Exploring Procedural Data Generation for Automatic Acoustic Guitar Fingerpicking Transcription

Sebastian Murgul
Klangio GmbH
Karlsruhe, Germany
sebastian.murgul@klang.io

Michael Heizmann
Karlsruhe Institute of Technology
Karlsruhe, Germany
michael.heizmann@kit.edu

Abstract

Automatic transcription of acoustic guitar fingerpicking performances remains a challenging task due to the scarcity of labeled training data and legal constraints connected with musical recordings. This work investigates a procedural data generation pipeline as an alternative to real audio recordings for training transcription models. Our approach synthesizes training data through four stages: knowledge-based fingerpicking tablature composition, MIDI performance rendering, physical modeling using an extended Karplus-Strong algorithm, and audio augmentation including reverb and distortion. We train and evaluate a CRNN-based note-tracking model on both real and synthetic datasets, demonstrating that procedural data can be used to achieve reasonable note-tracking results. Finetuning with a small amount of real data further enhances transcription accuracy, improving over models trained exclusively on real recordings. These results highlight the potential of procedurally generated audio for data-scarce music information retrieval tasks.

1 Introduction

Automatic guitar transcription is a longstanding challenge in the field of Music Information Retrieval (MIR), particularly for polyphonic and expressive styles such as fingerpicking. Fingerpicking is a widely used guitar technique in which strings are plucked individually using the fingers or a plectrum, resulting in complex rhythmic and harmonic textures. Transcribing such performances into symbolic representations like tablature or MIDI remains difficult due to both musical and practical constraints.

While substantial progress has been made in piano transcription using deep learning methods, automatic guitar transcription has received comparatively less attention. One major limitation is the scarcity of high-quality annotated data. Xi et al. (2018) introduced *GuitarSet*, one of the few datasets with detailed note-level annotations for real guitar recordings. It includes 360 performances across various genres, tempi, and styles, recorded using a hexaphonic pickup system. However, the dataset is relatively small and lacks annotations for expressive playing techniques such as slides, bends, or hammer-ons. Wiggins et al. used GuitarSet to train a Convolutional Neural Network (CNN) to transcribe the audio into string-wise MIDI representations (Wiggins and Kim, 2019). To address symbolic data scarcity, Sarmento et al. (2021) proposed *DadaGP*, a large-scale corpus of over 26,000 guitar pieces in GuitarPro format, accompanied by a tokenizer for sequence modeling. Although extensive, DadaGP consists solely of symbolic data and contains no audio. Building on this, Zang et al. (2024) introduced *SynthTab*, which renders audio from GuitarPro files using commercial VST instruments, enabling training of transcription models on paired symbolic and audio data. However, SynthTab depends on proprietary tools and does not provide expressive performance annotations. Recent work has also investigated learning transcription models without relying on detailed annotations. Wiggins and Kim (2020) proposed a weakly supervised framework for acoustic guitar transcription that leverages unpaired tablature and audio data. Their system attempts to align symbolic representations with real guitar recordings using heuristic constraints, bypassing the need for frame-level note labels. While promising, such methods still face limitations in accurately modeling articulation and timing nuances. Moreover, several studies have explored guitar synthesis as a tool

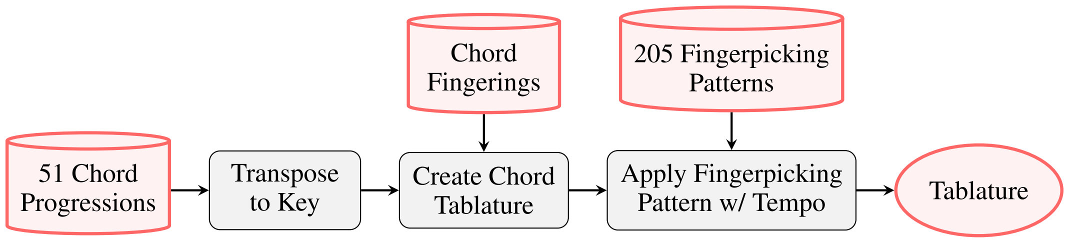

Figure 1: Flow chart of the fingerpicking tablature sampling process.

for data generation and interaction. The Karplus-Strong algorithm (Karplus and Strong, 1983) and its extensions (Jaffe and Smith, 1983; Sullivan, 1990) remain popular due to their efficiency and ability to simulate the dynamics of plucked strings. Convolution-based body modeling methods, such as those by López et al. (2008), enhance realism by simulating guitar body resonance through impulse responses. A more complex set of numerical simulation tools has been introduced by Tahvanainen et al. (2019) to leverage the virtual analysis of guitar mode frequencies, frequency responses, and radiation efficiency in an industrial context. More recently, Jonason et al. (2024) introduced a DDSP-based neural synthesis approach for generating polyphonic guitar audio from string-wise MIDI input. In 2024, Bilbao et al. presented a new, efficient, and numerically stable method for real-time guitar synthesis that models complex string dynamics and nonlinear interactions without requiring computationally expensive iterative solvers (Bilbao et al., 2024).

Despite these advances, most current transcription systems are trained using supervised methods, requiring large datasets of audio paired with precise note annotations. Legal restrictions connected with the use of copyrighted music recordings or MIDI files can make it difficult or expensive to acquire such datasets, particularly for commercial applications.

