

RHYTHM GAME CHART GENERATION WITH PRE-TRAINED MUSIC TRANSCRIPTION MODEL

Sojeong An¹

Chris Donahue²

Dasaem Jeong¹

¹ Music & Arts Learning (MALer) Lab, Sogang University, South Korea

² Carnegie Mellon University, USA

{suzuan, dasaemj}@sogang.ac.kr

ABSTRACT

We present a system that generates osu!taiko charts from audio by repurposing a pre-trained automatic music transcription (AMT) model as a frozen audio encoder. Given an audio file and a target difficulty, a four-layer Transformer decoder cross-attends to the intermediate hidden states of the AMT model and autoregressively emits each note as a timestamp on a 10 ms grid followed by one of six osu!taiko note types. The decoder is conditioned on the chart’s star rating, the real-valued difficulty score osu! assigns to each chart, with per-tier median values used at inference time. Post-processing snaps notes to the beat grid and enforces the osu!taiko ranking criteria, producing submission-ready charts at all five difficulty tiers.

1. SYSTEM OVERVIEW

A rhythm game chart specifies the time and type of every note the player must hit in a given song; in osu!, charts are called *beatmaps*. Prior approaches often split chart generation into two stages: one model locates note times and another assigns note types [1, 2]; end-to-end formulations have also been proposed [3, 4]. We treat chart generation as a single sequence prediction task conditioned on audio, using MuScriptor [5], a pre-trained AMT model, as a frozen encoder: a decoder cross-attends to MuScriptor’s hidden states and generates note timestamps and types autoregressively.

Given an audio file, a timing template (the tempo and beat positions of the song, provided with the task), and a target difficulty, the system produces a chart for osu!taiko, osu!’s taiko drum mode. The audio is divided into 5 s segments; the encoder processes each segment and the decoder generates notes for it in turn. The full chart is written to a .osu file after post-processing.

2. DATA

Training data is limited to osu!taiko charts *ranked* (i.e., approved through osu!’s ranking process), between 5 May

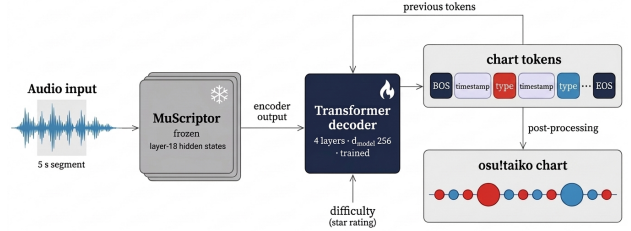


Figure 1. System overview. A frozen transcription model encodes the audio; the decoder cross-attends to its hidden states and emits a timestamp and a type per note. Post-processing aligns the notes to the beat grid before the chart is written.

2019 and 31 December 2025, as the task requires. Charts are distributed in *beatmapsets*, each holding one recording with its charts at several difficulties. We list the beatmapsets in this period with the osu! API v2 and download them from catboy.best.¹ The dataset contains 1,858 songs and 7,725 charts (4.16 per song), totalling 97.0 hours of audio with a mean song length of 188 s.

We split the data by song, not by chart. The same recording may appear under separate beatmapsets, such as a TV-size cut and the full version, so we group songs by normalized title and audio checksum and assign each group to a single split. Training holds 1,487 songs; validation and test hold 185 and 186 songs, respectively.

Difficulty is given to the model as the chart’s *star rating*, a non-negative real number that osu! computes from note density and pattern complexity; a higher value means a harder chart, and the charts in our dataset range from 0.63 to 10.36 stars. Each chart is also labeled with one of five osu!taiko difficulty tiers: Kantan, Futsuu, Muzukashii, Oni, and Inner Oni, from easiest to hardest. Table 1 lists the charts per tier with the median star rating of each tier.

3. MODEL

3.1 Audio representation

The audio representation comes from MuScriptor medium, a 24-layer decoder-only Transformer of 307M parameters,

¹ <https://catboy.best>



	Kantan	Futsuu	Muzukashii	Oni	Inner Oni	Other
Charts	1,239	1,388	1,444	1,348	751	1,555
Median ★	1.33	2.17	3.13	3.99	5.28	–

Table 1. Charts and median star rating per tier. *Other* covers difficulty names outside the five tiers.

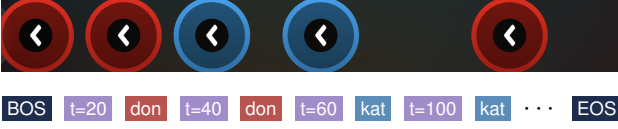


Figure 2. An osu!taiko chart segment (top) and its token sequence (bottom): a timestamp token followed by a type token per note. A timestamp t indexes the 10 ms grid from the segment start ($t = 20$ is 200 ms); each value is a distinct token. Type tokens take their in-game colors.

which encodes a 5 s mel spectrogram frame by frame as a prefix and autoregressively generates the note tokens. We take the hidden states at layer 18, selected by a linear on-set probe: 501 frames of 1024 dimensions per 5 s, one per 10 ms. The hidden states are computed once per song and kept fixed during training.

