing human users to guide or shape the generative process. Purely autonomous music generation can feel like a "black box," so many systems now include mechanisms for user control or conditional generation. We have already seen several examples: Piano Genie allows a human to drive a performance with simple button presses, effectively channeling high-level intentions (like rhythm and contour) into detailed music via the learned autoencoder mapping [11]. In drum generation, Tap2Drum (from GrooVAE) is a clear example of a controllable interface: the user provides a rudimentary pattern (e.g. tapping quarter notes on a single drum) and the model elaborates it into a full drum beat with appropriate "groove" (timing and dynamics) and embellishments [15]. This lowers the barrier for non-drummers to create complex, human-like drum tracks by starting from a simple idea. Other controllable dimensions include style, genre, and complexity. For instance, PocketVAE can transfer a specific drummer's feel or apply genre-specific groove patterns to a base beat, and it allows toggling of elements like ghost notes or swing level via conditional inputs [16]. Conditioned generation has also been explored in multi-track models – e.g. MuseGAN can take a fixed bassline or chord progression and generate a matching drum track, thereby

giving the user control over one aspect while automating others. Notochord represents a sophisticated approach to controllability: because it's a probabilistic model of next events, a user (or algorithmic rules) can steer the generation at runtime by constraining certain attributes of upcoming notes (instrument, pitch class, timing) without stopping the music [22]. This enables interactive improvisation scenarios where, say, a performer can cue the AI drummer to hit a crash cymbal on the next downbeat or to lay back for a few beats. Apple's Logic Pro in its recent versions (Logic Pro 11 on Mac and iPad), has introduced powerful AI-driven "Session Players," including an enhanced virtual Drummer that can intuitively generate rhythmic backing tracks in response to user-defined settings such as style, complexity, swing, and fills [23]. While the technical implementation of the Drummer has not been publicly disclosed, its behavior, such as consistent phrase structure, predefined drummer personas, and modulation via high-level parameters might suggest a combination of a rule-based and a hybrid expert system rather than a learned generative model. Unlike conventional loop-based or grid editing tools, the Drummer adapts to chord progressions, tempo, and arrangement in real time, offering structured control via UI "XY" pads and expressive variation through discrete parameter dials and sliders. Logic Pro's Drummer represents a commercially successful approach to interactive, AI-assisted drum generation, allowing non-drummers to produce expressive and stylistically appropriate grooves without manual sequencing. In summary, modern drum generation systems increasingly support human-in-the-loop creativity, whether through high-level constraints (genre, intensity), example-based conditioning (using reference rhythms or styles), or real-time performance control. Such controllability is crucial in practical applications, as it combines the generative power of AI with the musician's intention and intuition.

2.4 Datasets for Symbolic Drum Music

The progress in drum pattern generation has been fueled in part by new datasets of drum performances. A pivotal contribution is the Groove MIDI Dataset (GMD) introduced by Gillick et al. [15]. GMD contains 13.6 hours of expert drum performances recorded on electronic drum kits, with each hit precisely aligned to a metronome timeline and annotated with its velocity (loudness). Uniquely, the dataset also provides human-reference "quantized" versions of each beat (the drum scores) paired with the expressive performances. This allows training models to learn mappings from a flat drum pattern to a "groovy" human-played version (the Humanization task). GMD covers a variety of styles (rock, funk, jazz, etc.) and includes multiple drummers, providing a rich foundation for learning generalizable drum generation. Many deep learning models discussed (GrooVAE, PocketVAE, etc.) directly leverage this dataset for training and evaluation. Other datasets and resources have also contributed: Lakh MIDI [24] (though not specific to drums) contains thousands of songs in MIDI, many with drum tracks, which Notochord used for broad pretraining. Some researchers have used custom collections of drum loops or transcription data (e.g. extracted drum MIDI from audio recordings) to supplement training. Beyond MIDI, there are also audio-based drum datasets (ENST Drum [25], etc.), but for symbolic generation the focus is on MIDI or event data. As research progresses, we may see even larger and more diverse drum datasets – for example, community-driven collections of drum grooves or expansions of GMD – which will further improve the realism and variety of generative models.

2.5 Evaluation of Generative Music Models

Evaluating the quality of generated music, including drum patterns, remains a challenging problem. Traditionally, new generative models have been evaluated with subjective listening tests (asking human listeners to judge realism or preference) or sometimes with ad-hoc metrics (like comparing note frequency distributions). However, subjective tests are resource-intensive and can yield inconsistent results if not carefully designed. In response, researchers have proposed objective evaluation frameworks for symbolic music. Yang and Lerch [26] introduced a toolkit for multi-criteria evaluation of generated music, aiming to capture basic musical validity and diversity in a reproducible way. Instead of trying to measure "beauty" or high-level structure directly, their framework computes a suite of musically-informed statistics on the output, such as pitch range, rhythmic variety, tonal harmony indicators, repetition rates, etc. These metrics serve as formative evaluation, helping diagnose whether a model's output has issues like trivial repetition, lack of rhythmic diversity, or implausible note combinations. By comparing these metrics to those of real music or other models' outputs, researchers can quantitatively assess improvements. For example, in their experiments, they evaluated state-of-the-art models (including Magenta's and others) on metrics like number of empty bars, scale consistency, drum hit entropy, etc., revealing that many models struggled with basic musical coherence. Another recent proposal is the Fréchet Music Distance (FMD) [27], which adapts the idea of Fréchet Inception Distance from images to musical features, though it's still emerging. Ultimately, a combination of objective metrics and human listening is used to validate drum generation systems. As generative models become more musically sophisticated, developing reliable evaluation methods (covering both technical correctness and aesthetic plausibility) is an active area of research. The work of Yang and Lerch [26] is an important step toward standardizing evaluation for symbolic music generation, enabling more meaningful comparisons between models and helping to track progress in the field.

Chapter 3

Data

3.1 The GrooveIQ MIDI Dataset

To support research in drum pattern generation, we present the GrooveIQ MIDI Dataset — a large-scale, open-source symbolic drum corpus. This dataset builds upon the Groove MIDI Dataset [15] by aggregating symbolic drum performances from multiple online sources and applying a rigorous data cleaning, mapping, and normalization pipeline. Key characteristics of the dataset include:

- The dataset comprises approximately 164,079 MIDI files, totaling over 591,000 measures of expressive drumming. These include both drum beats and fills of varying complexity.
- Each MIDI file is annotated with a unique integer identifier, the drum mapping used for General MIDI alignment, a style label, time signature, a beat/fill type tag, and additional metadata.
- Style labels were heuristically inferred from directory structures and filenames using string-matching techniques against a large corpus of musical genres [28]. Compound genres (e.g., *Blues Rock*) were collapsed into their primary styles (e.g., *Blues*) to improve consistency. Performances for which no style could be confidently inferred were labeled as *Unknown*. The resulting dataset spans 29 unique styles, with metadata preserved for future refinement or fine-grained classification.
- Drum mappings [1] were resolved through manual inspection of file structures, documentation, and source formats. Files with irrecoverable or ambiguous mappings were discarded.
- 87.8% of the files are in 4/4 time; for consistency, only these files are used in this work. Non-4/4 files are retained in the dataset for future work exploring broader metrical diversity.
- Following prior work [15], all files are mapped to the General MIDI standard with 22 canonical drum hit categories, which are then reduced to a consistent 9-hit representation. Table 3.1 details this mapping, balancing expressivity with tractability for learning and evaluation
- A predefined train/validation/test split is provided to facilitate reproducible experiments and benchmark comparisons.

[1] Drum mapping refers to the process of aligning different MIDI note numbers or track conventions to a canonical drum set representation (e.g., kick, snare, hi-hat). This is necessary because MIDI files often use inconsistent or custom drum note encodings.

While this dataset aims to be comprehensive and diverse, certain limitations remain. Genre labels are weakly supervised and should be treated as noisy. Performance metadata may vary in quality across sources, and stylistic coverage may be biased toward Western popular and rock-oriented genres.

We release the GrooveIQ Dataset publicly to encourage further research in symbolic music generation and rhythmic style transfer. Complete code and preprocessing scripts are provided to enable full reproducibility.

TABLE 3.1: List of Drum Categories

Pitch	GM Mapping	Drum Category
36	Bass Drum 1	Bass (36)
38	Acoustic Snare	Snare (38)
40	Electric Snare	Snare (38)
37	Side Stick	Snare (38)
48	Hi-Mid Tom	High Tom (50)
50	High Tom	High Tom (50)
45	Low Tom	Mid Tom (47)
47	Low-Mid Tom	Mid Tom (47)
43	High Floor Tom	Low Tom (43)
58	Vibraslap	Low Tom (43)
46	Open Hi-Hat	Open Hi-Hat (46)
26	N/A	Open Hi-Hat (46)
42	Closed Hi-Hat	Closed Hi-Hat (42)
22	N/A	Closed Hi-Hat (42)
44	Pedal Hi-Hat	Closed Hi-Hat (42)
49	Crash Cymbal 1	Crash Cymbal (49)
55	Splash Cymbal	Crash Cymbal (49)
57	Crash Cymbal 2	Crash Cymbal (49)
52	Chinese Cymbal	Crash Cymbal (49)
51	Ride Cymbal 1	Ride Cymbal (51)
59	Ride Cymbal 2	Ride Cymbal (51)
53	Ride Bell	Ride Cymbal (51)

3.2 Preprocessing

We apply a series of preprocessing steps to simplify the modeling task while preserving musical expressivity.

First, we restrict our dataset to only include performances in 4/4 time, which accounts for approximately 87.8% of the total corpus. We make this choice for both pragmatic and musically grounded reasons: 4/4 is the most prevalent time signature in popular and contemporary music, and focusing on a single meter avoids introducing variability in temporal structure that could complicate model training and evaluation.

Next, all drum hits are mapped to a reduced set of 9 canonical drum categories, following Gillick et al. [15]. These categories capture the most common components of a standard drum kit—bass drum, snare, hi-hats, toms, and cymbals. The full list of categories and their corresponding MIDI mappings is shown in Table 3.1.

We then segment the remaining drum performances into fixed-length patterns using a sliding window. Specifically, we extract 2-bar (measure) segments using a window size of 2 bars and a hop size of 1 bar. This decision reflects common practice in music production, where 2-bar loops are widely used, and offers a balance between capturing meaningful variation and maintaining manageable sequence length for listening evaluations and training efficiency.

Finally, the resulting sequences are split into training, development, and test sets using a stratified partitioning strategy that preserves the distribution of styles, measured in number of bars per style across each split.

3.3 Dataset Statistics and Comparison with Groove MIDI

TABLE 3.2: Samples count per style across datasets

Style	**2bar-GrooveIQ**	**2bar-GMD**
metal	63,469 (16.5%)	-
rock	90,539 (23.6%)	6,612 (30.9%)
unknown	67,991 (17.7%)	-
latin	14,380 (3.7%)	3,869 (18.1%)
jazz	31,735 (8.3%)	2,492 (11.7%)
blues	23,979 (6.3%)	87 (0.4%)
fusion	8,935 (2.3%)	-
progressive	2,339 (0.6%)	-
country	8,145 (2.1%)	122 (0.6%)
pop	13,096 (3.4%)	314 (1.5%)
electronic	7,300 (1.9%)	-
rnb	8,504 (2.2%)	-
punk	11,166 (2.9%)	273 (1.3%)
funk	6,619 (1.7%)	-
reggae	3,709 (1%)	281 (1.3%)
soul	5,161 (1.3%)	613 (2.9%)
gospel	2,551 (0.7%)	69 (0.3%)
hiphop	2,870 (0.7%)	866 (4.0%)
disco	2,144 (0.6%)	-
indiefolk	1,890 (0.5%)	-
americana	1,374 (0.4%)	-
indie	2,143 (0.6%)	-
afrobeat	1,030 (0.3%)	976 (4.6%)
showtunes	130 (0%)	-
dance	544 (0.1%)	544 (2.6%)
neworleans	739 (0.2%)	739 (3.5%)
afrocuban	769 (0.2%)	769 (3.6%)
middleeastern	118 (0%)	118 (0.6%)
highlife	142 (0%)	142 (0.7%)

We refer to the pre-processed version of the GrooveIQ dataset as **2bar-GrooveIQ** and compare it with **2bar-GMD**, a similarly processed version of the Groove MIDI Dataset (GMD).

TABLE 3.3: Counts of beat and fill samples across datasets

Type	2bar-GrooveIQ	2bar-GMD
Beat	363,973 (94.9%)	21,091 (98.6%)
Fill	19,538 (5.1%)	291 (1.4%)

TABLE 3.4: Per-feature absolute measurements (mean ± std) across datasets

Feature	2bar-GrooveIQ	2bar-GMD
Total Density	0.07 ± 0.03	0.06 ± 0.02
Total Complexity	0.55 ± 0.39	0.79 ± 0.41
Total Avg. Intensity	0.72 ± 0.15	0.56 ± 0.14
Lowness	0.20 ± 0.16	0.10 ± 0.07
Midness	0.21 ± 0.15	0.23 ± 0.15
Highness	0.30 ± 0.17	0.28 ± 0.15
Combined Syncopation	0.54 ± 0.39	0.78 ± 0.41
Polyphonic Syncopation	2.52 ± 6.23	3.14 ± 6.31
Laidbackness	0.00 ± 0.17	-0.06 ± 0.31
Swingness	0.26 ± 0.45	0.53 ± 0.52
Timing Accuracy	7.19 ± 6.70	20.47 ± 9.38

Table 3.2 presents the sample counts per style in both datasets, revealing stark stylistic differences. 2bar-GrooveIQ, which is sourced from online MIDI repositories, is dominated by Western popular genres such as rock, metal, and pop. These styles are widely available in commercial loop packs and DAW libraries, likely leading to their over-representation. In contrast, 2bar-GMD, derived from human drumming performances, features a richer presence of groove-oriented and rhythmically expressive styles including latin, jazz, hip-hop, and various afro-diasporic genres, suggesting a deliberate emphasis on cultural and rhythmic diversity. Meanwhile, its stronger representation of sequenced genres like electronic and metal underscore its synthetic and loop-based origins. These stylistic contrasts potentially highlights the complementary nature of the datasets: GrooveIQ provides scale and stylistic breadth, while GMD contributes expressive depth and human nuance.