Furthermore, by using a procedural data generation process, it becomes possible to enable the transcription of more creative playing styles and techniques. This is achieved by creating a more balanced data distribution through the targeted generation of underrepresented musical features. As a result, AI transcription models trained on such data may better capture and notate creative or unconventional performances that are typically underrepresented in real-world datasets, thus ensuring that these artistic expressions are preserved and not lost.

In this work, we propose a fully procedural data generation pipeline as an alternative to using real recordings for training. Our method combines knowledge-based composition of fingerpicking tablature, expressive MIDI performance rendering, physical modeling using an extended Karplus-Strong algorithm, and audio augmentation simulating room acoustics and recording imperfections. This approach enables the scalable creation of musically coherent, expressive, and annotated training data, offering a promising solution for data-scarce transcription scenarios.

2 Procedural data generation

To address the scarcity of labeled training data for acoustic guitar transcription, particularly under legal constraints associated with copyrighted recordings, we propose a procedural data generation pipeline. The goal of this system is to synthesize large-scale, musically coherent, and expressive training data entirely through algorithmic means, eliminating the dependency on real audio or annotated MIDI files.

The key requirements for such a pipeline include: musical validity, ensuring that generated content adheres to realistic harmonic and rhythmic structures; expressive realism, capturing the imperfections and tonal nuance of human performances; computational efficiency, allowing large datasets to be created without introducing training bottlenecks.

Our pipeline consists of four main stages. It begins with the composition of fingerpicking tablatures, where chord progressions are combined with stylistic picking patterns to create musically plausible pieces. These tablatures are then transformed into expressive MIDI sequences using a performance rendering step that introduces timing and pitch variability to emulate human playing. The MIDI is rendered into audio using an extended Karplus-Strong algorithm, which simulates the physical behavior of plucked guitar strings. Finally, an audio augmentation stage applies effects such as distortion, filtering, reverb, and noise to emulate diverse recording environments and increase the acoustic diversity of the dataset.

This modular design enables the generation of rich, annotated audio data suitable for training transcription models in data-scarce settings, and provides a scalable alternative to curated real-world datasets.

2.1 Fingerpicking tablature sampling

The data generation process begins with the creation of synthetic fingerpicking guitar tablatures, as illustrated in Figure 1. This step aims to produce musically realistic and stylistically diverse compositions suitable for rendering into audio and training transcription models.

To achieve this, we use a curated database containing 51 chord progressions written in functional harmony and 205 fingerpicking patterns arranged on a 16th-note rhythmic grid. The picking patterns are inspired by classic fingerstyle repertoire as cataloged by Manzi (2000), and are encoded using the *PIMA* system; an abbreviation derived from Spanish finger names: *P* (pulgar, thumb), *I* (índice, index), *M* (medio, middle), and *A* (anular, ring). This notation allows us to specify which right-hand finger plucks which string in a given rhythmic slot.

Each generated tablature is created through a multi-step sampling process. First, a chord progression is randomly selected and transposed to a randomly chosen key. Each chord is then mapped to a specific fingering using a pre-defined lookup table. Next, a fingerpicking pattern is sampled from the database and applied over the chord progression at a randomly selected tempo. Patterns are available in various time signatures (4/4, 3/4, 6/8, and 12/8), and can be applied to any chord containing at least four active strings.

The plucking instructions rely on string position encoding: positive indices count downward from the highest-pitched string (high E_4), while negative indices count upward from the lowest-pitched string (low E_2). This system ensures compatibility with complex fingerings and variable chord voicings. Theoretically, it can be applied to any guitar tuning, but here we focus on the standard tuning.

This knowledge-based approach enables the generation of an extensive number of musically coherent and stylistically diverse fingerpicking pieces, without the need for manually annotated datasets. An example output, using a Travis picking pattern, is shown in Figure 2.

Tablatures are handled programmatically using the `PyGuitarPro` library (Abakumov, 2023), which supports reading, editing, and exporting Guitar Pro files. This allows for direct visualization and editing of generated tablatures using standard notation software.

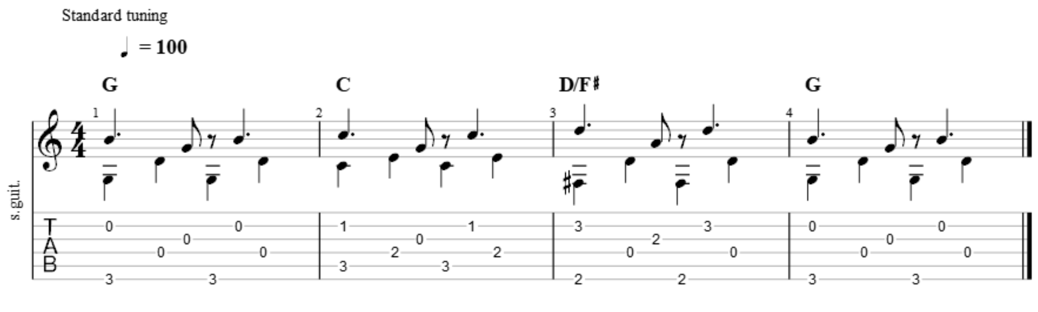

Figure 2: Example tablature generated by the proposed fingerpicking generator using a Travis picking pattern from the pattern database.

