dit dah delta token: Statistical Models of Music and Language Interfering via Morse Code

Victor Shepardson
Intelligent Instruments Lab
University of Iceland
victorshepardson@hi.is

Thor Magnusson
Intelligent Instruments Lab
University of Iceland
thormagnusson@hi.is

ABSTRACT

dit dah delta token is a live-coding performance using statistical machine learning models and Morse code as material. In the piece, a large language model and a MIDI model co-generate a text which is also a musical rhythm, combining probabilities over each durational element. In this paper, we explain the method in detail and reflect on the composition as a piece of music.

1. INTRODUCTION

Large-scale generative machine learning has rapidly become a major paradigm for artificial intelligence, drawing intense interest in the arts. In particular, so-called language models for sequential domains like text, audio or symbolic music are increasingly part of computer music practice, at both the critical [1] and technical [2] ends.

This paper documents *dit dah delta token*, a performance making a playful inquiry into the aesthetics of language models. The central question is how Morse code, a much older technology, can be used to make artistic interventions on contemporary language models. Our use of Morse is technically arbitrary, but conceptually motivated by the original performance site (Section 5). Morse code relates text to rhythm, with sequences of long and short elements corresponding to characters in an alphabet. In *dit dah delta token*, a text and a MIDI-based musical performance are co-generated by a pair of statistical language models conducted by a live-coding performer. Using Morse code, a music model and a text model are coaxed to 'speak through the same mouth', as the same generated sequence tries to be simultaneously a text and a musical rhythm.

Section 2 supplies background on Morse code and the generative models used in this project. In section 3, a method is described for combining probability distributions from a text and a MIDI model. Finally, section 4 describes the musical performance.

2. BACKGROUND

International Morse code [3] is a method for encoding texts into a binary series of $^{\text{high}}$ and $_{\text{low}}$ segments of different lengths. A $^{\text{high}}$ segment is either one or three time units long ($1u$ or $3u$), making either a *dit* ($\cdot$) or *dah* (–). Sequences of elements correspond to characters, for example in the famous 'SOS' or '$\cdot\cdot\cdot - - - \cdot\cdot\cdot$'.

A $_{\text{low}}$ segment of length $1u$ separates high elements, with longer versions also terminating a character ($3u$) or a word ($7u$). For example, writing 'I' for $1u\,^{\text{high}}$ and '.' for $1u\,_{\text{low}}$, SOS expands to 'I.I.I...III.III.III...I.I.I.....'.

In physical communication systems, $^{\text{high}}$ and $_{\text{low}}$ are often voltage levels in a circuit, as in the telegraph. But they may also be the presence and absence of a signal, such as a tone. An obvious mapping from Morse code to music is to let $^{\text{high}}$ be a note while $_{\text{low}}$ is a rest, evoking the familiar beeping of telegraph Morse.

2.1 Language Models

Deep learning-based statistical models of text have become capable at a surprising breadth of tasks, including machine translation, summarization, and question answering, simply by training on vast corpora of text from the internet and framing each task in terms of text completion. Chat interfaces for LLMs have quickly become ubiquitous tools since the introduction of ChatGPT in late 2022. Suddenly, they are used by millions of people, while surrounded by folklore and controversy regarding the nature of their apparent intelligence.

We don't use LLMs to generate music *per se*, rather exploring how they react to deep coupling to a MIDI model. For this purpose, we want the largest language models we can feasibly run locally. Mamba [4] is a family of LLM based on state-space models in place of the usual Transformer architecture. Mamba models are comparatively efficient and low-latency, and are available in multiple sizes with open source code and publicly available weights, making them ideal for developing our performance software.

2.2 MIDI Models

Many of the same statistical modeling techniques applied to text are also used to model music, on either an acoustic signal or symbolic level [2]. Unlike text, music involves timing and duration. However, elapsed time can be represented by numbers or symbols, just as in music notation, enabling the application of language modeling methods.

Notochord [5, 6] is a model for MIDI sequences which models polyphonic, multi-part music from large datasets, within the constraint that each MIDI event is processed at a realtime latency of about 10 milliseconds. It models MIDI Note On and Note Off events, each broken in four parts: a

MIDI note number (pitch), a velocity, an elapsed time since the previous event, and an operative program number.