Table 3.3 illustrates a notable difference in the distribution of beat and fill segments across the datasets. While both are dominated by beat segments, fills make up 5.1% of 2bar-GrooveIQ but only 1.4% of 2bar-GMD. This discrepancy likely stems from the nature of the sources: GrooveIQ includes curated loops where fills are deliberately included because they are musically desirable and often difficult to program manually. In contrast, GMD comprises continuous human performances in which fills occur more sparingly and are not always explicitly segmented. Consequently, fill examples in GMD are both rare and potentially underrepresented. These differences have implications for model training, particularly for fill generation, since models trained solely on GMD may lack sufficient exposure to stylistic and structural fill patterns.

To objectively characterize these datasets, we adopt the feature-based evaluation methodology proposed by Yang and Lerch [26], which compares distributions of hand-crafted musical descriptors across real and generated sequences. Specifically, they propose two analysis modes: (1) absolute comparisons of feature distributions,

and (2) relative comparisons between or within sets. In this section, we apply the first method to compare 2bar-GrooveIQ and 2bar-GMD. The second method is reserved for later evaluations of our trained models.

We extract the following features from all samples in both datasets:

- **Total Density**: Ratio of active steps (those with at least one hit) to total steps [29]
- **Total Complexity**: Combined metric based on density and syncopation [30, 31]
- **Total Average Intensity**: Average loudness across hits [31]
- **Lowness/Midness/Highness**: Proportional distribution of hits across low (kick, low tom), mid (snare, mid toms), and high (hi-hats, cymbals) frequency bands [29]
- **Combined Syncopation**: Sum of syncopation values across voices [31]
- **Polyphonic Syncopation**: Inter-voice syncopation, capturing temporal displacement across parts [30, 31]
- **Swingness**: Degree of swing in the rhythmic pattern [31]
- **Laidbackness**: Tendency to play behind or ahead of the metrical grid [31]
- **Timing Accuracy**: Mean microtiming deviation from the quantized grid [31]

Table 3.4 compares the absolute statistics (mean ± standard deviation) of these features across the two datasets. GrooveIQ shows traits typically associated with machine-generated or DAW-edited loops: it is slightly denser, more intense (louder), and significantly more temporally rigid (lower timing deviation and swing). In contrast, GMD exhibits higher rhythmic complexity, greater syncopation (both combined and polyphonic), and more expressive timing characteristics, such as stronger swing and a mild tendency to play behind the beat (negative laidbackness). The timing accuracy metric in particular highlights the distinction, with GMD showing substantially greater microtiming variation, indicative of natural human performance. These findings may support a division of roles: GrooveIQ serving well as a structured input domain, while GMD offering expressive target behavior suitable for modeling human-like generative output.

3.4 Data Representation

3.4.1 The Fixed-Grid Representation

Motivated by the approach introduced in Gillick et al. [15], we represent all sequences using a *Fixed-Grid* representation. Formally, this representation is defined as a tensor of shape $(T \times E \times M)$, where T denotes the number of timesteps per sequence, E is the number of instrument categories (or event slots) per timestep, and M is the number of modification parameters used to encode expressive timing and dynamics.

We use 16th notes as the fundamental rhythmic unit and encode each 2-bar segment accordingly, resulting in $T = 33$ timesteps (32 for the 2-bar grid and one additional step to capture overflow hits at the start of the next segment). Each drum

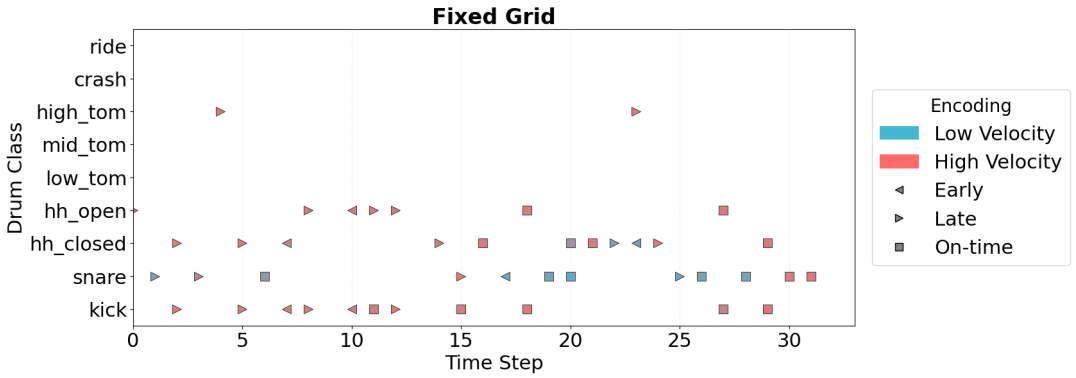

FIGURE 3.1: Fixed-Grid representation of a two-bar GrooveIQ sample at 16th note resolution. 7 notes were lost due to hits at the same time step.

hit is mapped to its nearest 16th-note metrical position. In cases where multiple hits occur in the same timestep and instrument category, only the hit with the highest velocity is retained. While this preprocessing discards fine-grained temporal nuance (e.g., rapid rolls occurring faster than 16th notes), we find that most perceptual expressivity is preserved at this resolution.

After encoding, each sequence is represented as a fixed-length tensor with shape $T \times E \times M$, where $E = 9$ drum instrument categories and $M = 3$ modification parameters:

- **Hits (H)**: A binary-valued matrix indicating the presence or absence of drum onsets. Each column corresponds to one of the nine drum instruments, and each row represents a single timestep.
- **Offsets (O)**: A real-valued matrix encoding microtiming deviations, with values in $[-0.5, 0.5)$. These offsets represent the signed temporal difference between the actual onset time and the nearest 16th-note grid position, allowing the preservation of humanization effects such as swing or laid-back timing.
- **Velocities (V)**: A real-valued matrix capturing dynamic intensity. MIDI velocities (integers in $[0, 127]$) are normalized to the range $[0, 1]$ for continuous representation.

This representation offers a practical balance between temporal resolution and model tractability. It enables efficient batch processing while still retaining essential expressive cues through the velocity and offset channels. Figure 3.1 illustrates a drum pattern encoded in the Fixed-Grid format.

3.4.2 Note Loss and the Flexible-Grid Representation

To investigate loss of information due to our encoding scheme, We define *note loss* as the ratio of dropped notes to the total number of notes in a given sample. A higher note loss indicates a greater proportion of expected drum events being missed or discarded during preprocessing. Using this metric, we compute and visualize the distribution of note loss per style at a 16th-note resolution.

Figure 3.2 presents a violin plot showing the distribution of note loss across musical styles. Each violin illustrates the density and variability of note loss values

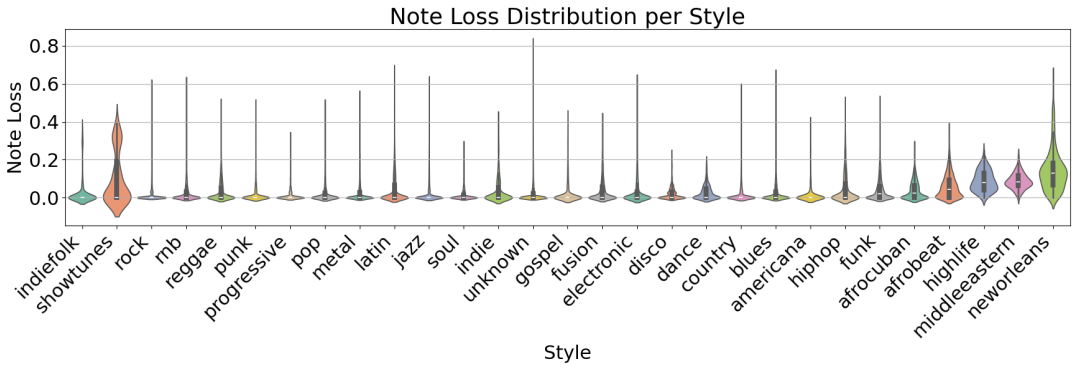

FIGURE 3.2: Distribution of note loss across styles using the Fixed-Grid representation at a 16th-note resolution.

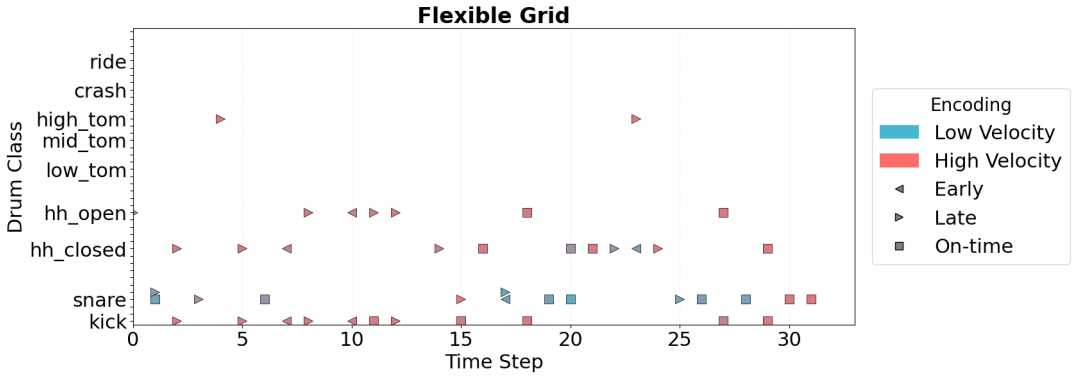

FIGURE 3.3: Flexible-Grid representation of a two-bar GrooveIQ sample at 16th note resolution. Note the snare hits that are mapped to separate slots to prevent information loss.

for that style. Narrow violins centered near zero indicate consistent, low note loss, while wider violins or long tails suggest high variability and frequent reconstruction failures.

Several trends are evident. Popular Western styles such as *rock*, *pop*, *hiphop*, *electronic*, and *metal* exhibit consistently low note loss with tight distributions. These styles are well-represented in training data and typically adhere to grid-aligned rhythmic patterns, making them easier for models to reconstruct accurately.

Conversely, rhythmically rich or culturally specific styles such as *neworleans*, *afrocuban*, *highlife*, and *middleeastern* exhibit higher median note loss and greater variability. These styles often feature syncopation, polyrhythms, swing, and expressive microtiming—all of which pose challenges to models trained predominantly on Western, straight-time datasets. Additionally, many of these genres are underrepresented in the dataset, limiting exposure during training and increasing generalization error.

While low note loss in mainstream genres may reflect a system's ability to handle structurally consistent input, the elevated note loss in more expressive or underrepresented styles underscores the limitations of the Fixed-Grid representation. In particular, its one-event-per-category constraint can lead to note omissions when multiple events occur within the same timestep.

The *Flexible-Grid* representation proposed by Gillick et al. [32], addresses this limitation, which extends the Fixed-Grid format by dynamically allocating additional event slots per timestep. Unlike increasing the resolution (which can harm generalization and increase sparsity), the Flexible-Grid allows multiple hits per instrument category at a given timestep by expanding the E dimension. The number of required slots per drum category can be calculated in advance to ensure that all sequences in the dataset can be fully encoded without information loss.

Table 3.5 shows the number of additional slots needed per drum category to support lossless encoding of all sequences in the 2bar-GrooveIQ dataset. Figure 3.3 illustrates a drum pattern encoded the Flexible-Grid format.

TABLE 3.5: Statistics of the GrooveIQ Dataset used to build a Flexible Grid Representation at 16th note resolution.

Drum	**Max # of Onsets within time step**
Kick	3
Snare	9
Closed Hi-Hat	5
Open Hi-Hat	6
Low Tom	4
Mid Tom	4
High Tom	4
Crash Cymbal	5
Ride Cymbal	6

Chapter 4

Methodolody

4.1 Modeling Objective

The core modeling task in GrooveIQ is to learn a conditional mapping from a sparse, low-dimensional rhythmic control sequence—such as a 2-button input pattern—to a full drum performance that captures nuanced dynamics and microtiming deviations. This can be framed as a structured sequence generation problem, where the model learns to predict a high-dimensional, temporally aligned output from compact control signals.

4.1.1 Notation and Formulation

Let:

- T denote the number of timesteps (e.g., 33 for a 2-bar segment),
- E be the number of expressive drum instrument categories (e.g., 9),
- N be the number of control channels (e.g., 2),
- M be the number of expressive dimensions per instrument (e.g., hit presence, velocity, microtiming offset),
- $\mathbf{b} \in \mathbb{R}^{T \times N}$ denote the control sequence,
- $\mathbf{y} \in \mathbb{R}^{T \times E \times M}$ denote the target expressive drum performance.

The goal is to learn a mapping:

$$f_\theta : \mathbb{R}^{T \times N} \to \mathbb{R}^{T \times E \times M}$$

parameterized by θ, such that:

$$\hat{\mathbf{y}} = f_\theta(\mathbf{b}),$$

where $\hat{\mathbf{y}}$ is the predicted sequence and should approximate the ground truth $\mathbf{y}$ as closely as possible. Notably, the control input does not include expressive dimensions such as timing or velocity. This design choice reflects our goal of creating an interface accessible to amateur performers—requiring only simple, discrete button inputs to drive full expressive generation.

4.2 Probabilistic Modeling Formulation

To capture the variability in mapping sparse control sequences to full drum performances, GrooveIQ employs a structured latent variable model with a time-varying latent sequence $\mathbf{z}_{1:T} \in \mathbb{R}^{T \times d_z}$.

We model the conditional distribution of expressive sequences given control inputs as:

$$p_\theta(\mathbf{y} \mid \mathbf{b}) = \int p_\theta(\mathbf{y} \mid \mathbf{b}, \mathbf{z}_{1:T})\, p_\theta(\mathbf{z}_{1:T} \mid \mathbf{b})\, d\mathbf{z}$$

Here $\mathbf{z}_{1:T}$ is a sequence of latent variables preferably capturing global and local performance characteristics.