2.2 MIDI performance creation

Once the fingerpicking tablatures are generated, they are converted into expressive MIDI performances that emulate human playing. This step bridges the gap between symbolic composition and audio synthesis by introducing temporal and pitch-level variations through a process known as humanization.

Each tablature is translated into a sequence of MIDI-style note events, where timing and pitch characteristics are perturbed to increase realism and variability. Timing deviations are introduced by adding random jitter to both the note onset and offset, with a maximum deviation of 10 % of the note's nominal duration. This simulates subtle imperfections in timing typically present in human performances.

To enhance pitch diversity, we apply probabilistic pitch perturbations that introduce melodic variations and bass runs into the underlying fingerpicking tablature. For each note, there is an 80 % chance the original pitch is retained. With a 5 % chance each, the pitch is shifted up or down by one or two semitones. Although this reduces the strict fidelity of the original composition, it introduces melodic contour, particularly on higher strings, that has been found to improve transcription performance in solo guitar passages.

This performance-level variation enhances the acoustic diversity of the training data, helping the transcription model generalize better to expressive or imperfect recordings.

2.3 Audio rendering

To synthesize audio from MIDI sequences, we employ an extended Karplus-Strong algorithm (Jaffe and Smith, 1983), which models the behavior of a vibrating string using a delay line and a series of digital filters. This physically inspired method is both computationally efficient and highly controllable, making it well suited for scalable procedural data generation. An overview of the synthesis process is illustrated in Figure 3.

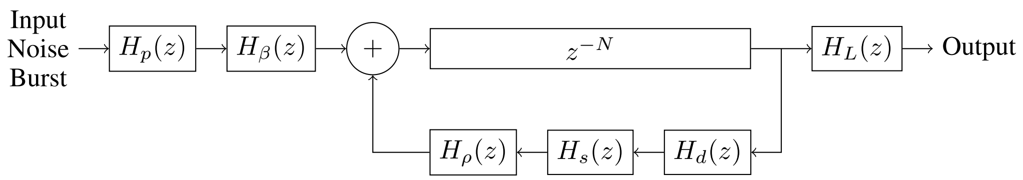

Figure 3: Flow chart of the extended Karplus-Strong synthesis method, adapted from Jaffe and Smith (1983).

The synthesis begins with an excitation signal, typically a burst of filtered white noise, which is fed into a recursive delay loop. The delay length N corresponds to the pitch period in samples (twice the simulated string length). The loop includes several digital filters that simulate various physical properties of the string:

$$\begin{aligned}
H_p(z) &= \frac{1-p}{1-pz^{-1}} && \text{(pick-direction lowpass filter)} \\
H_\beta(z) &= 1 - z^{-|\beta N + 1/2|} && \text{(pick-position comb filter), } \beta \in (0,1) \\
H_d(z) &= \text{string-damping filter} && \text{(must satisfy } |H_d\left(e^{j\omega T}\right)| \leq 1 \text{ for stability)} \\
H_s(z) &= \text{string-stiffness allpass filter} && \text{(simulating dispersion)} \\
H_\rho(z) &= \frac{\eta(N) - z^{-1}}{1-\eta(N)z^{-1}} && \text{(tuning allpass filter)} \\
H_L(z) &= \frac{1-R_L}{1-R_L z^{-1}} && \text{(dynamic-level lowpass filter)}
\end{aligned}$$

Each filter plays a distinct role in modeling the acoustic behavior of the string: $H_p(z)$ adjusts spectral roll-off based on pick direction. The parameter $p \in [0, 1]$ defines the pole position in the lowpass filter, with $p = 0$ for one direction and $p \in (0, 1]$, for the opposite. $H_\beta(z)$ simulates the effect of pick position along the string, controlled by the normalized position parameter β. $H_d(z)$ applies damping to simulate energy decay over time. For stability, the frequency response must satisfy $|H_d(e^{j\omega T})| \leq 1$. $H_s(z)$ models stiffness-related dispersion using an allpass filter with multiple poles and zeros. $H_\rho(z)$ allows fine pitch adjustment via a fractional-delay allpass filter. The coefficient $\eta \in [-1/11, 2/3]$ adjusts the effective delay in the range $[0.2, 1.2]$ samples. $H_L(z)$ simulates dynamic-level dependent brightness, with $R_L = e^{-\pi LT}$, where L is the desired bandwidth in Hz and T is the sampling interval.

To enhance diversity and realism, we randomly sample the following synthesis parameters for each note:

$$\begin{aligned}
\text{Amplitude } A &\in [0.2, 1.3] \\
\text{Brightness } \beta &\in [0.1, 0.9] \\
\text{Level } L &\in [0.1, 0.9] \\
\text{Pick Position } p &\in [0.1, 0.9] \\
\text{Detune } \delta &\in [-0.49, 0.49] \text{ semitones}
\end{aligned}$$

Detuning is applied by offsetting the MIDI pitch m before converting to frequency. The fundamental frequency f_0 is calculated as

$$f_0 = 440 \cdot 2^{\frac{m+\delta-69}{12}}, \tag{1}$$

where δ is the detune value in fractional semitones. The delay length is then computed as

$$N = \frac{f_s}{f_0}, \tag{2}$$

with f_s being the sampling rate of 16 kHz. This method introduces subtle, realistic pitch variations between notes, mimicking tuning inconsistencies found in real guitar performances and improving model robustness.