3.2 Chart representation

A chart is represented as a sequence of tokens, a timestamp and a type for each note. The timestamp is an integer from 0 to 499 indexing a 10 ms grid from the start of the segment, and the type is one of six osu!taiko note types: don and kat (the red and blue hits), big don and big kat (their two-handed finisher variants), and drumroll and denden (held notes with a duration).

A segment with N notes becomes the sequence BOS, t_1 , τ_1 , ..., t_N , τ_N , EOS, where $t_i \in \{0, \dots, 499\}$ is the timestamp token of the i -th note and τ_i is its type token; the sequence has length $2N + 2$, and the timestamps are strictly increasing. The vocabulary holds 509 entries: 500 timestamps, 6 types, BOS, EOS, and PAD.

3.3 Decoder

The encoder hidden states are projected to the decoder dimension with a linear layer followed by a LayerNorm, and the decoder cross-attends to the projected states. The decoder is a four-layer Transformer ($d_{\text{model}} = 256$, 4 heads, $d_{\text{ff}} = 1024$, dropout 0.2). Sinusoidal positional encodings are added to the token embeddings and to the projected encoder states. The decoder has 4.8M trainable parameters; the frozen encoder is not counted.

A two-layer MLP maps the star rating to the model dimension; its output is added to every token embedding, and the sum is normalized with a LayerNorm. The last layer of the MLP is zero-initialized, so the difficulty conditioning has no effect at the start of training.

4. TRAINING

We train the model with a next-token cross-entropy objective. Segments are 5 s long, matching the encoder; as data augmentation, we slice each song at five offsets 1 s apart. Difficulty is balanced by inverse-frequency sampling over star rating bands.

Training runs for 450,000 optimizer steps with Adam (learning rate $3e-4$, weight decay $1e-4$) and a batch size of 64, with a 2% linear warm-up followed by cosine decay to zero. We save the checkpoint with the lowest validation loss for generation.

5. CHART GENERATION

5.1 Decoding

Decoding is conditioned on a fixed difficulty score per tier. We use the median star rating of each tier (Table 1): 1.33, 2.17, 3.13, 3.99, and 5.28 for Kantan through Inner Oni. At each position the logits are masked to the tokens the grammar allows: a timestamp must exceed the previous one, a type must follow each timestamp, and EOS may appear wherever a timestamp is expected. Notes are placed in the song by adding the segment’s start time to each timestamp.

5.2 Post-processing

An osu! chart lies on the beat grid of the song, with per-difficulty rules set by the ranking criteria.² The model predicts times on its own 10 ms grid, so we post-process the generated chart onto the beat grid taken from the timing template. We first snap each note to the nearest grid point, where the finest division allowed depends on the difficulty (Table 2); the median shift is 4.3 ms. We then remove notes closer than the minimum spacing for that difficulty; Inner Oni has no such rule. Finally, we demote finishers inside fast patterns to plain notes, when the pattern speed exceeds the threshold set by the ranking criteria for that difficulty.

Tier	Snap grid	Min spacing	Finisher threshold
Kantan	1/2	1/2	1/2
Futsuu	1/3	1/3	1/3
Muzukashii	1/6	1/6	1/4
Oni	1/8	1/8	1/6
Inner Oni	1/16	–	–

Table 2. Post-processing rules per difficulty tier (beat fractions). Finisher and spacing rules from the osu!taiko ranking criteria.

We give drumrolls and denden a default length in beats, since the model does not predict hold lengths (these types are rare in the training data); it scales with the tempo and is cut short of the next note. Post-processing thus only moves, removes, or demotes notes; it never adds one.

² https://osu.py.sh/wiki/en/Ranking_criteria/osu!taiko

6. REFERENCES

- [1] C. Donahue, Z. C. Lipton, and J. McAuley, “Dance Dance Convolution,” in *Proc. ICML*, 2017.
- [2] E. Halina and M. Guzdial, “TaikoNation: Patterning-focused Chart Generation for Rhythm Action Games,” in *Proc. FDG*, 2021.
- [3] J. Yi, S. Lee, and K. Lee, “Beat-Aligned Spectrogram-to-Sequence Generation of Rhythm-Game Charts,” in *Proc. ISMIR (LBD)*, 2023.
- [4] R. Tripodi, “audio2chart: End to End Audio Transcription into Playable Guitar Hero Charts,” arXiv:2511.03337, 2025.
- [5] S. Rouard, M. Krause, A. Roebel, C.-J. Simon-Gabriel, and A. Défossez, “MuScriptor: An Open Model for Multi-Instrument Music Transcription,” arXiv:2607.08168, 2026.