The true posterior distribution $p_\theta(\mathbf{z}_{1:T} \mid \mathbf{y}, \mathbf{b})$, defined by Bayes' rule, is intractable due to the marginal likelihood in the denominator. To enable efficient training, we introduce a variational approximation $q_\phi(\mathbf{z}_{1:T} \mid \mathbf{y}, \mathbf{b})$ and optimize the evidence lower bound (ELBO):

$$\log p_\theta(\mathbf{y} \mid \mathbf{b}) \geq \mathbb{E}_{q_\phi(\mathbf{z}_{1:T} \mid \mathbf{y}, \mathbf{b})} \left[\log p_\theta(\mathbf{y} \mid \mathbf{b}, \mathbf{z}_{1:T})\right] - \mathrm{KL}\left[q_\phi(\mathbf{z}_{1:T} \mid \mathbf{y}, \mathbf{b}) \,\|\, p_\theta(\mathbf{z}_{1:T} \mid \mathbf{b})\right]$$

The prior and posterior are parameterized by neural networks that generate per-timestep Gaussian distributions over the latent sequence. The model is trained to maximize this lower bound, encouraging reconstructions while learning a temporally structured latent representation aligned with the control input.

4.3 The GrooveIQ Architecture

The GrooveIQ model follows an encoder–latent–decoder paradigm conditioned on a sparse control interface, designed to transform simple, binary rhythmic inputs into drum performances. This section outlines the core components and operations of the model. Figure 4.1 summarizes this.

4.3.1 Input Representation

The model takes as input a tensor $\mathbf{y} \in \mathbb{R}^{B \times T \times E \times M}$ representing a batch of drum performances, where B is the batch size.

4.3.2 Control Sequence Derivation

A core challenge addressed in this work is the absence of a deterministic inverse mapping from a full drum performance to a sparse, low-dimensional rhythmic control sequence. To bridge this gap between compact user input and expressive drum generation, GrooveIQ explores two complementary strategies for deriving the control sequence $\mathbf{b} \in \mathbb{R}^{T \times N}$ used to condition the model: (1) a learned representation approach that infers control from the performance itself, and (2) a heuristic simplification approach that generates control signals through musically informed reduction. Together, these methods support both flexible learned control and interpretable, rule-based conditioning for training and evaluation.

(1) Representation Learning via Encoder Projection. In this setting, the model learns a control sequence jointly with the generation task. The encoder's contextualized output $\mathbf{d} \in \mathbb{R}^{B \times T \times D}$ is projected into button logits $\mathbf{b}^{\text{raw}} \in \mathbb{R}^{B \times T \times N}$ using a linear layer. These logits are passed through a sigmoid activation and thresholded using a straight-through estimator [33] to produce binary button activations $\mathbf{b}_{\textbf{learned}} \in \{0, 1\}^{B \times T \times N}$.

To prevent degenerate solutions and to encourage temporally sparse, interpretable button patterns, we apply one of two sparsity-inducing penalties to $\mathbf{b}^{\text{raw}}$ during training:

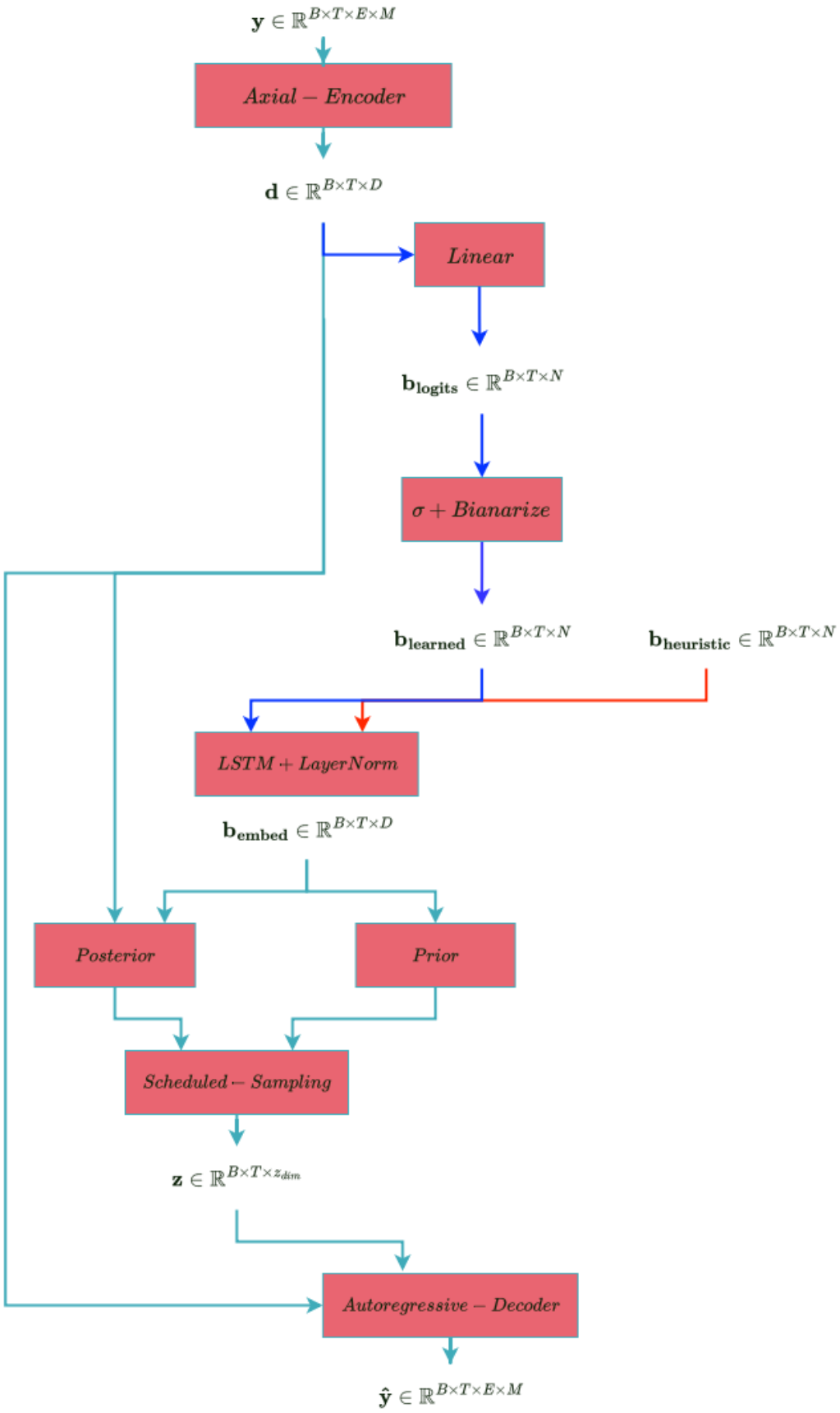

FIGURE 4.1: Overview of the GrooveIQ architecture. The blue path denotes the representation learning setup, where the button sequence is learned and regularized for sparsity. The red path illustrates the self-supervised setup, where the button sequence is derived heuristically from the input performance. Both paths share a common decoding pipeline.

- **L1 Penalty:** Encourages element-wise sparsity by minimizing the absolute activation magnitudes across all timesteps and channels.
- **Group Penalty:** Encourages entire timesteps to be inactive by computing the L2 norm across channels at each timestep and summing over time [34].

These penalties are described in detail in Section 4.4.3. Regardless of the constraint, dropout is applied to the learned button embeddings to prevent overreliance on fixed activation patterns.

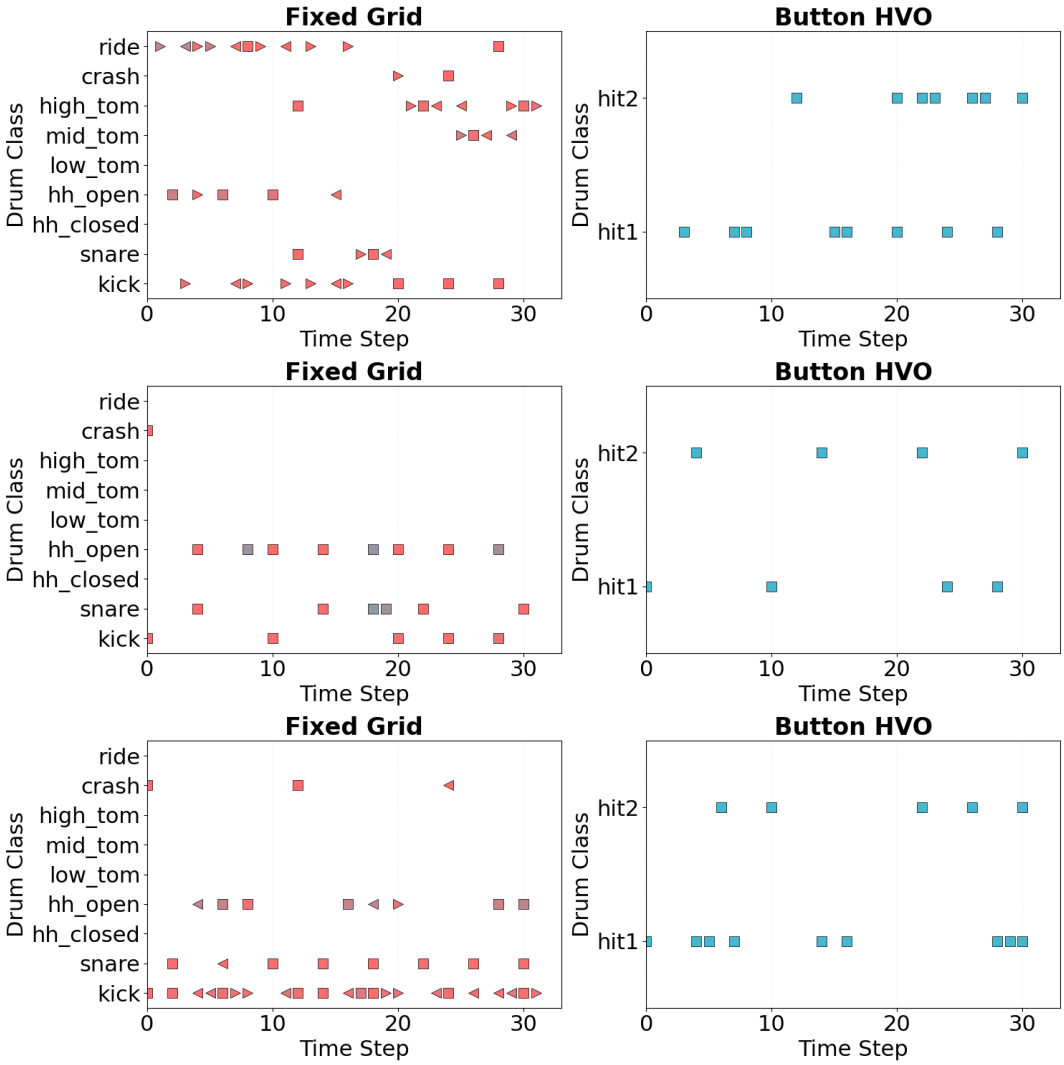

FIGURE 4.2: Examples of three fixed-grid representations (left) and their corresponding button sequences derived using the heuristic simplification method (right).”

(2) Heuristic Simplification-Based Derivation. In this setting, we hypothesize that a valid control sequence can be derived as a simplified rhythmic abstraction of the original drum performance. To this end, we also explore rhythmically-informed heuristics for extracting control sequences directly from the input.

The 9-instrument drum kit is collapsed into three control groups:

- **Low:** kick, low tom
- **Mid:** snare, mid tom, high tom
- **High:** closed hi-hat, open hi-hat, cymbal, ride

A sliding-window simplification procedure is applied over the performance grid $\mathbf{y}$ to identify salient hits in the low and mid groups (the high group is ignored). Within each window, hits above a sampled velocity threshold are retained probabilistically. The top-k hits are selected by velocity, perturbed by timing and velocity jitter to simulate human-like variation. Miss and spurious probabilities inject randomness to mimic imperfect input gestures. The final output is a button sequence tensor $\mathbf{b} \in \mathbb{R}^{T\times N\times 3}$ with binary hit, velocity, and timing offset values, though only the hit channel is retained for deriving $\mathbf{b}_{\mathbf{heuristic}} \in 0, 1^{B\times T\times N}$ in our experiments. The simplification heuristic incorporates the following stages:

- **Window Sampling:** Variable window sizes are selected probabilistically, emphasizing metrical variation.
- **Velocity Thresholding:** Each window samples a threshold as a fraction of the maximum velocity in the sequence.
- **Top-k Selection:** Within each window, a fixed number of high-velocity hits are selected.
- **Augmentations:** Jitter and drop-in/drop-out perturbations are added to simulate imprecise human control.

This approach provides a controllable and interpretable input modality that is musically grounded. Dropout is applied to button embeddings in this setting as well, ensuring the model generalizes across multiple abstraction patterns. Figure 4.2 illustrates the effect of the heuristic simplification method, showing how dense fixed-grid drum representations are transformed into sparse, low-dimensional button sequences suitable for control.

4.3.3 Encoder

The encoder processes $\mathbf{y}$ using a multi-layer Axial Transformer [35], which applies alternating temporal and instrument-wise self-attention to model structured dependencies across time and instrument dimensions. The output is flattened along the instrument axis and projected to form a contextual representation:

$$\mathbf{d} = \text{Linear}(\text{Reshape}(\text{AxialTransformer}(\mathbf{y}))) \in \mathbb{R}^{B\times T\times D}$$

4.3.4 Control Path

The learned or heuristically-derived binarized control sequence is embedded using an LSTM followed by LayerNorm to yield a temporally contextualized control embedding:

$$\mathbf{b}_{\text{embed}} = \text{LayerNorm}(\text{LSTM}(\bar{\mathbf{b}})) \in \mathbb{R}^{B\times T\times D}$$

4.3.5 Latent Networks

Posterior. During training, the encoder output $\mathbf{d}$ and control embedding $\mathbf{b}_{\text{embed}}$ are divided into fixed-size chunks. Each chunk is averaged over time and concatenated to form the input to the posterior network:

$$\mathbf{h}_t^{\text{post}} = \text{ChunkAvg}(\mathbf{d}_{w_t}) \,\|\, \text{ChunkAvg}(\mathbf{b}_{\text{embed},w_t}) \in \mathbb{R}^{2D}$$

$$\boldsymbol{\mu}_t^{\text{post}}, \log \boldsymbol{\sigma}_t^{2,\text{post}} = \text{Linear}_{\text{post}}(\mathbf{h}_t^{\text{post}})$$

Prior. The prior is computed using an LSTM over the control embedding, followed by chunking and temporal averaging:

$$\mathbf{h}_t^{\text{prior}} = \text{ChunkAvg}\left(\text{LSTM}(\mathbf{b}_{\text{embed}})_{w_t}\right) \in \mathbb{R}^{D}$$

$$\boldsymbol{\mu}_t^{\text{prior}}, \log \boldsymbol{\sigma}_t^{2,\text{prior}} = \text{Linear}_{\text{prior}}(\mathbf{h}_t^{\text{prior}})$$

Sampling. A latent sequence $\mathbf{z}_{1:T}$ is sampled via the reparameterization trick [36]:

$$\mathbf{z}_t = \mu_t + \sigma_t \odot \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, I)$$

Scheduled Sampling. To regularize training and encourage consistency between the prior and posterior distributions, we apply scheduled sampling to interpolate between posterior and prior latent sequences [37]:

$$\mathbf{z}_t = m \cdot \mathbf{z}_t^{\text{post}} + (1 - m) \cdot \mathbf{z}_t^{\text{prior}}$$

where $m \sim \text{Bernoulli}(p)$ is a binary mask sampled independently for each training instance, with probability p determining the reliance on the posterior sample. As training progresses, p is annealed to favor the prior network at inference time.