By combining physical modeling with randomized parameter modulation, the extended Karplus-Strong synthesis engine produces highly expressive, diverse, and controllable audio signals.

2.4 Audio augmentation

To enhance realism and bridge the gap between synthetic and real-world recordings, we apply a post-processing audio augmentation pipeline using the `Pedalboard` library by Sobot (2021). This stage introduces acoustic variability of the recording equipment and environment, aiming to improve the robustness of the transcription model when deployed on non-synthetic data.

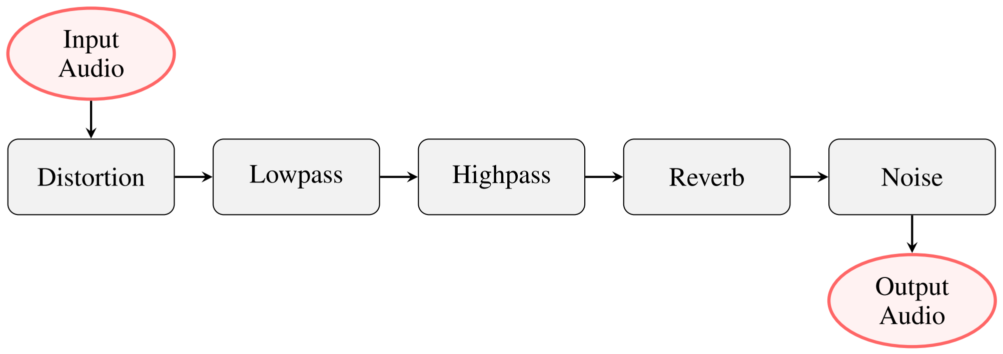

Figure 4: Flow chart of the audio augmentation pipeline applied to synthesized fingerpicking recordings.

As illustrated in Figure 4, the augmentation chain includes distortion, highpass and lowpass filtering, convolutional reverb, and additive noise. Each effect is applied independently with a 50 % probability, and its parameters are randomly modulated to ensure diverse and plausible output variations. Distortion is applied with a randomly sampled drive level in the range of 1 to 4 dB, simulating nonlinear saturation such as pickup overdrive or analog warmth. Lowpass filters are configured with cutoff frequencies uniformly sampled between 1.5 kHz and 8 kHz, while highpass filters use cutoff frequencies in the range of 50 Hz to 500 Hz, mimicking tonal shaping introduced by different microphones or hardware. Reverb is added using convolution with impulse responses, where the virtual room size is randomly chosen between 0.25 and 1.0, simulating a range from small practice spaces to large halls. Finally, white noise is injected at a randomly selected signal-to-noise ratio (SNR) between 30 dB and 50 dB, representing environmental or equipment noise often present in non-studio conditions.

This controlled randomness introduces meaningful variability into the dataset without significantly altering the underlying musical content. As a result, the augmented audio more accurately reflects the diversity and imperfections of real acoustic guitar recordings, leading to better generalization in downstream transcription tasks. Although designed for synthesized audio, the augmentation pipeline is generalizable and can also be applied to VST-generated data or real recordings.

3 Experimentation setup

This section details the model architecture, dataset, and training configuration used in our experiments.

3.1 Model

We adopt the CRNN-based *Onsets and Frames* (OaF) model proposed by Hawthorne et al. (2018) as the core architecture for note tracking. Despite the emergence of more recent approaches such as transformer-based models (Gardner et al., 2022) and regression-based networks (Riley et al., 2024), we selected OaF for its training efficiency, simplicity, and strong baseline performance.

The model is adapted for acoustic guitar transcription by modifying its output dimensionality and tuning hyperparameters accordingly. Input audio is resampled to 16 kHz and converted into a log-scaled Mel spectrogram using a window size of 2048 samples and a hop size of 512. This results in a time-frequency representation with 229 frequency bins, starting from a minimum frequency of 30 Hz.

The architecture is visualized in Figure 5 and consists of two parallel processing branches, one for note onsets and one for sustained frames, each producing a $(B \times N \times 49)$ piano roll representation. Here, B denotes the batch size and N the dynamically set number of time frames. The pitch range spans from E_2 (MIDI 40) to E_6 (MIDI 88), covering standard acoustic guitar tuning.

Each branch begins with a convolutional stack of three 3×3 layers, each followed by batch normalization and ReLU activation. Max pooling and dropout are applied after the second and third convolutional layers. The resulting feature maps are passed through a fully connected layer that compresses the embedding to 256 dimensions. This embedding is then fed into a bidirectional LSTM (BiLSTM), followed by a final fully connected layer with sigmoid activation to output note probabilities.

For training, we use binary cross-entropy losses for both the onset and frame outputs. The total loss is computed as the sum of these two components.

3.2 Datasets

We use the GuitarSet dataset (Xi et al., 2018) to train our baseline checkpoints and evaluate our models. GuitarSet consists of 360 annotated recordings, including both solo and accompaniment (comping) performances from six guitarists. Recordings are captured using a hexaphonic pickup, enabling semi-automatic note-level annotations.