Together, Mamba and Notochord supply the material for extemporaneous play with language model aesthetics in *dit dah delta token*.

3. COMBINING TOKEN AND RHYTHM PROBABILITIES

The Mamba language model (LM) and Notochord MIDI model (MM) are both autoregressive: given a partial sequence, they compute a probability distribution over the next element in the sequence, $\mathrm{P}(x_i \mid x_{0..i-1})$. In order to sample 'from both models at once', the two model distributions over the next element can be multiplied together, to favor sequences which look plausible to both models.

However, the LM and MM operate in different domains. The LM models text with a discrete set of tokens, each made up of a variable number of characters. Meanwhile, the MM models music via MIDI Note On and Note Off events and the elapsed time between them. These can be related using Morse code, which represents text characters with sequences of durations. Still, the MM works at a finer granularity than the LM: each Note On or Note Off makes either a dit or dah $_{\text{high}}$ segment, or an interelement, intercharacter, or interword gap $_{\text{low}}$ segment. It takes one or more pairs of elements and gaps to form a character, and one or more characters to form an LM token.

Though the Notochord model can deal with polyphonic and multichannel MIDI, this application only uses monophonic lines. Thus, in the exposition here we ignore the instrument identity, and can assume that Note On and Note Off events always alternate. Furthermore, while velocity and pitch are sampled from the MM and contribute to the music, they don't interact directly with the LM and require no special handling. Therefore, we discuss only the MM's model of durations in the following sections.

To combine the LM and MM model distributions, we need to compute the distribution over time deltas according to the LM at each (shorter) step of the MM. At the same time, we must constrain the MM (LM) only to generate rhythms (texts) which encode (can be encoded to) Morse.

3.1 Notation

A Morse sequence can be written as a series of segments h. These are alternating $1u$ *dit*s or $3u$ *dah*s written as 'd', and $1u$, $3u$ or $7u$ gaps written 'g':

$$h_1, h_2, .. h_{2n} = d_1, g_1, .. d_n, g_n \tag{1}$$

The Morse sequence encodes a text, which can be written as a series of language model tokens $t_1, .. t_i$, each of which is made up of individual characters $c^i_1, .. c^i_j$.

We can mix these notations to write a Morse sequence with some elements at the end which do not yet form complete tokens or characters. For brevity, we write $s_{<ijk}$ to denote a sequence of $i-1$ tokens, followed by $j-1$ characters beginning the ith token, and k Morse segments beginning the jth character:

$$\begin{aligned} s_{<ij} &= t_1, .. t_{i-1}, c^i_1, .. c^i_{j-1} \\ s_{<ijk} &= s_{<ij}, d^{ij}_1, g^{ij}_1, .. d^{ij}_{k-1}, g^{ij}_{k-1} \end{aligned} \tag{2}$$

Thus, g^{ij}_k denotes the kth gap in the jth character of the ith token.

3.2 Token Tree Traversal

At the start of generation, and anytime a token is completed, we query the LM for the probability distribution over next tokens. First, any tokens which don't encode to Morse are excluded, i.e., any tokens containing characters which aren't in our Morse alphabet of lowercase latin letters and limited punctuation, and any tokens containing multiple consecutive spaces.

From the remaining tokens, we construct a prefix tree: each token is a node in the tree, with a child for each string which is identical but longer by one character. Some intermediate nodes don't correspond to tokens and get a probability of zero; otherwise probabilities of each token come from the LM.

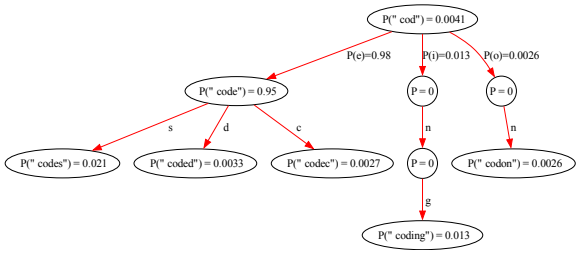

Figure 1. Example token subtree for the prefix " cod".