4.3.6 Decoder

The decoder is a multi-layer Transformer Decoder [17] conditioned on both the latent sequence and the button embedding. It operates autoregressively over a shifted version of the input sequence, prepended with a learnable start-of-sequence (SOS) token:

$$\tilde{\mathbf{y}} = [\text{SOS},\ \mathbf{y}_{1:T-1}]$$

This sequence is projected and enriched with positional encodings:

$$\mathbf{x}_t = \text{PosEnc}(W_{\text{in}} \cdot \text{vec}(\tilde{\mathbf{y}}_t))$$

The memory input to the decoder is formed by projecting and summing the latent and control sequences:

$$\mathbf{m}_t = \text{PosEnc}(W_z \cdot \mathbf{z}_t + W_b \cdot \mathbf{b}_{\text{embed},t})$$

The decoder attends to past targets (causal self-attention) and to the full memory sequence (causal or non-causal cross-attention depending on the generation mode).

Output Projection. The decoder output is mapped back to the performance space via a linear projection:

$$\hat{\mathbf{y}}_t = \text{Reshape}\left(W_{\text{out}} \cdot \mathbf{z}_t\right) \in \mathbb{R}^{E \times M}$$

where $M = 3$ corresponds to hit logits, velocity, and timing offset.

Output Activation. The final outputs are interpreted as follows:

$$\hat{h}_{b,t,e} = \hat{\mathbf{y}}_{b,t,e,0} \in [0, 1]$$

$$\hat{v}_{b,t,e} = \frac{1}{2}\left(\tanh(\hat{\mathbf{y}}_{b,t,e,1}) + 1\right) \in [0, 1]$$

$$\hat{o}_{b,t,e} = 0.5 \cdot \tanh(\hat{\mathbf{y}}_{b,t,e,2}) \in [-0.5, 0.5]$$

representing the hit logits, velocity and offset predictions respectively.

4.4 Learning Objectives and Constraints

Training in GrooveIQ follows the principles of variational inference, where the goal is to maximize the log-likelihood of the observed drum sequence $\mathbf{y}$ conditioned on a sparse control sequence $\mathbf{b}$. Since the model includes a latent sequence $\mathbf{z}_{1:T}$, the marginal likelihood $p_\theta(\mathbf{y} \mid \mathbf{b})$ becomes intractable to compute directly due to the integral over the latent space. Instead, we optimize its variational lower bound, or *evidence lower bound* (ELBO):

$$\log p_\theta(\mathbf{y} \mid \mathbf{b}) \geq \mathbb{E}_{q_\phi(\mathbf{z}_{1:T} \mid \mathbf{y}, \mathbf{b})}\left[\log p_\theta(\mathbf{y} \mid \mathbf{b}, \mathbf{z}_{1:T})\right] - \text{KL}\left[q_\phi(\mathbf{z}_{1:T} \mid \mathbf{y}, \mathbf{b}) \,\|\, p_\theta(\mathbf{z}_{1:T} \mid \mathbf{b})\right]$$

Where:

- $q_\phi(\mathbf{z}_{1:T} \mid \mathbf{y}, \mathbf{b})$ is the approximate posterior over the latent sequence, parameterized by the encoder and control path,
- $p_\theta(\mathbf{y} \mid \mathbf{b}, \mathbf{z}_{1:T})$ is the conditional likelihood modeled by the autoregressive decoder,
- $p_\theta(\mathbf{z}_{1:T} \mid \mathbf{b})$ is the prior over the latent sequence, parameterized by the control embedding.

This results in the following training objective:

1. **Reconstruction loss**: A differentiable approximation to the negative log-likelihood, computed between the predicted and ground-truth outputs.
2. **KL divergence loss**: A regularization term that aligns the approximate posterior $q_\phi(\mathbf{z}_{1:T})$ with the prior $p_\theta(\mathbf{z}_{1:T})$, encouraging structured latent representations while preventing overfitting.
3. **Auxiliary losses**: Additional constraints such as control sparsity, temporal alignment, or button faithfulness. These are not part of the ELBO but act as inductive biases that guide training toward musically meaningful behaviors.

The total loss used during training is:

$$\mathcal{L}_{\text{GrooveIQ}} = \mathcal{L}_{\text{recons}} + \lambda_{\text{kl}} \cdot \mathcal{L}_{\text{kl}} + \lambda_{\text{aux}} \cdot \mathcal{L}_{\text{aux}}$$

where λ_{kl} and λ_{aux} are tunable coefficients that balance the contributions of the KL regularization and auxiliary losses.

4.4.1 Reconstruction Losses

The model predicts outputs as a tensor $\hat{\mathbf{y}} \in \mathbb{R}^{B \times T \times E \times M}$, decomposed into predicted hit logits $\hat{h}$, velocity $\hat{v}$, and offset $\hat{o}$. Let $\mathbf{y} = (h, v, o)$ be the corresponding ground truth.

The hit prediction loss is computed using binary cross-entropy (BCE) with a weighting term α to balance positive and negative classes:

$$\mathcal{L}_{\text{hit}} = \text{BCE}(\hat{h}, h;\ \text{pos\_weight} = \alpha)$$

For all experiments, α is set to 15.0 to emphasize true positives, compensating for the class imbalance between hit and non-hit frames.

To ensure that velocity and offset are only penalized when a hit is present in the ground truth, a hit penalty mask is applied:

$$m_{b,t,e} = \begin{cases} \gamma & \text{if } h_{b,t,e,0} = 1 \\ 0 & \text{otherwise} \end{cases}$$

Here, γ is the `hit_penalty` hyperparameter (also set to 15.0 in all experiments). The velocity and offset reconstruction losses are computed using masked mean squared error:

$$\mathcal{L}_{\text{vel}} = \frac{1}{BTE} \sum_{b,t,e} m_{b,t,e} \cdot (\hat{v}_{b,t,e} - v_{b,t,e})^2$$

$$\mathcal{L}_{\text{off}} = \frac{1}{BTE} \sum_{b,t,e} m_{b,t,e} \cdot (\hat{o}_{b,t,e} - o_{b,t,e})^2$$

These three terms are combined to form the total reconstruction loss:

$$\mathcal{L}_{\text{recons}} = \lambda_{\text{recons}} \cdot (\mathcal{L}_{\text{hit}} + \mathcal{L}_{\text{vel}} + \mathcal{L}_{\text{off}})$$

4.4.2 KL Divergence Loss

The KL divergence between the approximate posterior $q_\phi(\mathbf{z}_{1:T} \mid \mathbf{y}, \mathbf{b})$ and the prior $p_\theta(\mathbf{z}_{1:T} \mid \mathbf{b})$ is computed in closed form under the diagonal Gaussian assumption:

$$\mathcal{L}_{\text{kl}} = \frac{1}{T} \sum_{t=1}^{T} \text{KL}\left(q_\phi(\mathbf{z}_t \mid \cdot) \,\|\, p_\theta(\mathbf{z}_t \mid \cdot)\right)$$

This term encourages the learned latent codes to remain close to the prior induced by the control embedding, thereby regularizing the latent space.

4.4.3 Button Penalty Loss

In the learned control sequence setting, the model infers a dense, continuous representation of button activations at each timestep prior to binarization. To prevent overuse of this control interface and to encourage sparse, interpretable button usage, we introduce a regularization term on the raw (pre-sigmoid) button logits $\mathbf{b}^{\text{raw}} \in \mathbb{R}^{B \times T \times N}$.

We experiment with two forms of regularization:

$$\mathcal{L}_{\text{btn}} = \begin{cases} \sum_{b,t,n} \left| b^{\text{raw}}_{b,t,n} \right| & \text{(L1 penalty)} \\ \sum_{b,t} \left\| \mathbf{b}^{\text{raw}}_{b,t,:} \right\|_2 & \text{(Group penalty)} \end{cases}$$

L1 Penalty. The L1 norm penalizes the absolute magnitude of each individual button activation. This encourages element-wise sparsity, that is, fewer total activations across all buttons and timesteps. It biases the model toward using buttons only when they are truly necessary, reducing redundancy and over-activation.

Group Penalty. The group penalty, inspired by Group Lasso, computes the L2 norm over each button vector $\mathbf{b}_{b,t,:}$ at every timestep and then applies an L1-like penalty over time. This encourages temporal sparsity at the group level, favoring entire timesteps with no button activity at all. It helps isolate discrete moments of control activation, aligning well with the musical intuition that control inputs should be sparse and deliberate.

Both penalties serve to regularize the learned button interface, promote interpretable control patterns, and reduce over-reliance on dense activations. The selected penalty is weighted by a hyperparameter λ_{btn} and contributes to the overall training loss.

4.4.4 Total Loss

The full objective combines all terms:

$$\mathcal{L}_{\text{GrooveIQ}} = \mathcal{L}_{\text{recons}} + \lambda_{\text{kl}} \cdot \mathcal{L}_{\text{kl}} + \lambda_{\text{btn}} \cdot \mathcal{L}_{\text{btn}}$$

where λ_{recons}, λ_{kl}, and λ_{btn} are tunable coefficients that control the trade-off between reconstruction fidelity, latent regularization, and control simplicity.

4.5 Inference and Generation

Once trained, GrooveIQ enables drum generation conditioned on sparse button sequences. Inference proceeds by preparing two key inputs: a control embedding derived from the control sequence (eg. two-bar two-button 16th-note quantized hit sequences), and a latent sequence $\mathbf{z}_{1:T}$, which modulates stylistic variation.

4.5.1 Latent Conditioning Pathways

At inference time, *GrooveIQ* supports two modes of generating the latent sequence $\mathbf{z}_{1:T} \in \mathbb{R}^{T \times d_z}$, depending on the availability of a reference drum performance $\mathbf{y}$:

- **Posterior Sampling:** When the expressive drum performance $\mathbf{y}$ is available, it is encoded into contextual features $\mathbf{d}_{1:T}$ using the encoder. These features are combined with the button embedding sequence $\mathbf{b}_{\text{embed}}$ to produce chunk-level posterior parameters:

  $$\mu_t^{\text{post}}, \log \sigma_t^{2,\text{post}} = f_{\text{post}}(\text{ChunkAvg}(\mathbf{d}_t), \text{ChunkAvg}(\mathbf{b}_{\text{embed},t}))$$

  A latent vector is then sampled at each timestep:

  $$\mathbf{z}_t^{\text{post}} \sim \mathcal{N}(\mu_t^{\text{post}}, \sigma_t^{2,\text{post}})$$

  This mode enables high-fidelity reconstruction and stylization aligned with the reference performance.

- **Prior Sampling:** In the absence of a reference performance, the model samples from a learned prior network that depends only on the button embedding. An LSTM processes $\mathbf{b}_{\text{embed}}$, and the output is chunk-averaged and projected to obtain prior parameters:

  $$\mu_t^{\text{prior}}, \log \sigma_t^{2,\text{prior}} = f_{\text{prior}}(\text{ChunkAvg}(\text{LSTM}(\mathbf{b}_{\text{embed}})_t))$$

  A latent vector is then sampled from:

  $$\mathbf{z}_t^{\text{prior}} \sim \mathcal{N}(\mu_t^{\text{prior}}, \sigma_t^{2,\text{prior}})$$

  This enables fully generative inference based solely on the sparse button sequence.

4.5.2 Stylistic Variation via Latent Sampling

To explore expressive diversity, GrooveIQ leverages the stochastic nature of both the prior and posterior latent sequences. Specifically, stylistic variation can be introduced by manipulating:

- The reference performance $\mathbf{y}$ used by the encoder to derive posterior parameters.
- The random sample $\mathbf{z}_t^{\text{post}} \sim \mathcal{N}(\mu_t^{\text{post}}, \sigma_t^{2,\text{post}})$ drawn from the posterior.
- The random sample $\mathbf{z}_t^{\text{prior}} \sim \mathcal{N}(\mu_t^{\text{prior}}, \sigma_t^{2,\text{prior}})$ drawn from the prior (in the absence of $\mathbf{y}$).

These manipulations allow for diverse expressive realizations of the same control sketch $\mathbf{b}$. During inference, we demonstrate stylistic modulation by fixing the button sequence while varying either the reference performance or the stochastic latent sample. This approach facilitates both faithful reconstructions and imaginative generation from sparse inputs.

4.5.3 Full Generation Pipeline

At test time, generation proceeds as follows:

1. **Encode the control sequence:** Derive $\mathbf{b}_{\text{embed}}$ from either a learned or user-supplied button hit tensor $\mathbf{b} \in \{0,1\}^{T \times N}$.

2. **Obtain latent:** Use posterior or prior networks to generate $\mathbf{z}_{1:T}$.
3. **Generate output:** Pass $\mathbf{z}_{1:T}$ and $\mathbf{b}_{\text{embed}}$ to the decoder to produce an expressive drum sequence $\mathbf{y}_{\text{gen}} \in \mathbb{R}^{T \times E \times M}$.

Chapter 5

Experiments and Results

5.1 Experimental Setup

This section outlines the experimental framework used to train and evaluate the GrooveIQ model. All experiments use the train, validation, and test splits described in Section 3.2. Models are trained for 15 epochs, with checkpoints saved after each epoch. Training is conducted on a single NVIDIA V100 GPU with 32GB of RAM and a batch size of 512. Each epoch takes approximately 3–4 minutes in the representation learning setup and 11–13 minutes in the semi-supervised setup, primarily due to the non-parallelizable, on-the-fly heuristic derivation of control sequences. To preserve stochasticity, control sequences are not cached. After each epoch, validation is performed on 5,000 samples to monitor the evaluation metrics discussed in Section 5.1.3.