To ensure subject independence during evaluation, we use recordings from one guitarist (the first subject) as the test set, and the remaining five as the training set. Evaluation metrics are reported separately for solo and accompaniment subsets within the test data.

3.3 Training

All models are trained using the Adam optimizer with an initial learning rate of 6×10^{-4}. Gradient clipping with a threshold of 3 is applied to stabilize training. We train for 10,000 steps using a batch size of 8. On an NVIDIA Tesla V100 GPU, each full training run takes approximately 2 h.

For finetuning experiments, we reduce the learning rate to 6×10^{-5} and halve the batch size to 4, allowing for more fine-grained updates on the real data.

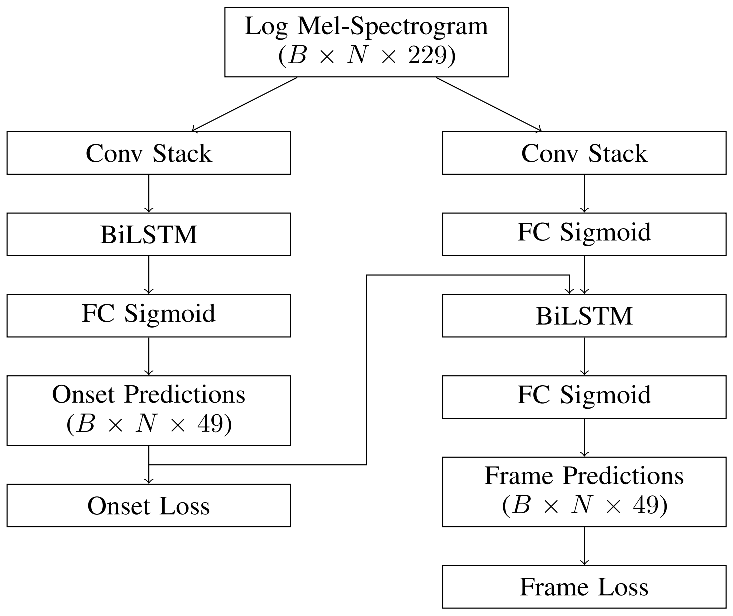

Figure 5: Flow chart of the Onsets and Frames model architecture by Hawthorne et al. (2018) adapted for guitar note-tracking.

Table 1: Baseline comparison of the note Precision (P), Recall (R), and F1-Score (F1) results in percent for different audio sources on the GuitarSet test split. We apply metrics to the full test split, as well as to the accompaniments and solos individually.

Audio Source	**Full**			**Comp**			**Solo**		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
Real Recordings	92.05	73.84	81.42	90.75	63.02	74.15	93.35	84.66	88.70
VST Synthesis	84.87	63.88	71.58	87.36	51.23	64.17	82.37	76.52	78.98
Karplus-Strong	80.62	60.83	68.22	79.02	48.56	59.41	82.23	73.09	77.02

4 Results

To assess the quality of the MIDI transcriptions, we report note-level precision, recall, and F1-Score, using a 50 ms tolerance window in accordance with the `mir_eval` library (Raffel et al., 2014).

4.1 Baseline evaluations

We begin by evaluating transcription performance on three different training sources: real audio recordings, VST-synthesized audio, and audio synthesized using the extended Karplus-Strong model. Each model is trained for $10,000$ steps to ensure a fair and consistent comparison across sources. Table 1 summarizes the results.

VST Audio rendering is performed using the `DAWDreamer` Python library (Braun, 2021) and Ample Sound's sample-based virtual guitar instruments[1], following a methodology similar to SynthTab (Zang et al., 2024). Our Karplus-Strong synthesis setup includes all parameter modulations and audio augmentations described in Section 2.

As expected, training on real recordings yields the highest scores for both accompaniment and solo tracks. Across all three audio sources, solos consistently outperform accompaniment recordings, likely due to the higher note density in comping tracks. VST training data results in an F1-Score approximately 12 % lower than real recordings. The gap can be attributed to the VST's lack of expressive imperfections and recording noise present in amateur performances.

[1] `https://amplesound.net/en/index.asp`

Table 2: Impact of individual Karplus-Strong synthesis parameter modulations on transcription performance. Metrics (Precision, Recall, F1-Score) are reported on the full test set. Each row isolates a single modulated parameter, while the "Combined" row reflects the use of all modulations together, demonstrating their cumulative effect on model generalization.

Parameter Modulation	**Precision**	**Recall**	**F1-Score**
None	70.16 %	18.02 %	27.65 %
Amplitude	58.30 %	24.63 %	33.54 %
Brightness	71.08 %	30.53 %	41.25 %
Level	70.49 %	21.51 %	31.73 %
Position	74.28 %	20.48 %	30.88 %
Detune	79.65 %	53.18 %	61.67 %
Combined	70.62 %	61.42 %	64.44 %

Despite being fully synthetic and not derived from any real guitar audio, the Karplus-Strong synthesis achieves performance comparable to the VST baseline. In contrast, VST rendering such as with Ample Sound requires running within a headless DAW environment like DAWDreamer braun2021dawdreamer, which not only introduces overhead and slows down the generation process but also complicates the implementation of the procedural data generation pipeline. Furthermore, most plugins are only available for Windows and macOS, making them unsuitable for Linux-based workflows. Our method runs natively and efficiently on Linux systems with minimal resource demands, making it well suited for scalable procedural data generation.

