Embedding Alignment in Code Generation for Audio

Sam Kouteili
Yale University
sam.kouteili@yale.edu

Hiren Madhu
Yale University
hiren.madhu@yale.edu

George Typaldos
Yale University
george.typaldos@yale.edu

Mark Santolucito
Barnard College, Columbia University
msantolu@barnard.edu

Abstract

Large Language Model (LLM) code generation has the potential to enhance creative coding by allowing users to focus on structural and musical motifs rather than syntactic details. For live-coding and other music-oriented settings, users would benefit from diverse candidates that reflect meaningful differences in the resulting audio. However, current models struggle to produce such diversity, as they lack direct insight into the code's sonic output and are typically evaluated using text-based similarity metrics. In this paper, we propose a predictive MLP model that learns an embedding alignment map between code and audio, enabling reasoning about musical similarity directly from code embeddings. This alignment introduces musical awareness into code generation workflows, supporting more perceptually relevant candidate selection and opening the door to musically informed code assistants.

1 Introduction

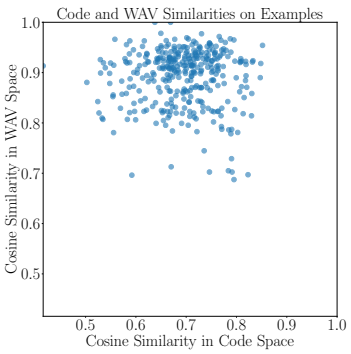
Creative coding endeavors are emerging as a vibrant space at the intersection of art and computation. One such example is *live-coding*, where performers write music-generating code in real-time. This can often be challenging, as performers need to write syntactically correct code under both time constraints and the pressure of an audience.

Recent advances in code generation with LLMs [Jiang et al., 2024, Seo et al., 2025, Tong and Zhang, 2024] present an exciting opportunity for such domains. By lifting much of the syntactic burden, LLMs allow live-coders to focus on higher-level creative motifs and musical ideas. However, existing code generation models struggle to provide diverse code candidates in multi-modal domains, where code output is not text, as they do not possess mechanisms to semantically process multi-modal output [Vasilakis et al., 2024]. Such models typically evaluate candidate outputs using text-based similarity metrics, which do not capture perceptual or semantic audio differences[1].

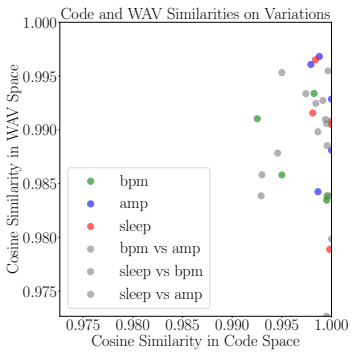
Embedding models offer a potential path forward. Embedding spaces represent meaningful relationships in a given domain by mapping entities to a high-dimensional vector space where "similar" items are less distant. In the context of live-coding, building an alignment map between code and audio would offer a mechanism to reason about musical similarity based only on produced code.

After an initial exploration of the code-audio embedding latent space relationship, we propose a dual Multi-Layer Perceptron (MLP) framework that aligns code and audio embedding spaces. Such a model can provide insights into the topology of the code-audio relationship, helping bridge what an LLM writes and what a user hears. We conduct a study simulating code completion for melody, drum, and bass generation, showing that even on code artifacts with major overlaps, our proposed MLP distinguishes distinct musical semantics only with source code. This highlights our model's potential as a supplement to code completion environments, augmenting them with the ability to reason about code candidates in the auditory domain.

[1] We discuss these techniques more in Appendix Section A.



(a) Similarity between Sonic Pi tutorial entries.



(b) Similarity of small code variations.

Figure 1: Distances between sample embeddings show nontrivial alignment mapping. We extract code and audio embeddings with `distilroberta-base` and `wav2vec2` respectively. Audio is clipped to 9 measures at 120 BPM. Embedding similarity is computed as vector cosine similarity.