By summing the model probabilities over each subtree, we can compute the distribution of probability of each next *character* according to the LM:

$$\mathrm{P}_{\text{LM}}(c^i_j \mid s_{<ij}) \propto \sum_{t_i \succeq c^i_{1..j}} \mathrm{P}_{\text{LM}}(t_i \mid t_{1..i-1}) \tag{3}$$

Where $t_i \succeq c^i_{1..j}$ denotes that the character sequence $c^i_{1..j}$ is a prefix of (or equal to) a potential next token t_i.

3.3 Morse Tree Traversal

As individual time-deltas are sampled in the MM domain, they begin to form a character in Morse code, which we track by traversing a binary Morse code tree. In the Morse tree, nodes are characters of the alphabet (excluding the space character). The children of each node are the two characters appending either one more *dit* or *dah* $_{\text{high}}$ element.

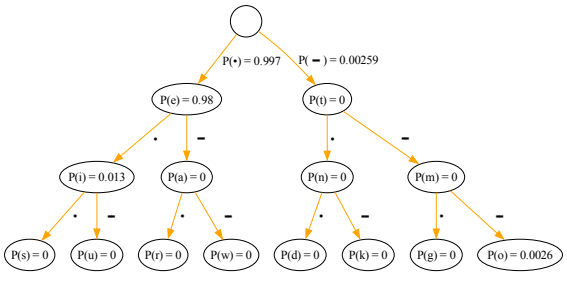

Figure 2. Partial Morse tree corresponding to Figure 1. Character probabilities computed over token subtrees have been assigned to nodes in the Morse tree, and now Morse element probabilities can be computed over the *dit* and *dah* subtrees.

Each Morse node is populated with the character probability computed over the current token subtree, and by summing Morse subtrees we arrive at probabilities for each time delta specifying the duration of a $^{\text{high}}$ segment:

$$\mathrm{P}_{\text{LM}}(d_k \mid s_{<ijk}) \propto \sum_{c_j \succeq d_{1..k}} \mathrm{P}_{\text{LM}}(c_j \mid s_{<ij}) \tag{4}$$

Where d_k denotes the kth *dit* or *dah* element in the next unfinished character c_j.

What about the other half of MM time-deltas, which indicate the length of $_{\text{low}}$ segments? There are three lengths of $_{\text{low}}$ segment: interelement gap $1u$; intercharacter gap $3u$; interword gap (i.e. space character $7u$). If the next gap is $3u$ or $7u$, it implies that the current character is complete. Conversely, if the next gap is $1u$, it means the current character is unfinished. Thus, the LM probability of a character-terminating gap is the ratio of the character probability at the current Morse node to the sum over the whole subtree:

$$\mathrm{P}_{\text{LM}}(g_k > 1u \mid s_{<ijk}) = \frac{\mathrm{P}_{\text{LM}}(c_j = d_{1..k} \mid s_{<ij})}{\mathrm{P}_{\text{LM}}(c_j \succeq d_{1..k} \mid s_{<ij})} \tag{5}$$

Where g_k denotes the kth gap within, or terminating, the next character c_j. If a $1u$ gap is sampled, traversal of the Morse tree continues with the next $^{\text{high}}$ segment. Otherwise, the current node in the Morse tree becomes the next character, which advances traversal of the token tree, possibly completing a token.

3.4 End of Token Sampling

Whenever a character is finished, the token tree is traversed. If we've reached a leaf node, the token is complete. Otherwise, we need to decide whether the token is completed or continues. This works analogously to testing the end of each Morse character in equation (5): we compare probabilities of the current node in the token tree with the probability of its entire subtree, and randomly sample whether to stop. When a token is complete, we feed the completed token to the LM, query for next token probabilities, and reset the token tree, also recomputing character probabilities for the Morse tree.

3.5 Space Handling

In the Mamba tokenizer, whitespace occurs only at the beginning of tokens. So, we resolve character-terminating gaps to either $3u$ or $7u$ only once a token ends. The LM probability of a $7u$ versus $3u$ gap is the relative probability of the next character being a space versus anything else, according to the token tree and equation (3).

When a $7u$ rather than $3u$ gap is sampled, a space character is also inserted and the token tree is traversed to the subtree beginning with a space before continuing to the next $^{\text{high}}$ segment.