4.2 Effect of synthesis modulation

To assess the impact of parameter modulation in the Karplus-Strong synthesis, we trained models on audio generated from the JAMS annotation files of the GuitarSet training split using both static and modulated parameters (without further audio augmentation). Default values were: amplitude = 1, brightness = 0.5, level = 0.2, position = 0.5, and no detuning.

As shown in Table 2, the static version yields high precision but poor recall, indicating overfitting to a narrow sound profile. Parameter modulation substantially improves generalization, especially with brightness and detune having the largest impact. The combined modulation achieves the highest F1-Score (64.44 %), tripling recall while maintaining precision. These results emphasize the importance of timbral variability over strict audio fidelity.

4.3 Effect of audio augmentation

Since the Karplus-Strong model simulates only string excitation, we evaluated the role of post-processing audio effects in simulating realistic recordings. Building upon the combined modulation setup, we applied each augmentation individually and in combination.

As shown in Table 3, individual augmentations yield modest improvements. However, combining all effects (distortion, filtering, reverb, and noise) produces a noticeable performance boost, particularly for solos, where the F1-Score improves from 70.35 % to 77.02 % (not shown in Table 3). This highlights the value of environmental realism in audio generation.

4.4 Procedural data generation performance

In this experiment, we evaluate the full procedural pipeline's capability to generate useful training data from scratch, without relying on existing tablatures. We compare three data generation strategies:

1. **Simple Greedy Generator**: A naive algorithm that iteratively inserts random notes for each string with randomly sampled durations until a fixed target length is reached (see Figure 6). This method does not incorporate any harmonic, rhythmic, or stylistic constraints.
2. **MMM Transformer Model** (Ens and Pasquier, 2020): A neural generative model trained on quantized note sequences from the GuitarSet training split. The model is used to generate

Table 3: Evaluation of individual and combined audio augmentation strategies applied to Karplus-Strong synthesized training data. Results are reported as Precision, Recall, and F1-Score on the full test set, illustrating the contribution of each effect (distortion, filtering, reverb, noise) to transcription performance and the cumulative benefit of combined augmentation.

Augmentation	**Precision**	**Recall**	**F1-Score**
None	67.29 %	63.79 %	64.41 %
Distortion	70.09 %	60.85 %	63.94 %
Lowpass	68.95 %	60.84 %	63.43 %
Highpass	68.81 %	62.84 %	64.67 %
Reverb	75.36 %	57.81 %	64.38 %
Noise	72.28 %	61.23 %	64.79 %
Combined	80.62 %	60.83 %	68.22 %

Table 4: Evaluation results (in percent) for models trained on procedurally generated datasets composed using different tablature composition methods. Metrics include Precision (P), Recall (R), and F1-Score (F1), reported separately for full GuitarSet test data, accompaniments (Comp), and solos (Solo).

Composer	**Full**			**Comp**			**Solo**		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
Simple	70.11	46.65	52.64	73.81	28.95	40.37	66.41	64.35	64.91
MMM	68.68	55.30	60.21	68.39	46.35	54.28	68.97	64.24	66.14
Fingerpicking	74.69	61.99	66.23	73.79	47.08	56.42	75.59	76.90	76.04

MIDI representations, which are then converted to tablature and rendered into audio using the synthesis and augmentation pipeline.

3. **Proposed Fingerpicking Composer**: A rule-based system that samples realistic fingerstyle patterns from a curated database and applies them to structured chord progressions. The compositions are transposed, humanized, rendered to audio using Karplus-Strong synthesis, and finally augmented to emulate realistic performance conditions.

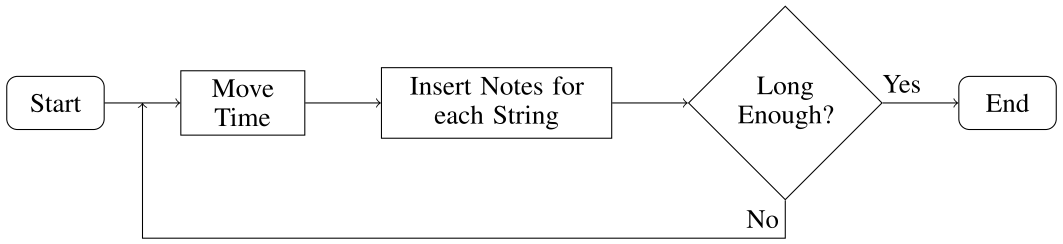

Figure 6: Flowchart of the greedy random tablature generation algorithm.

The comparative results are presented in Table 4. Among the three methods, the simple generator performs the worst, especially for accompaniment tracks, where its lack of structure and musicality results in poor recall and overall low transcription quality. The MMM transformer offers a significant improvement, particularly for comping, due to its data-driven understanding of harmonic and rhythmic structure. However, its results still lag behind those achieved with real or structured synthetic data.

Our proposed fingerpicking composer outperforms both baselines across all metrics and subsets. It achieves a notable 15 % relative increase in F1-Score on solo tracks compared to the MMM model and an even greater improvement over the simple generator. This demonstrates the importance of musical structure and stylistic relevance in synthetic training data for transcription tasks.