5.1.1 Model Setup

The GrooveIQ model follows a fixed configuration across all experiments.

We use a 2-layer axial Transformer encoder with 4 attention heads per layer and an embedding dimension of 128. The encoder output is chunked using a chunk size of 11, averaged within each chunk, and expanded across three temporal groups to create a compressed representation. This is combined with button embeddings to compute posterior parameters.

Two separate LSTM networks, each with a hidden size of 128, are used for the posterior and prior latent networks. The decoder is a 2-layer causal Transformer with 4 attention heads per layer and positional encoding which auto-regressively generates drum features from the concatenated button embedding and sampled latent vector at each timestep. All internal representations — including encoder outputs, latent variables, and button embeddings — operate in a shared embedding space of dimension 128. The model comprises approximately 6.8 million parameters, with around 5 million in the encoder and 1.8 million in the decoder.

5.1.2 Training Configuration

We outline the key hyperparameters used across all experiments which remained fixed throughout.

Optimization We use the AdamW optimizer with a base learning rate of 0.001 and a weight decay of 1×10^{-6}. Learning rate is annealed from 0.001 to 1×10^{-5} over 15 epochs using a cosine annealing schedule.

Hit Loss and Class Imbalance Handling Drum sequences are inherently sparse, with significantly more silent frames than active hits. To address this imbalance, we apply a weighted binary cross-entropy loss to the predicted hit probabilities $\hat{h}_{b,t,e} \in [0,1]$ and the ground-truth hits $h_{b,t,e} \in \{0,1\}$:

$$\mathcal{L}_{\text{hit}} = -\frac{1}{BTE} \sum_{b=1}^{B} \sum_{t=1}^{T} \sum_{e=1}^{E} \left[\alpha \cdot h_{b,t,e} \cdot \log(\hat{h}_{b,t,e}) + (1 - h_{b,t,e}) \cdot \log(1 - \hat{h}_{b,t,e}) \right]$$

Here, $\alpha = 15.0$ is a positive class weight chosen to counteract the class imbalance. A higher α penalizes false negatives more strongly, encouraging the model to avoid trivially predicting silence.

Hit-Conditioned Regression Losses Velocity and timing offset predictions are only meaningful when a hit is present. To enforce this, we define a hit penalty mask $m_{b,t,e}$:

$$m_{b,t,e} = \begin{cases} \gamma & \text{if } h_{b,t,e} = 1 \\ 0 & \text{otherwise} \end{cases} \quad \text{with } \gamma = 15.0$$

This mask is applied to the mean squared error (MSE) losses for velocity and timing offset:

$$\mathcal{L}_{\text{vel}} = \frac{1}{BTE} \sum_{b,t,e} m_{b,t,e} \cdot (\hat{v}_{b,t,e} - v_{b,t,e})^2$$

$$\mathcal{L}_{\text{off}} = \frac{1}{BTE} \sum_{b,t,e} m_{b,t,e} \cdot (\hat{o}_{b,t,e} - o_{b,t,e})^2$$

Total Reconstruction Loss The full reconstruction loss is computed as a weighted sum of all components:

$$\mathcal{L}_{\text{recons}} = \lambda_{\text{hit}} \cdot \mathcal{L}_{\text{hit}} + \lambda_{\text{vel}} \cdot \mathcal{L}_{\text{vel}} + \lambda_{\text{off}} \cdot \mathcal{L}_{\text{off}}$$

In practice, all coefficients $\lambda_{\text{hit}}, \lambda_{\text{vel}}, \lambda_{\text{off}}$ can be set to 1.0 unless otherwise tuned for a specific use case.

$$\mathcal{L}_{\text{recons}} = \text{BCE}_{\text{hit}} + \text{MSE}_{\text{vel}} \cdot \text{mask} + \text{MSE}_{\text{offset}} \cdot \text{mask}$$

Regularization Strategies

To improve generalization and ensure stable training, the following regularization techniques and scheduling strategies were employed:

KL Weight Annealing To prevent premature regularization of the latent space, the Kullback–Leibler divergence (KLD) term is gradually introduced using a linear schedule over the first 10 epochs. The KL weight λ_{KLD} increases linearly from 0 to 0.1:

$$\lambda_{\text{KLD}}(e) = \begin{cases} 0.1 \cdot \frac{e}{15} & \text{if } e \leq 10 \\ 0.1 & \text{otherwise} \end{cases}$$

This strategy encourages the model to first focus on reconstructing meaningful outputs before enforcing latent structure.

Dropout on Button Embeddings To promote robustness under sparse control inputs, dropout with a probability of 0.2 is applied to the learned button embeddings during training.

Scheduled Sampling As described in Section 4.3.5, we employ scheduled sampling during training with a fixed probability $p = 0.5$. This exposes the decoder to latent sequences from both the prior and posterior, to improve generalization at inference time when only the prior is available.

5.1.3 Evaluation Protocol

We evaluate GrooveIQ across multiple dimensions of reconstruction and control fidelity. The evaluation process consists of generating drum sequences from control inputs and comparing them against ground-truth performances using a suite of classification and regression metrics. Evaluation is conducted both during validation (to guide training) and post-training (for analysis and visualization).

Metrics

We report the following metrics, computed across the validation set:

- **Hit Accuracy (Acc)**: The fraction of timesteps and instruments where the predicted hit matches the ground truth (binary classification).
- **Hit Precision (PPV)**: Measures the proportion of predicted hits that are true hits.
- **Hit Recall (TPR)**: Measures the proportion of actual hits that are correctly predicted.
- **Hit F1-Score (F1)**: Harmonic mean of precision and recall.
- **Velocity MSE**: Mean squared error of predicted hit velocities, computed only at positions with ground-truth hits.
- **Offset MSE**: Mean squared error of expressive microtiming offsets, also computed only at hit positions.

Threshold Refinement

To improve hit prediction accuracy, we compute an optimal classification threshold for each validation epoch by sweeping over the hit probability outputs and computing the precision-recall curve. The threshold corresponding to the highest F1-score is selected and used for subsequent validation and test evaluations.

5.1.4 Ablation Setup

To evaluate the impact of different control sequence representations and architectural constraints, we conduct a series of ablations across two dimensions: (i) the type of control representation, and (ii) whether the button sequence is used causally during decoding. Table 5.1 summarizes the six model variants explored. All models are trained for 15 epochs using the configuration described in Section 5.1.2. We checkpoint at every epoch and select the best model based on a balanced tradeoff

TABLE 5.1: Configuration of ablation models across control type, regularization strategy, and causality

Model Name	Control Representation	Control Penalty	Button Causality
Model 1	Learned	L1	Causal
Model 2	Learned	L1	Non-Causal
Model 3	Learned	Group	Causal
Model 4	Learned	Group	Non-Causal
Model 5	Heuristic	N/A	Causal
Model 6	Heuristic	N/A	Non-Causal

between reconstruction performance (hit F1, velocity MSE, offset MST) and latent space regularization (KL loss).
Notes:

- **Control Representation:**
  - **Learned:** Control sequences are derived during training via backpropagation, allowing the model to discover optimal button activations.
  - **Heuristic:** Control sequences are derived using a rhythm-aware simplification of the input drum performance (see Section 4.3.2). Windows of length 2, 3, or 4 timesteps are sampled with probabilities 0.5, 0.25, and 0.25, respectively. Within each window, hits are considered only if their velocity exceeds a randomly sampled threshold between 45% and 80% of the sequence's maximum velocity. Up to 2 high-velocity hits are selected per window and mapped to control buttons. To mimic human-like variation, spurious hits are inserted with a 10% probability per window.

    For each window (randomly sampled from lengths of 2–4 timesteps with associated probabilities), high-velocity hits are selected if their intensity exceeds a dynamic threshold (45–80% of max velocity). The strongest hits within each window (up to 2 per window) are mapped to button categories, perturbed with small timing (± 0.05) and velocity ($\pm 5\%$) jitter, and retained with a 95% probability. Additionally, random spurious hits are introduced with a 10% chance per window to simulate human-like errors.
- **Control Penalty:** In the learned setting, we apply either an L1 or Group sparsity penalty to the control sequence, as detailed in Section 4.3.2, to encourage concise and structured button activations. No regularization is applied in the heuristic setting, since control sequences are externally computed and fixed.
- **Button Causality:**
  - **Causal:** At generation time, only past and current control inputs are accessible to the decoder, enforcing a left-to-right, real-time constraint.
  - **Non-Causal:** The decoder has access to the full control sequence, enabling globally optimized generation.

5.2 Results

We evaluate the models listed in 5.1 in both reconstruction and generation tasks, exploring their performance under different control conditions and latent configurations. Our primary goals are to assess how effectively the model captures expressive nuance from sparse input, how well it generalizes to unseen control sequences, and how rhythmically aligned the generated outputs are with respect to the intended control.

5.2.1 Quality of Predicted Hits, Velocities and Offsets

We start the evaluations by analyzing the distribution of hits, velocities, and timing offsets in the generated patterns obtained from our models. The analysis in this section was conducted on the test set of the GrooveIQ dataset, which was held out during both training and hyperparameter tuning.

To evaluate the fidelity of the generated drum sequences, we employ a set of metrics that capture both symbolic accuracy and expressive performance:

- **Offset MAE (Mean Absolute Error):** Measures the average absolute difference between predicted and ground-truth onset timing offsets for true positive hits. Lower values indicate better temporal precision.
- **Velocity MAE:** Computes the average absolute error between predicted and reference velocity values for correctly matched hits. It reflects how well the model preserves expressive dynamics.
- **TPR Hits (True Positive Rate):** The proportion of ground-truth hits that were successfully predicted. This measures recall and indicates the model's ability to capture rhythmic structure.
- **PPV Hits (Positive Predictive Value):** The proportion of predicted hits that match the ground truth. It reflects the model's precision and ability to avoid spurious activations.
- **Total Hits:** The total number of hits predicted in each sequence. Comparing this distribution with the ground truth helps assess if a model tends to under- or over-generate events.

These metrics jointly capture the quality of both the symbolic content and the expressive attributes of the generated patterns.

Total Hit Count Consistency

For every 2-bar pattern, the model must predict 297 events (33 time steps × 9 instruments), the majority of which are non-hits (i.e., silences). This results in a highly imbalanced prediction space, making it crucial for models to (1) maintain the correct proportion of hits to non-hits, and (2) accurately identify the locations of hit events. Figure 5.1 examines the distribution of total predicted hits compared to the ground truth. All models are centered closely around the ground truth distribution, with minor underprediction tendencies.

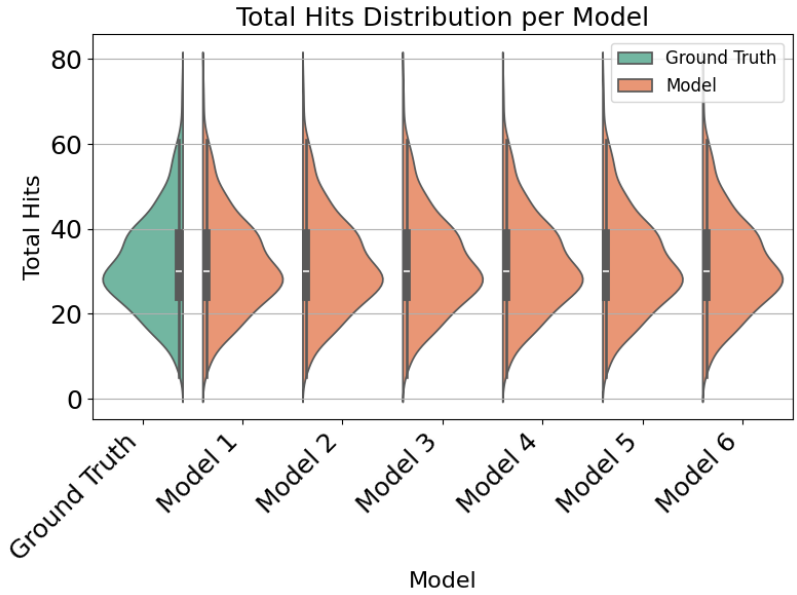

FIGURE 5.1: Distribution of total predicted hits per model, compared with ground truth

Hit Prediction Accuracy

Figure 5.2 shows the distribution of True Positive Rate (TPR) for hit predictions. All models perform well, with median values close to or exceeding 0.95. Notably, the heuristic-based models (Model 5 and 6) exhibit the tightest distributions and maintain consistently high recall, underscoring the effectiveness of rhythmically informed control sequences in preserving essential onsets..

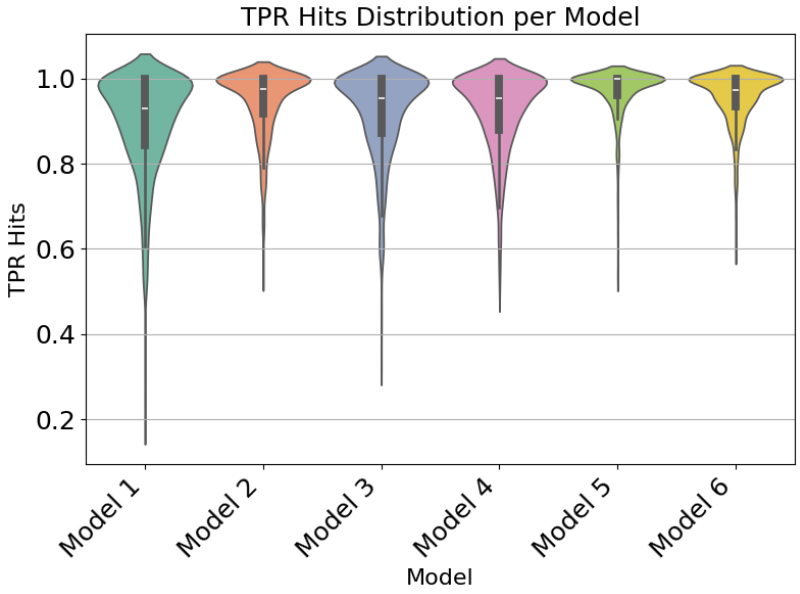

FIGURE 5.2: TPR (True Positive Rate) distribution per model

Velocity Accuracy

As shown in Figure 5.3, the velocity MAE distributions follow a similar trend. Causal models—particularly Model 1 and Model 5 outperform their non-causal counterparts, with lower mean errors and tighter interquartile ranges. Learned control models with Group penalty (Model 3 and 4) perform slightly worse, indicating that group-level sparsity may not align as well with the local precision required for expressive dynamics. Model 6 shows the widest spread and highest outliers.