3.6 Music Model Probabilities

The Notochord models the elapsed time between consecutive MIDI events. It parameterizes a cumulative distribution function f_{MM} over durations, meaning the probability mass can easily be computed over a given range. We use this fact to discretize time into the $1u$, $3u$ and $7u$ durations of Morse code. Omitting indices for brevity,

$$\begin{aligned}
\mathrm{P}_{\text{MM}}(h = 1u \mid \cdot) &\propto f_{\text{MM}}(1.5u \mid \cdot) - f_{\text{MM}}(0.5u \mid \cdot) \\
\mathrm{P}_{\text{MM}}(h = 3u \mid \cdot) &\propto f_{\text{MM}}(4.5u \mid \cdot) - f_{\text{MM}}(1.5u \mid \cdot) \\
\mathrm{P}_{\text{MM}}(g = 7u \mid \cdot) &\propto 1 - f_{\text{MM}}(4.5u \mid \cdot)
\end{aligned} \tag{6}$$

Where h_n is the nth Morse segment, i.e. $h_n = d_{(n+1)/2}$ for odd n and $h_n = g_{n/2}$ for even n.

At each time step, the MM is conditioned on all previous MIDI events and the globally selected MIDI program (General MIDI instrument).

3.7 Combining Model Probabilities

For each MIDI event, we can now compute the probability of each possible duration according to both LM (equations 4, 5) and MM (equation 6). All that remains is to combine P_{LM} and P_{MM} into one distribution before sampling.

This can be accomplished by multiplying P_{LM} and P_{MM} together, then normalizing so probabilities sum to 1. We also include separate temperature parameters so the relative influence of MM and LM can be tuned:

$$\mathrm{P}(\,\cdot\,) \propto \mathrm{P}_{\text{MM}}(\,\cdot\,)^{\frac{1}{\tau}}\, \mathrm{P}_{\text{LM}}(\,\cdot\,)^{\frac{1}{\gamma}} \tag{7}$$

By merging LM probabilities into sampling of time deltas in this way, we produce a MIDI sequence which attempts to make sense simultaneously as music and as a text.

4. *DIT DAH DELTA TOKEN*

The technique described in Section 3 was developed concurrently with a musical piece entitled *dit dah delta token*.

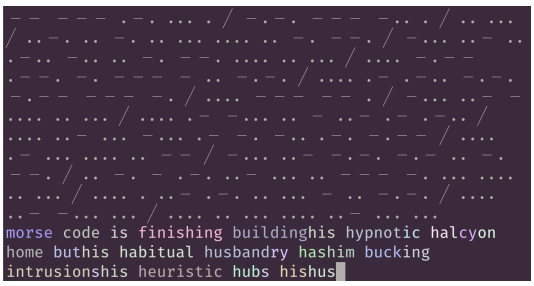

Figure 3. Visual element for one voice.

After some experimentation with very low latency and live MIDI input, it was instead decided to take advantage of the precise machinic rhythm of algorithmically generated Morse code, while also enabling the use of larger language models, by buffering several MIDI events ahead to enable precise timing. This soon led to the idea to run multiple instances of the software at once, which would act independently, but always remain in a precise rhythmic relationship via the setting of u. This was found to give a satisfying degree of serendipity and performative latitude in the layering of otherwise often drab and random sounding MIDI lines, without detracting from the intention to let the models speak for themselves.

dit dah delta token is performed in a brutalist live-coding style, with the performer navigating a multi-pane terminal,

running and stopping instances of the software from the command line. A visual element displays the text and a graphic representation of the corresponding Morse code in time with the sound (Figure 3). The text is colored to show the boundaries between tokens, so that the token, character, Morse, and musical layers can be simultaneously perceived. This is printed directly into the terminal panes, alongside the MIDI stream running through fluidsynth.

The performer starts each instance with a text prompt (e.g., "morse code is") which the models continue. The performer decides when to start and stop each instance, chooses which instrument each one plays, and writes new prompts in response to the texts generated by the models. Each performance is unique, due to both random sampling of the language models and improvisation by the performer. Documentation can be viewed online. [1]

4.1 Implementation

The time unit u was chosen to be 50 milliseconds, which is fast enough to maintain interest, but slow enough to run the models comfortably in real time and for many of the instrument sounds to work well.