Table 5: Ablation study evaluating the impact of humanization and audio augmentation on transcription performance. Metrics (Precision, Recall, F1-Score) are reported for the full test split. The proposed method includes both techniques; ablated versions omit either augmentation or humanization to assess their individual contributions.

Ablation	**Precision**	**Recall**	**F1-Score**
Proposed	74.69 %	61.99 %	66.23 %
No Augmentation	61.38 %	56.95 %	57.54 %
No Humanization	79.18 %	52.36 %	61.90 %

To further investigate the components contributing to the performance of the fingerpicking pipeline, we conduct an ablation study isolating the effects of audio augmentation and MIDI humanization.

As shown in Table 5, removing audio augmentation results in a 15 % drop in F1-Score, underscoring the importance of simulating recording imperfections and environmental acoustics. Similarly, removing the humanization step, responsible for introducing small timing and pitch variations, yields a 7 % performance reduction. These variations likely improve model robustness to expressive nuance in real recordings.

Taken together, these findings validate our full procedural pipeline as an effective approach for generating realistic and diverse training data. While the fingerpicking composer captures the musical essence of fingerstyle guitar, it is the combination with expressive synthesis and augmentation that enables generalization to real-world recordings.

Nevertheless, the results also highlight that the main performance bottleneck lies not in the composition, but in the fidelity of the audio rendering. Future improvements in physical modeling or differentiable synthesis could help bridge the remaining performance gap with real recordings.

To illustrate the diversity and realism achieved through our pipeline, we provide a curated set of audio examples covering the full spectrum of generated data[2]. This includes excerpts from the three compositional strategies (simple, MMM, and fingerpicking-based), as well as side-by-side comparisons of different synthesis settings and augmentation effects.

4.5 Finetuning with real audio

Table 6: Transcription performance after finetuning on real recordings, comparing models trained from scratch on real data, trained on procedural data only, and pretrained on procedural data followed by finetuning. Results are reported in percent as Precision (P), Recall (R), and F1-Score (F1) for the full test set, accompaniment (Comp), and solo (Solo) subsets.

Training Data	**Full**			**Comp**			**Solo**		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
Real Recordings	92.05	73.84	81.42	90.75	63.02	74.15	93.35	84.66	88.70
Procedural Data	74.69	61.99	66.23	73.79	47.08	56.42	75.59	76.90	76.04
Finetuning	92.14	76.95	83.49	90.44	68.02	77.41	93.85	85.87	89.56

To investigate the utility of procedural data for pretraining, we first trained models for $10,000$ steps on procedurally generated audio, then finetuned on real recordings. As shown in Table 6, pretraining yields a modest 2 % F1-Score gain over training solely on real data.

The benefits become more pronounced with reduced finetuning data. Figure 7 illustrates that pretraining enables comparable performance with only a fifth of the real recordings. For example, training from scratch with 60 recordings results in an F1-Score of 63.32 %, whereas pretraining lifts that to 77.45 %.

This demonstrates the value of procedural pretraining, especially in low-data scenarios or for underrepresented instruments that could benefit from similar synthesis pipelines.

[2]https://github.com/klangio/procedural-data-training

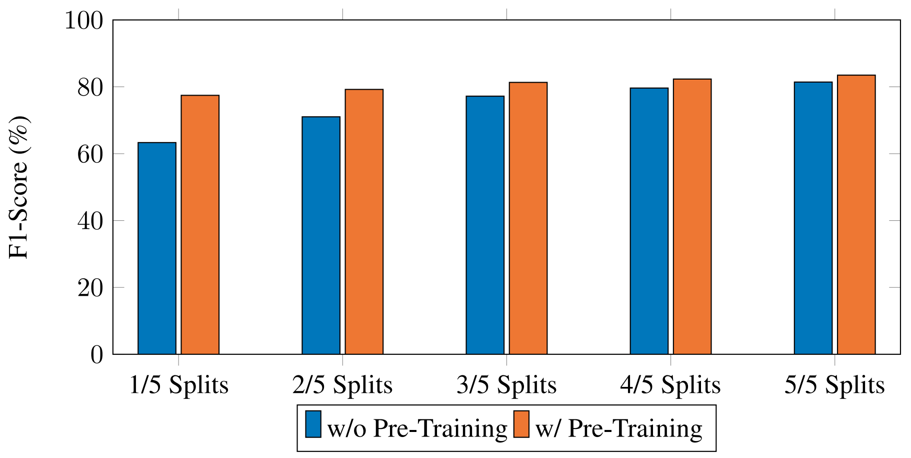

Figure 7: Demonstration of the effectiveness of pre-training when reducing the amount of real data. The training dataset is divided by guitarist into five splits with 60 recordings each. The evaluation is performed on the full test split.

5 Conclusion

In this work, we investigated the use of procedurally generated training data for the task of automatic transcription of fingerpicked acoustic guitar performances. We introduced a novel data generation pipeline comprising the composition of fingerpicking tablatures, conversion to performance-level MIDI, audio synthesis using an extended Karplus-Strong algorithm, and audio augmentation. This fully synthetic pipeline enables the creation of annotated training data without reliance on copyrighted recordings. Our experiments demonstrated that models trained solely on procedurally generated audio can achieve competitive transcription accuracy. Moreover, pretraining on synthetic data followed by finetuning on real recordings yielded improved performance compared to training exclusively on real data and requires significantly fewer real recordings to achieve comparable results. These findings suggest that procedural data generation can be a powerful tool for overcoming data scarcity in music transcription tasks.