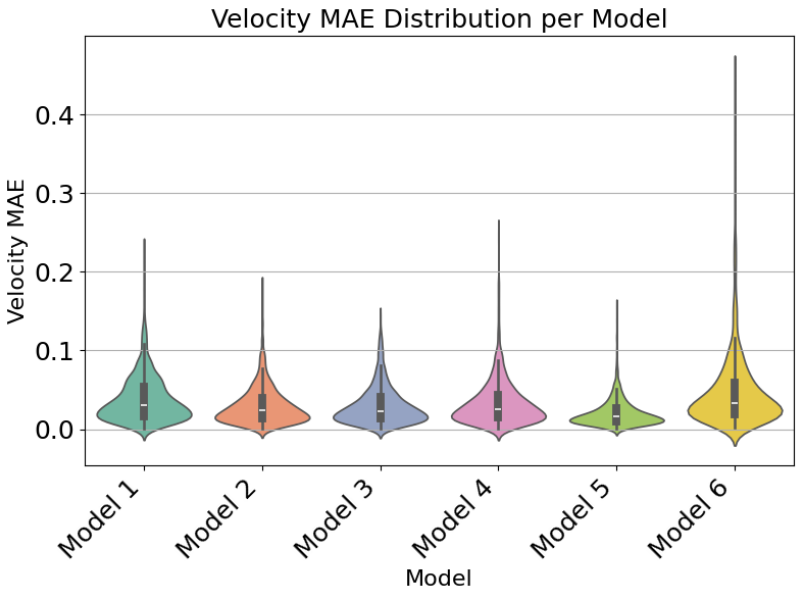

FIGURE 5.3: Velocity MAE distribution per model

Offset Accuracy

Figure 5.4 shows the distribution of Mean Absolute Error (MAE) for timing offsets across all models. All six models achieve low median offset errors, generally below 0.01. Notably, Model 1 (learned, L1, causal) and Model 5 (heuristic, causal) demonstrate the tightest offset error distributions, suggesting that causal conditioning, regardless of control source, contributes to more precise onset alignment. Model 6 exhibits a heavier tail, indicating higher variance in offset predictions when using heuristic, non-causal control.

5.2.2 Relative Analysis

In this section, we adopt one of the objective evaluation methods for generative music models proposed by [26]. Specifically, we assess the ability of our models to preserve the statistical characteristics of real drum performances by comparing feature distributions between generated and ground-truth sequences.

For this analysis, we sampled 1000 random examples from the test split of the GrooveIQ dataset. Corresponding to each target drum pattern, we generated a sample from each model by passing a fixed, heuristically derived control sequence through the model's decoder. We then extracted the same set of structural features used in our dataset analysis (see Section 3.3) from both the ground-truth and generated sequences.

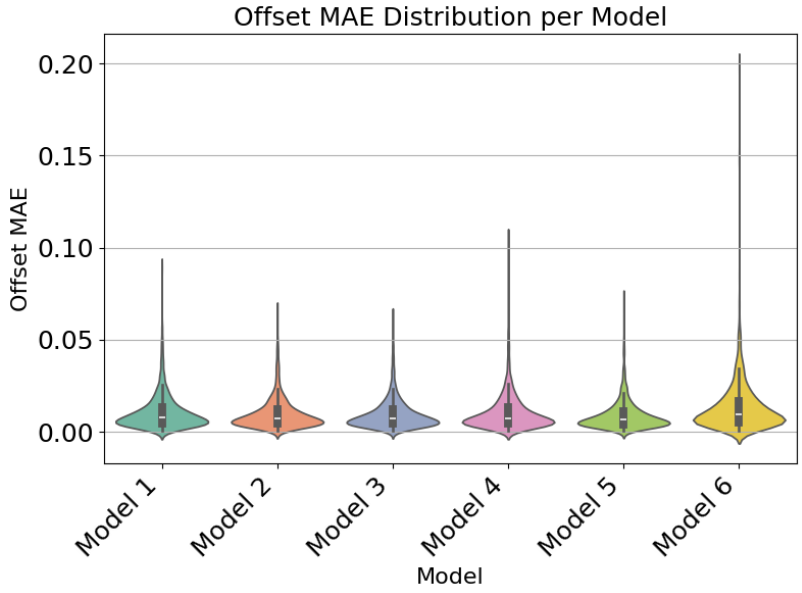

FIGURE 5.4: Offset MAE distribution per model

To quantify the similarity between these distributions, we computed two sets of pairwise distances:

1. **Intra-set distances:** Pairwise distances between feature vectors within the same set (i.e., within ground-truth or generated samples).

2. **Inter-set distances:** Pairwise distances between each generated sample and all samples in the ground-truth set.

Following [26], we estimated the probability density function (PDF) of these distance arrays using Gaussian kernel density estimation with Scott's rule for bandwidth selection [38, 39]. We then measured the divergence between distributions using two complementary metrics:

- **Kullback–Leibler (KL) Divergence:** Quantifies the dissimilarity between the inter-set and intra-set distributions. Lower values indicate that the generated samples better match the structural diversity of the reference set.

- **Overlapping Area (OA):** Measures the integral overlap between the two PDFs. Higher OA indicates greater statistical similarity and thus better generalization.

As illustrated in Figure 5.5, plotting KL divergence against OA for multiple models allows for direct visual comparison. Points closer to the upper-left corner (i.e., low KL and high OA) reflect stronger alignment with the data distribution, and are thus preferred.

Across most features, the top-performing models cluster tightly near the upper-left corner, indicating strong alignment with the feature distribution of the real data. Specifically, Models 1, 3, 4, and 5 consistently exhibit low KL divergence and high OA across features such as *Total Hits*, *Total Density*, *Total Complexity*, *Average Intensity*, and *Combined Syncopation*. This suggests that both learned and heuristic control mechanisms, when combined with causal conditioning or group sparsity, can support generalization to real-world expressive characteristics.

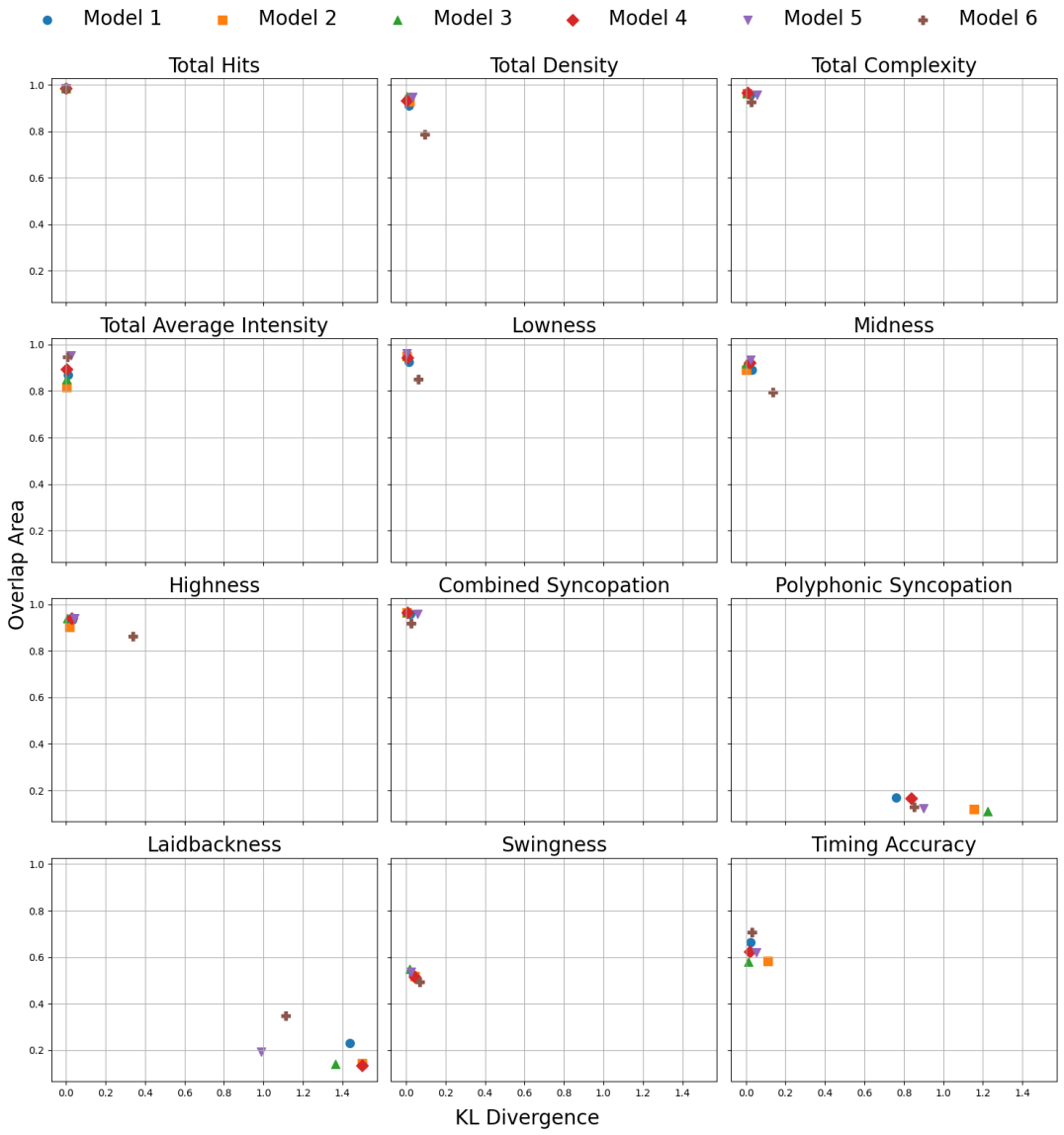

FIGURE 5.5: KL Divergence vs. Overlapping Area per feature across models

Model 6, which uses heuristic control and non-causal decoding, generally performs worse across the board. It shows higher KL divergence and lower OA in multiple features including *Lowness*, *Highness*, and *Intensity*, indicating that relaxing temporal causality compromises the model's ability to preserve core rhythmic and timbral distributions. Model 2 similarly shows elevated KL and reduced OA in key features like *Swingness* and *Laidbackness*, suggesting that the absence of causal structure harms temporal feel and nuance.

Among all features, *Swingness*, *Laidbackness*, and *Polyphonic Syncopation* stand out as the most challenging to reproduce accurately. All models—including the strongest performers—show elevated KL divergence and wide separation from ground-truth intra-set baselines. This indicates that expressive timing and multi-voice rhythmic coordination are particularly difficult to learn or reproduce given the current modeling setups.

Conversely, features like *Total Density*, *Combined Syncopation*, and *Total Complexity* are well-modeled by nearly all systems, with minimal inter-set divergence and tightly overlapping distributions. These global or structural features may be easier to match given their correlation with the control input.

Overall, these results suggest that the GrooveIQ variants can successfully learn and generalize many core statistical properties of human drum performances, particularly when causal decoding and sparsity-inducing control regularization are employed. However, certain expressive features—particularly those governing temporal nuance—remain challenging and represent promising directions for future model design and control formulation.

5.2.3 Control Alignment

To evaluate how rhythmically aligned the generated output is with respect to the control sequence, we assess the relationship between control inputs and model predictions using two alignment metrics: hit density correlation and temporal cross-correlation lag.

- **Hit Density Correlation** This metric captures the statistical correspondence between the number of active control inputs and the number of drum hits generated over time. For each sequence, we compute a control activation vector by summing the binary button hits at each timestep, and compare it to the total predicted hit density obtained by summing the hit activations across all drum instruments. The Pearson correlation coefficient [40] between these two sequences is then computed on a per-example basis. A higher correlation implies that the model responds to increased control activity with denser rhythmic output, indicating effective use of control information.

- **Temporal Cross-Correlation Lag** While hit correlation quantifies strength of association, temporal cross-correlation lag estimates when the output responds relative to the control. For each sample, we compute the lag at which the cross-correlation [41] between control activation and predicted hit density reaches its maximum. A lag of zero indicates that the model's rhythmic output is temporally aligned with the control, while positive or negative lags imply delayed or anticipatory behavior. We report the distribution of these lag values across all test examples.

Figure 5.6 shows the distribution of per-example hit correlation values for each model. Models 1 and 3 achieve the highest mean correlations, indicating strong

alignment between control activation and output density, even under unseen control sequences. Model 4 also performs competitively, followed closely by Model 5, which uses a heuristic control path. The results demonstrate that learned control mechanisms, when trained with sufficient regularization and evaluated on held-out sequences, are capable of producing consistent and meaningful rhythmic control behavior.

Model 6 shows the weakest alignment, with a mean correlation of only 0.10 and significantly lower spread. This might suggest that non-causal control relationship combined with heuristic control undermines rhythmic responsiveness. Model 2, which also lacks causal conditioning, performs moderately but shows a wider lag distribution and lower mean correlation than its causal counterparts.

Figure 5.7 plots the distribution of cross-correlation lags. The best-performing models exhibit lag distributions clustered near zero. Specifically, Model 5 achieves the smallest average lag (-0.53 timesteps), followed closely by Models 1 and 3 (both within -2 timesteps on average). These results might indicate that both learned and heuristic control sequences can lead to temporally aligned generation, provided that a causal relationship is enforced between the control input and the decoder. The variance of lag values remains low for causal models, reinforcing the benefit of unidirectional conditioning in producing stable rhythmic outputs.

By contrast, Model 6 exhibits the largest average lag (-7.85 timesteps) and the widest variance, indicating highly delayed and inconsistent responses to the control input. Model 2 also shows degraded temporal alignment, with a substantial average lag of -4.47 timesteps.

The alignment results might emphasize the importance of both control formulation and architectural design. In particular, models employing *causal decoding* (Models 1, 3, 4, and 5) consistently show stronger alignment than their non-causal counterparts. The marginal improvement of Model 5 over Models 1 and 3 in terms of lag variance suggests that heuristic control, while less flexible, may provide a more stable mapping between input and output timing.