The software is implemented into the `notochord` open source Python package from version `0.5.7`, and is run on a MacBook Pro computer during the performance. MM is a Notochord model [2] running on CPU, while LM is a Mamba model [3] running on the GPU.

Both models are pre-trained on large scale internet data. For Notochord, the Lakh MIDI dataset [7], contains MIDI Program Change events indicating the standard General MIDI instruments, so we use fluidsynth [4] for audio synthesis, which supplies the full range of instruments with vintage MIDI aesthetics appropriate to the dataset.

5. DISCUSSION

dit dah delta token was inspired by a collaboration with the Loftskeytastöðin museum in Reykjavík, which formerly housed the city's radio equipment. It was first performed in Loftskeytastöðin in November 2024, then again in the ErkiTíð computer music festival.

We preferred to leave digits out of the Morse alphabet, finding that the models would often sample long strings of digits which were less interesting to read. The music model has a strong preference for making simple rhythms, particularly liking the letters 'h / ····' and 'o / – – – –', which are the longest possible runs of the same element before a longer gap must occur. Often the letter 'h' becomes gradually more prominent until entering a loop repeatedly sampling only 'h', or some short cycle of tokens, with the music model seeming to dominate. Yet occasionally, the language model does continue a text for a long time with no persistent rhythm; either model might become very uncertain after seeing a certain amount of nonsense, leaving the other model with most of the influence.

The constraints each model imposes on the other limit their performance from an objective point of view. However, the tension between models an ensuing failures are precisely what generates interest and humor in the piece.

6. CONCLUSIONS

dit dah delta token is a playful performance using modern statistical machine learning models as a material. In the process of composing it, we learned about the agencies and detail latent in this material. Our process was technically rigorous, yet artistically whimsical; our posture toward the technology, informed but skeptical, seeking to recover a space for absurdity and play amid the machinery. We hope for some of our curiosity to be transmitted to an audience, and satisfied in turn by this paper and our open source software.

Acknowledgments

The Intelligent Instruments Lab is supported by the European Research Council (ERC) as part of the Intelligent Instruments project (INTENT), under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101001848). This research was also supported by an NVIDIA hardware grant.

7. REFERENCES

[1] B. L. T. Sturm *et al.*, "AI Music Studies: Preparing for the Coming Flood," *AIMC 2024*, Aug. 2024. [Online]. Available: https://aimc2024.pubpub.org/pub/ej9b5mv1/release/1

[2] Y. Ma *et al.*, "Foundation Models for Music: A Survey," Aug. 2024, arXiv:2408.14340 [cs, eess]. [Online]. Available: http://arxiv.org/abs/2408.14340

[3] "International Morse Code Recommendation," International Telecommunication Union, Tech. Rep. ITU-R M.1677-1, 2009.

[4] A. Gu and T. Dao, "Mamba: Linear-Time Sequence Modeling with Selective State Spaces," May 2024, arXiv:2312.00752 [cs]. [Online]. Available: http://arxiv.org/abs/2312.00752

[5] V. Shepardson, J. Armitage, and T. Magnusson, "Notochord: a Flexible Probabilistic Model for Embodied MIDI Performance," in *Proceedings of the 1st Conference on AI Music Creativity*, 2022. [Online]. Available: https://zenodo.org/record/7088404

[6] J. Armitage and V. Shepardson, "Augmenting the Expressivity of the Notochord Generative MIDI Model for Arca's The Light Comes in the Name of the Voice" Magnetic Resonator Piano Installation," in *AIMC 2024*, 2024. [Online]. Available: https://aimc2024.pubpub.org/pub/0lh6s86c/release/1

[7] C. Raffel, "Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching," Ph.D. dissertation, Columbia University, 2016.

[1] `https://www.youtube.com/playlist?list=PL8pgdxhelhfi4Hb_AteyT9v358kZ7LmDz`

[2] `notochord_lakh_50G_deep`

[3] `state-spaces/mamba-1.4b-hf`

[4] `https://www.fluidsynth.org`