Future work could explore applying this procedural training approach to more advanced model architectures such as transformer-based models (Gardner et al., 2022) or regression-based CRNNs (Riley et al., 2024). Additionally, integrating differentiable digital signal processing (DDSP) synthesis techniques (Jonason et al., 2024) could enable richer supervision, such as direct prediction of string and fret positions. Beyond guitar transcription, the procedural generation framework could be adapted to other instruments, offering a scalable solution for tasks with limited annotated real-world data.

References

Abakumov, S. (2023). Pyguitarpro. https://github.com/Perlence/PyGuitarPro.

Bilbao, S., Russo, R., Webb, C. J., Ducceschi, M., et al. (2024). Real-time guitar synthesis. In *Proceedings of the 27th International Conference on Digital Audio Effects (DAFx24)*.

Braun, D. (2021). DawDreamer: Bridging the Gap Between Digital Audio Workstations and Python Interfaces. In *Extended Abstracts for the Late-Breaking Demo Session of the 22nd International Society for Music Information Retrieval Conference (ISMIR)*.

Ens, J. and Pasquier, P. (2020). MMM : Exploring Conditional Multi-Track Music Generation with the Transformer. *arXiv preprint arXiv:2008.06048*.

Gardner, J., Simon, I., Manilow, E., Hawthorne, C., and Engel, J. (2022). MT3: Multi-Task Multitrack Music Transcription. In *International Conference on Learning Representations (ICLR)*.

Hawthorne, C., Elsen, E., Song, J., Roberts, A., Simon, I., Raffel, C., Engel, J., Oore, S., and Eck, D. (2018). Onsets and Frames: Dual-Objective Piano Transcription. In *Proceedings of the 19th International Society for Music Information Retrieval Conference (ISMIR)*, pages 50–57.

Jaffe, D. A. and Smith, J. O. (1983). Extensions of the Karplus-Strong Plucked-String Algorithm. *Computer Music Journal*, 7(2):56–69.

Jonason, N., Wang, X., Cooper, E., Juvela, L., Sturm, B. L., and Yamagishi, J. (2024). DDSP-based Neural Waveform Synthesis of Polyphonic Guitar Performance from String-wise MIDI Input. In *Proceedings of the 27th International Conference on Digital Audio Effects (DAFx)*.

Karplus, K. and Strong, A. (1983). Digital Synthesis of Plucked-String and Drum Timbres. *Computer Music Journal*, 7(2):43–55.

López, M., González, J., Esquef, P., and Välimäki, V. (2008). Software Based Acoustic Guitar Simulation by Means of Its Impulse Response. In *Proceedings of the 11th International Conference on Digital Audio Effects (DAFx)*.

Manzi, L. (2000). *Fingerpicking Pattern Encyclopedia: Over 200 Useful Fingerpicking Patterns*. Alfred Music.

Raffel, C., McFee, B., Humphrey, E. J., Salamon, J., Nieto, O., Liang, D., Ellis, D. P., and Raffel, C. C. (2014). MIR_EVAL: A Transparent Implementation of Common MIR Metrics. In *Proceedings of the 15th International Society for Music Information Retrieval Conference (ISMIR)*, page 2014.

Riley, X., Edwards, D., and Dixon, S. (2024). High Resolution Guitar Transcription via Domain Adaptation. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1051–1055.

Sarmento, P., Kumar, A., Carr, C., Zukowski, Z., Barthet, M., and Yang, Y.-H. (2021). DadaGP: A Dataset of Tokenized GuitarPro Songs for Sequence Model. In *Proceedings of the 22nd International Society for Music Information Retrieval Conference (ISMIR)*.

Sobot, P. (2021). Pedalboard. https://doi.org/10.5281/zenodo.7817838.

Sullivan, C. R. (1990). Extending the karplus-strong algorithm to synthesize electric guitar timbres with distortion and feedback. *Computer Music Journal*, 14(3):26–37.

Tahvanainen, H., Matsuda, H., and Shinoda, R. (2019). Numerical Simulation of the Acoustic Guitar for Virtual Prototyping. In *Proceedings of ISMA*, volume 2019, pages 13–17.

Wiggins, A. and Kim, Y. (2019). Guitar Tablature Estimation With a Convolutional Neural Network. In *Proceedings of the 20th International Society for Music Information Retrieval Conference (ISMIR)*, pages 284–291.

Wiggins, A. and Kim, Y. (2020). Towards Unsupervised Acoustic Guitar Transcription. In *Extended Abstracts for the Late-Breaking Demo Session of the 21st International Society for Music Information Retrieval Conference (ISMIR)*, pages 43–55.

Xi, Q., Bittner, R. M., Pauwels, J., Ye, X., and Bello, J. P. (2018). GuitarSet: A Dataset for Guitar Transcription. In *Proceedings of the 19th International Society for Music Information Retrieval Conference (ISMIR)*, pages 453–460.

Zang, Y., Zhong, Y., Cwitkowitz, F., and Duan, Z. (2024). Synthtab: Leveraging Synthesized Data for Guitar Tablature Transcription. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1286–1290.