Crucially, the learned control models (especially Models 1 and 3) now demonstrate comparable or superior performance to the heuristic baseline, even when evaluated on unseen control sequences. This supports the claim that learned control pathways can generalize effectively when regularized appropriately, and that such mechanisms can be used for generation or stylistic manipulation.

Overall, these findings suggest that the combination of causal decoding and structured control — whether learned or heuristic — plays a central role in ensuring reliable, responsive, and musically interpretable drum generation.

5.2.4 Qualitative Results

While more work is needed to fully understand the expressive control and real-world usability of the model—particularly through qualitative research with musicians and producers—this section presents an initial informal evaluation based on visual inspection and listening tests. **Model 5** combines causal decoding with a rhythmically informed heuristic control path to achieve the best balance of rhythmic alignment, interpretability, and expressive fidelity but quantitatively and based on informal listening tests. It responds to control input in a stable and temporally coherent manner, generalizes well across diverse rhythmic contexts, and is readily adaptable to real-time or user-in-the-loop generation tasks, making it the preferred final model for further evaluation in this section.

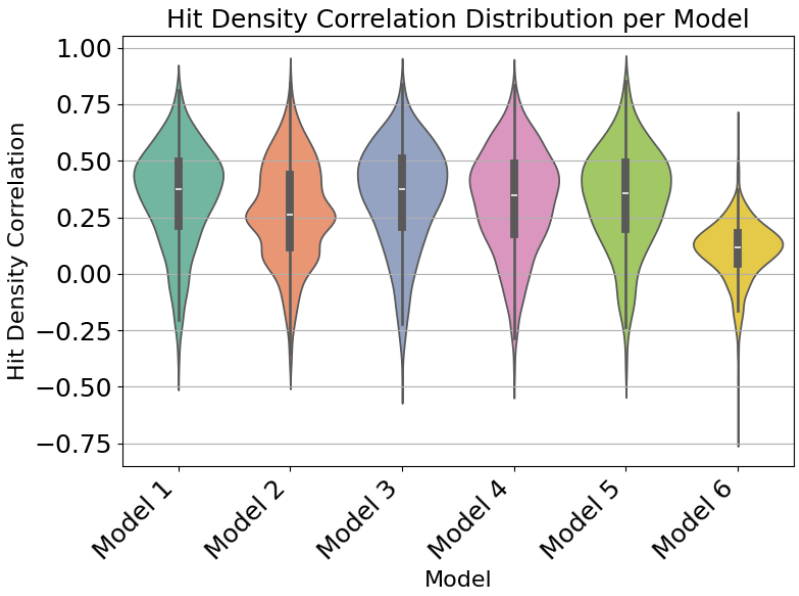

FIGURE 5.6: Distribution of hit density correlation between control and output per model

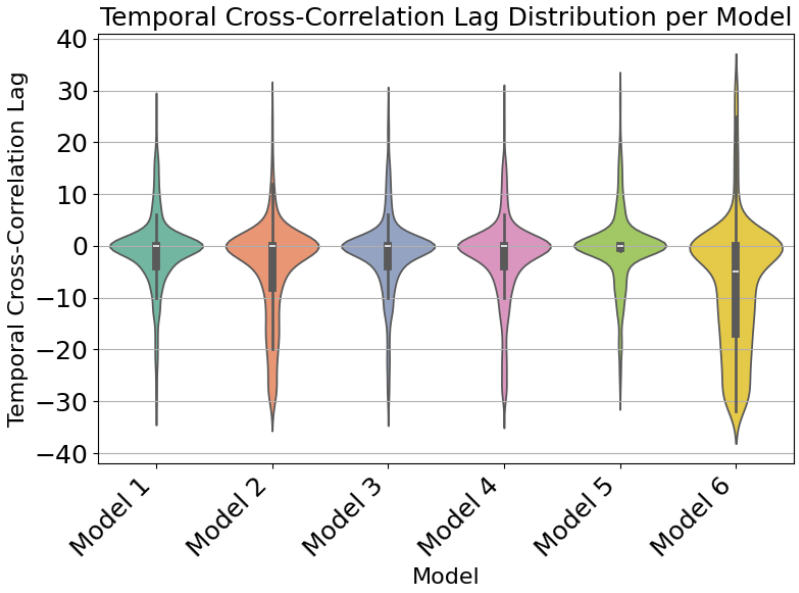

FIGURE 5.7: Distribution of cross-correlation lag between control and output per model

Experiment 1: Masking Control Inputs

In our first experiment, we simulate varying levels of user input by progressively masking different portions of the control sequence and observing how this affects the generated drum pattern. For this experiment, we sample from the *prior network*, allowing the model to infer a plausible groove given sparse or incomplete control. The masking follows a beat-wise progression:

We define five masking configurations applied per-bar, each retaining a different subset of beats to simulate varying degrees of control sparsity:

1. **No Masking:** All four beats are retained, corresponding to the full control sequence.
2. **Strong Beats Only:** Beats 1 and 3 (the downbeats) are preserved, while beats 2 and 4 are masked.
3. **Off-Beats Only:** Only beats 2 and 4 are retained, omitting the strong beats.
4. **Downbeat + Upbeat:** Beats 1 and 2 are kept, simulating the beginning of a phrase or partial input.
5. **Downbeat Only:** Only the first beat is preserved, representing the most minimal input scenario.

Figure 5.8 visualizes this process. The left column shows the control sequence, while the right column shows the generated fixed-grid drum performance. As expected, the full control input yields a complete, well-articulated groove with consistent timing and instrument usage. As we progressively reduce the control information, the generated sequence becomes sparser and simpler, but maintains musical coherence. Notably, the model fills in plausible intermediate hits even when only downbeats are provided (bottom row), suggesting that the prior has learned meaningful rhythmic priors conditioned on minimal cues. The model also maintains strong alignment in the kick and snare layers, even under extreme masking, demonstrating its capacity for temporal extrapolation.

Experiment 2: Groove Style Transfer via Posterior Sampling

In a second experiment, we explore the model's ability to transfer stylistic information by sampling from the *posterior network* using different reference drum grooves, while keeping the control sequence fixed. This simulates a form of groove-conditioned style transfer, where the high-level structure is fixed by the control input, but the expressive surface (timing, instrumentation, intensity) is modulated by the posterior sample.

Figure 5.9 shows the result of sampling five distinct grooves using the same control sequence but conditioning on reference grooves drawn from different musical styles. Despite having identical control inputs, the outputs exhibit clear stylistic variation, particularly in the cymbal patterns, velocity dynamics, and degree of syncopation. For example, some samples introduce denser hi-hat work or delayed snare placements, evoking funkier or jazz-like feels, while others remain minimal and on-grid. Importantly, low and mid-frequency instruments such as kick and snare retain strong alignment to the control inputs, a consequence of the heuristic simplification scheme used during training.

Informal listening tests support the visual findings. The generated patterns adhere to the control sequence in both timing and structure, particularly in the kick

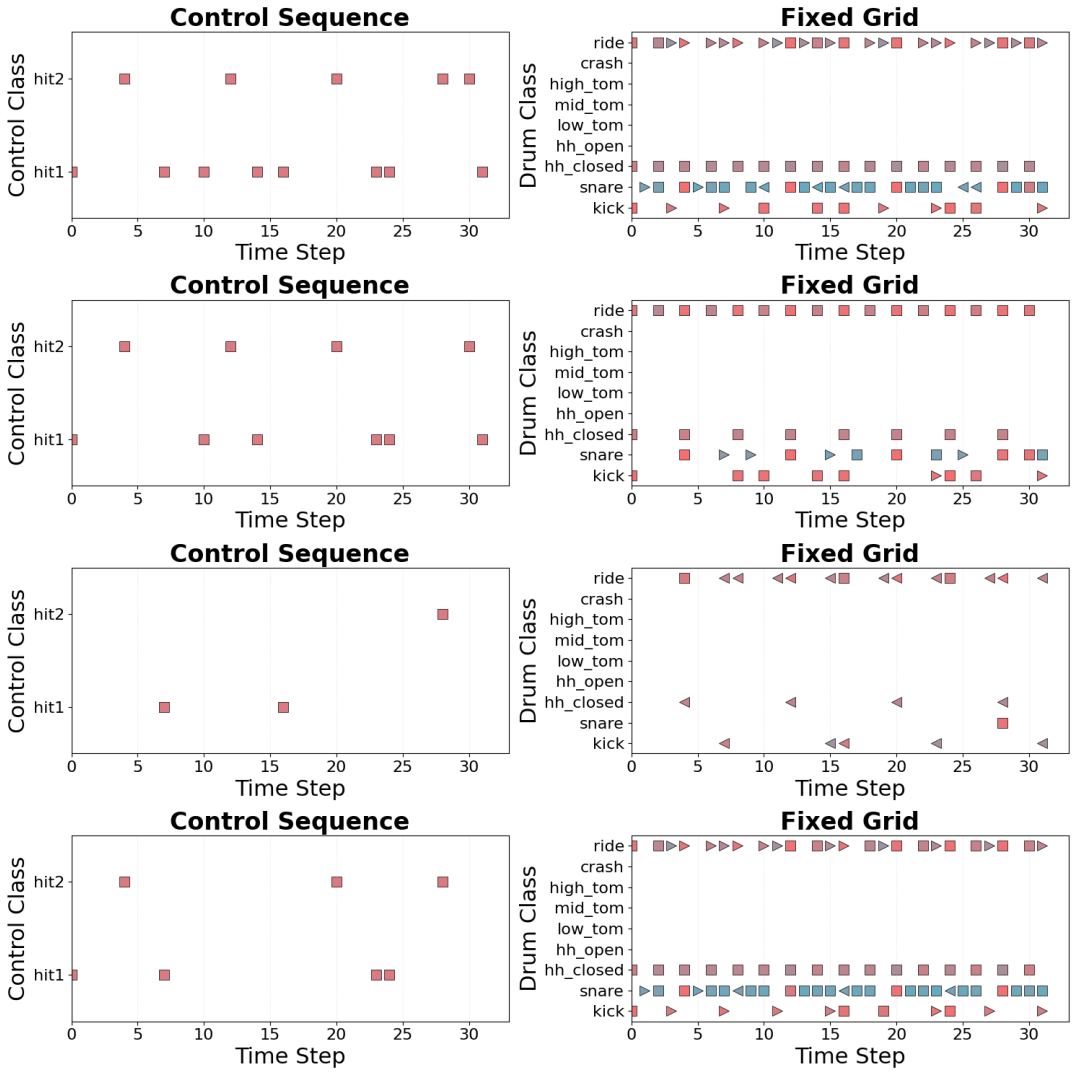

FIGURE 5.8: Left: control sequences with decreasing coverage over 4-beat patterns. Right: corresponding model outputs sampled from the prior. The model maintains coherent structure even with minimal input, filling in missing elements based on rhythmic priors.

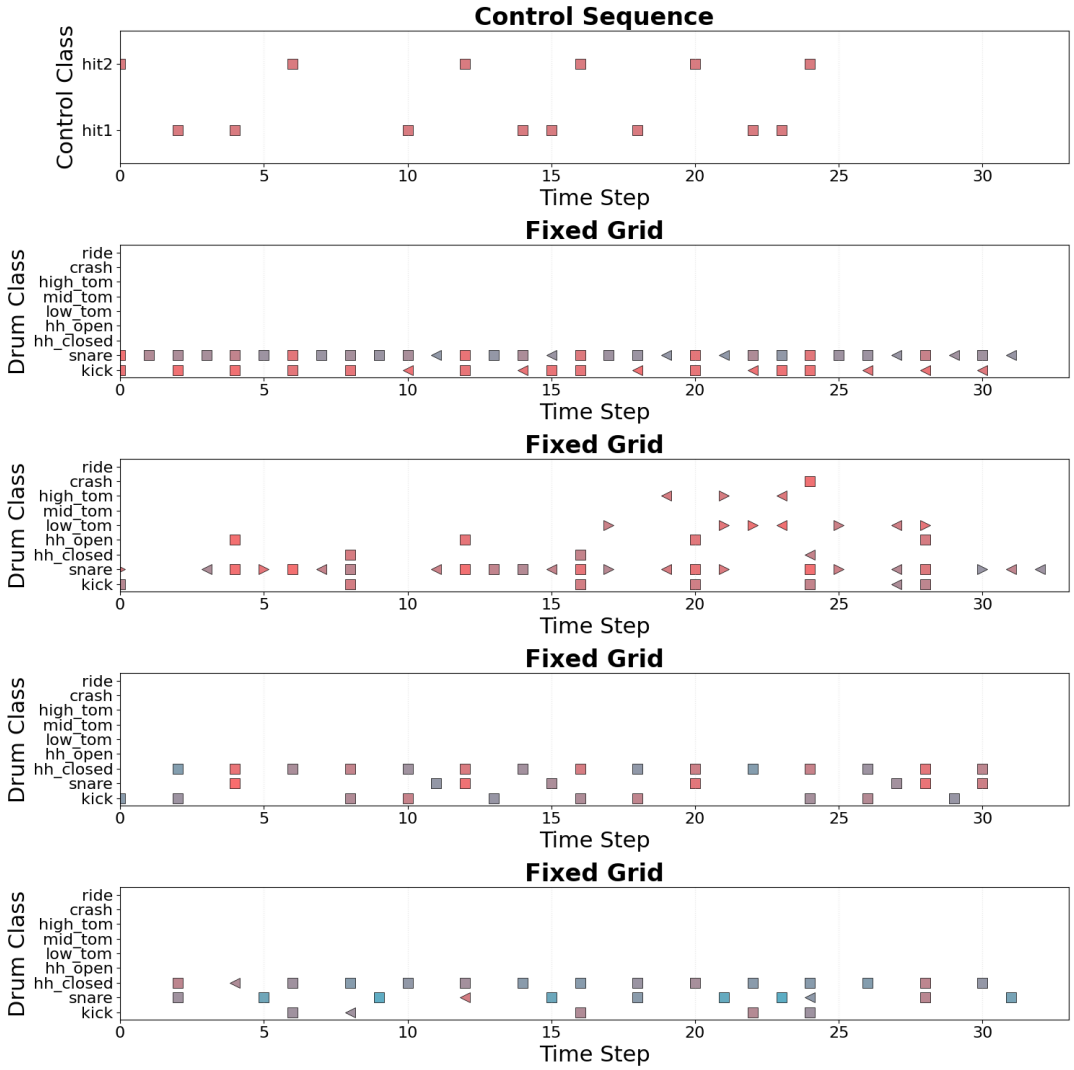

FIGURE 5.9: All rows use the same 2-button control input (top-row), but different posterior latents inferred from expressive drum performances. The generated outputs (bottom four rows) exhibit stylistic diversity in dynamics, timing, and instrument usage, while respecting the control structure in the low and mid-range drum classes.

and snare lines. Sampling from the prior yields stylistically conservative but coherent grooves, while posterior sampling introduces expressive variability aligned with the character of the reference groove. This suggests that the model can serve both as a reliable sketch interpreter and a generative style transfer system.

Evaluating groove transfer remains a challenging and largely open problem, particularly due to the lack of standardized benchmarks or evaluation protocols. While the results in this section highlight the creative potential of this modeling framework, they also raise important questions about how to systematically assess style, feel, and user control in generative rhythm systems. These findings suggest a range of possible directions for future work, including deeper investigation into user interaction paradigms, integration into live performance and production workflows, and the development of more nuanced metrics for groove quality and stylistic variation. We believe that this line of research holds promise not only for technical advancement but also for new forms of musical cocreation.

Chapter 6

Conclusion and Future Work

6.1 Summary of Contributions

In this work, we demonstrate that both learned and heuristically derived sparse control sequences can serve as effective rhythmic anchors for controllable drum generation. We introduce the GrooveIQ MIDI dataset, which contains an order of magnitude more drum performances than the widely used Groove MIDI Dataset (GMD), and offer a comparative analysis of its scale, stylistic breadth, and loop-based structure in contrast to GMD's more expressive and human-performed recordings. We propose a representation learning and semi-supervised framework for training a drum generative model that translates sparse, intuitive input—such as simple button presses—into detailed and expressive drum performances, potentially enabling novice users to create complex rhythms with minimal effort. Finally, we informally showcase the model's capacity for style transfer via posterior sampling, illustrating its potential for a range of creative and interactive musical applications.

6.2 Future Directions

6.2.1 Improving Posterior Sampling with Learned Controllers

Although our current approach uses fixed posterior sampling conditioned on existing drum grooves, future work could explore learning more expressive, controllable latent priors or posterior "controllers" directly. Methods such as mixture-of-experts or flow-based priors can enable richer style control while preserving alignment to the control sequence. Such approaches would better balance generativity and control in posterior sampling, possibly enabling style interpolation or attribute-specific sampling (e.g. rhythmic complexity or energy).

6.2.2 Exploring Alternative Feature Representations

While our current expressive features: hit presence, velocity, and timing offset, provide meaningful control signals, they are limited in their ability to capture higher-order rhythmic properties such as syncopation, groove density, and inter-voice relationships. To better model microtiming and expressive nuance, more flexible representations may be needed. For example, the *Flexible-Grid* representation discussed in 3.4.2 could provide a more adaptable encoding of temporal structure across instruments.

Additionally, recent work on perceptually informed rhythm complexity measures for polyphonic drums [42, 43, 44] may offer valuable insights for designing richer latent spaces. Incorporating features such as rhythmic entropy, polyphonic

syncopation, and instrument-specific timing variance could lead to improvements in both generative quality and interpretability.

These ideas may also be fruitfully combined with advances in token-based modeling. Modern symbolic music tokenizers such as REMI, REMI+, Compound Word, and Compressive Compound Word, encode musical events as discrete tokens with rich meta-information including bar position, instrument class, pitch, velocity, and temporal offset [45, 46]. Used in models like Museformer and MLAT, these token-based approaches reduce sequence length while preserving structural detail, offering a scalable and expressive alternative to fixed-grid representations.

Integrating rhythm-informed feature encodings with token-based sequence modeling may thus support more structured generation, long-range coherence, and interpretable control over expressive and stylistic attributes in groove generation tasks.

6.2.3 Disentangling Latent Factors of Variation

While our model uses a shared latent sequence, it currently does not explicitly disentangle stylistic, rhythmic, or structural factors. Disentangled representation learning has been shown to improve controllability in music generation [47], though disentanglement alone does not guarantee controllability. Future research could investigate supervised or self-supervised disentanglement (e.g. style, instrument density, rhythmic feel) and evaluate whether semantically meaningful latent subspaces enable more precise style transfer or groove manipulation.

6.2.4 Building a Real-Time Interactive Interface

To support live usage and integration into musical workflows, developing a real-time interface is crucial. Prior systems such as the Neural Drum Machine with Max for Live integration demonstrate the feasibility of lightweight, interactive generation models in studio settings [48]. We plan to prototype a live controller interface (e.g. two-button device, MIDI, or gestural input via platforms like ChucK or SuperCollider) that connects to the GrooveIQ model, allowing users to tap sparse rhythmic cues and receive immediate drum generation. This aligns with recent discussions on interactive music generation systems and HCI design for creative AI tools [49].

6.2.5 Towards User Studies and Evaluation Benchmarks

Finally, while we have conducted informal listening and visual evaluations, there remains a broader need to systematically assess groove quality, user affordance, and creative efficacy. Future work should include user studies with musicians and producers, standardized evaluation frameworks (e.g. subjective rating of “feel” or groove), and possibly benchmark tasks comparing against systems likeGrooVAE. Establishing community benchmarks would enable more consistent comparison across generative rhythm models and foster adoption.

Bibliography

[1] Carolyn Drake and Daniel Bertrand. "The quest for universals in temporal processing in music". In: *Annals of the New York Academy of Sciences* 930.1 (2001), pp. 17–27.

[2] Olivier Senn et al. "Groove in drum patterns as a function of both rhythmic properties and listeners' attitudes". In: *PLOS ONE* 13 (June 2018), e0199604. DOI: 10.1371/journal.pone.0199604.

[3] Peter Knees et al. "The GiantSteps Project: A Second-Year Intermediate Report". In: (Jan. 2016).

[4] Matthew Wright and Edgar Berdahl. "Towards Machine Learning of Expressive Microtiming in Brazilian Drumming". In: *International Computer Music Conference, ICMC 2006* (Jan. 2006).

[5] Axel Tidemann and Yiannis Demiris. "Imitating the groove: Making drum machines more human". In: (Jan. 2007).

[6] Yu-Siang Huang and yi-hsuan Yang. "Pop Music Transformer: Beat-based Modeling and Generation of Expressive Pop Piano Compositions". In: Oct. 2020, pp. 1180–1188. DOI: 10.1145/3394171.3413671.

[7] Hao-Wen Dong et al. "MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32 (Apr. 2018). DOI: 10.1609/aaai.v32i1.11312.

[8] Li-Chia Yang, Szu-Yu Chou, and yi-hsuan Yang. "MidiNet: A Convolutional Generative Adversarial Network for Symbolic-domain Music Generation using 1D and 2D Conditions". In: (Mar. 2017). DOI: 10.48550/arXiv.1703.10847.

[9] Cheng-Zhi Anna Huang et al. *Music Transformer*. 2018. arXiv: 1809.04281 [cs.LG]. URL: https://arxiv.org/abs/1809.04281.

[10] Douglas Eck and Jürgen Schmidhuber. "Finding temporal structure in music: Blues improvisation with LSTM recurrent networks". In: *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*. IEEE. 2002, pp. 747–756.

[11] Chris Donahue, Ian Simon, and Sander Dieleman. "Piano Genie". In: Mar. 2019, pp. 160–164. DOI: 10.1145/3301275.3302288.

[12] Ian Simon and Sageev Oore. *Performance RNN: Generating Music with Expressive Timing and Dynamics*. https://magenta.withgoogle.com/performance-rnn. Blog. 2017.

[13] Diederik P Kingma and Max Welling. "Auto-encoding variational Bayes". In: *arXiv preprint arXiv:1312.6114* (2014).

[14] Adam Roberts et al. *A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music*. 2019. arXiv: 1803.05428 [cs.LG]. URL: https://arxiv.org/abs/1803.05428.

[15] Jon Gillick et al. *Learning to Groove with Inverse Sequence Transformations*. 2019. arXiv: 1905.06118 [cs.SD]. URL: https://arxiv.org/abs/1905.06118.

[16] Kyungyun Lee, Wonil Kim, and Juhan Nam. *PocketVAE: A Two-step Model for Groove Generation and Control*. 2021. arXiv: 2107.05009 [cs.SD]. URL: https://arxiv.org/abs/2107.05009.

[17] Ashish Vaswani et al. "Attention is all you need". In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017.

[18] Zihang Dai et al. "Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics. 2019, pp. 2978–2988.

[19] OpenAI. *MuseNet*. https://openai.com/index/musenet/. Accessed: 2025-07-25. 2019.

[20] Jeff Ens and Philippe Pasquier. *MMM : Exploring Conditional Multi-Track Music Generation with the Transformer*. Aug. 2020. DOI: 10.48550/arXiv.2008.06048.

[21] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in Neural Information Processing Systems*. Vol. 27. 2014.

[22] Victor Shepardson, Jack Armitage, and Thor Magnusson. "Notochord: a Flexible Probabilistic Model for Embodied MIDI Performance". In: *Proceedings of the 3rd Conference on AI Music Creativity*. AIMC, Sept. 2022. DOI: 10.5281/zenodo.7088404. URL: https://doi.org/10.5281/zenodo.7088404.

[23] Apple Inc. *Logic Pro Takes Music Making to the Next Level With New AI Features*. Accessed: 2025-08-01. 2024. URL: https://www.apple.com/newsroom/2024/05/logic-pro-takes-music-making-to-the-next-level-with-new-ai-features/.

[24] Colin Raffel. "Learning-based methods for comparing sequences, with applications to audio-to-MIDI alignment and matching". PhD thesis. Columbia University, 2016. URL: https://colinraffel.com/publications/raffel2016phd.pdf.

[25] Olivier Gillet and Gaël Richard. "ENST-Drums: An extensive audio-visual database for drum signals processing". In: *Proceedings of the 7th International Conference on Music Information Retrieval (ISMIR)*. 2006.

[26] Li-Chia Yang and Alexander Lerch. "On the evaluation of generative models in music". In: *Neural Computing and Applications* 32 (May 2020). DOI: 10.1007/s00521-018-3849-7.

[27] Jan Retkowski, Jakub Stępniak, and Mateusz Modrzejewski. *Frechet Music Distance: A Metric For Generative Symbolic Music Evaluation*. 2025. arXiv: 2412.07948 [cs.SD]. URL: https://arxiv.org/abs/2412.07948.

[28] *Music Genre List*. https://www.musicgenreslist.com. [Accessed 24-07-2025].

[29] David Gómez-Marín, Sergi Jordà, and Perfecto Herrera. "Drum rhythm spaces: from polyphonic similarity to generative maps". In: *Journal of New Music Research* 49.5 (2020), pp. 438–456. DOI: 10.1080/09298215.2020.1806887.

[30] Maria A. G. Witek et al. "Syncopation, Body-Movement and Pleasure in Groove Music". In: *PLOS ONE* 9.4 (Apr. 2014), pp. 1–12. DOI: 10.1371/journal.pone.0094446. URL: https://doi.org/10.1371/journal.pone.0094446.

[31] Fred Bruford et al. "Multidimensional similarity modelling of complex drum loops using the GrooveToolbox". In: *Proceedings of the 21th International Society for Music Information Retrieval Conference, ISMIR 2020, Montreal, Canada, October 11-16, 2020*. Ed. by Julie Cumming et al. 2020, pp. 263–270. URL: http://archives.ismir.net/ismir2020/paper/000211.pdf.

[32] Jon Gillick et al. "Drumroll Please: Modeling Multi-Scale Rhythmic Gestures with Flexible Grids". In: *Transactions of the International Society for Music Information Retrieval* (2021). DOI: 10.5334/tismir.98.

[33] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. "Estimating or propagating gradients through stochastic neurons for conditional computation". In: *arXiv preprint arXiv:1308.3432*. 2013.

[34] Ming Yuan and Yi Lin. "Model selection and estimation in regression with grouped variables". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68.1 (2006), pp. 49–67.

[35] Jonathan Ho et al. "Axial attention in multidimensional transformers". In: *arXiv preprint arXiv:1912.12180*. 2019.

[36] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. "Stochastic backpropagation and approximate inference in deep generative models". In: *Proceedings of the 31st International Conference on Machine Learning*. 2014.

[37] Samy Bengio et al. "Scheduled sampling for sequence prediction with recurrent neural networks". In: *Advances in neural information processing systems*. Vol. 28. 2015.

[38] David W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. New York: John Wiley & Sons, 1992.

[39] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Monographs on Statistics and Applied Probability. London: Chapman and Hall, 1986.

[40] David J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. 3rd. Chapman and Hall/CRC, 2003. ISBN: 9781584884408.

[41] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*. 3rd. Pearson Education, 2010. ISBN: 9780131988422.

[42] G. Percival and G. Tzanetakis. "Measuring the rhythmic complexity of musical percussion patterns". In: *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*. 2014, pp. 141–146.

[43] George Sioros et al. "Syncopation matters: Groove perception in music and its dependence on structure". In: *Frontiers in Psychology* 4 (2013), pp. 1–14.

[44] Fabien Gouyon, Perfecto Herrera, and Pedro Cano. "A computational approach to rhythm description: Audio features for the computation of rhythmic content descriptors". In: *AES Convention 118*. 2005.

[45] Botao Yu et al. "Museformer: Transformer with Fine- and Coarse-Grained Attention for Music Generation". In: *arXiv preprint arXiv:2210.10349*. 2022.

[46] Lianyu Zhou et al. "MLAT: a multi-level attention transformer capturing multi-level information in compound word encoding for symbolic music generation". In: *EURASIP Journal on Audio, Speech and Music Processing* 2025 (2025), p. 17.

[47] Francesco Locatello et al. *Disentangling Factors of Variation Using Few Labels*. 2020. arXiv: 1905.01258 [cs.LG]. URL: https://arxiv.org/abs/1905.01258.

[48] Moe Bretan and Gil Weinberg. “Neural Drum Machine: An Interactive System for Real-time Synthesis of Drums Using Neural Networks”. In: *Proceedings of the 2019 International Conference on New Interfaces for Musical Expression (NIME)* (2019).

[49] Craig Vear et al. “Human-AI Musicking: A Framework for Designing AI for Music Co-creativity”. In: Aug. 2023.