2 Preliminary Investigation

As an initial exploration, we investigate the latent relationship between code and audio embedding spaces. For our first study, we select the 27 Sonic Pi tutorial entries and record 27 corresponding audio outputs, collecting code and audio embeddings. We chose Sonic Pi due to its prominence in the live-coding community, terseness, and strong documentation. For each entry (code+audio) in the dataset, we plot code and audio embedding distances to all other entries, comparing code to code and audio to audio. Figure 1a displays our results: looking at our findings, no evident relationship emerges. Low Pearson (0.0159, p=0.6770) and Spearman correlations (0.0409, p=0.4450) affirm no linear or rank-order relationship between the embedding spaces.

We proceed to investigate the sensitivity of code and audio embeddings to small program code modifications. We modify the six longest programs in the tutorials (most parameters to modify) by varying the values of three parameters: sleep time, amplitude (amp), and beats per minute (bpm). We posit that minor code modifications should yield similar code embeddings but varying audio embeddings depending on the altered variable. Changes in amplitude, for example, should affect audio embeddings less than changes in sleep, which may change syncopation. Figure 1b confirms that fuzzed code artifacts maintain a high (> 0.990) code embedding similarity. While similarity is also high in the audio embedding space, the range is larger, with similarity scores below 0.975. Interestingly, there is no evident trend between the modified variable and the resultant audio embedding: sleep, bpm, and amp variations exhibit inconsistent changes in the audio embedding space. The associative, albeit not equivalent, range compression of code and audio embedding distances suggests some coarse association between the domains; however, such an association is not trivial.

3 Model Implementation

To train an embedding alignment model, we utilize the 27 Sonic Pi code entries mentioned in section 2, along with the Jinja template engine [Ronacher, 2008] to augment the dataset and randomize various parameters. A table of parameters used for templating is provided in Table A1, and a template example can be seen in Listing 1. We render 500 different Sonic Pi code files, generating a total of 13,500 entries. We use `distilroberta-base` [Sanh et al., 2019] to generate code embeddings and Meta's `wav2vec2` [Baevski et al., 2020b] for the respective audio embeddings. We adopt a symmetric architecture consisting of two independent MLPs: one for code embeddings $\texttt{MLP}_c$ and one for audio embeddings $\texttt{MLP}_a$. Both networks take as input the respective modality's pretrained embeddings and project them into a common embedding space of dimension d_{out}.

Each MLP consists of L linear layers, with intermediate hidden layers of dimension d_{hidden}, each followed by `BatchNorm` and `GELU` activations. We use `BatchNorm` to stabilize training by normalizing activations across the batch, and `GELU` as the activation function due to its smooth, non-linear behavior that improves gradient flow and empirical performance over ReLU in deep networks. We project the pre-trained code and audio embeddings as $c_i = \texttt{MLP}_c(c_i^0), a_i = \texttt{MLP}_a(a_i^0)$ where, c_i^0

and a_i^0 are the embeddings extracted from the pre-trained model, and c_i and a_i are the aligned embeddings. This formulation was selected to capture non-linear transformations without introducing architectural biases toward either modality. Unlike attention-based architectures, MLPs efficiently map pre-trained embeddings into an aligned representation space.

To train the models, we employ **InfoNCE** loss, a contrastive learning objective that brings semantically aligned code-audio pairs closer while pushing apart mismatched pairs in the same batch [van den Oord et al., 2018]. This choice is motivated by the need for self-supervised alignment, where explicit labels are not available, but semantic consistency can be inferred from pairing. Given a batch of N aligned code-audio embeddings $\{(c_i, a_i)\}_{i=1}^N$, cosine similarity is defined as:

$$\text{sim}(c_i, a_j) = \frac{c_i^\top a_j}{\|c_i\| \cdot \|a_j\|}.$$

The InfoNCE loss $\mathcal{L}_i$ for a single positive pair (c_i, a_i) is shown in Equation 1, with τ being the temperature hyperparameter that controls the sharpness of the similarity distribution. This contrastive formulation applies well to this setting, where each code-audio pair is semantically meaningful but hard supervision is unavailable. InfoNCE encourages the model to preserve pairwise relationships and learn embeddings that are useful for downstream retrieval and matching tasks.

$$\mathcal{L}_i = -\log \frac{\exp(\text{sim}(c_i, a_i)/\tau)}{\sum_{j=1}^N \exp(\text{sim}(c_i, a_j)/\tau)}, \qquad \mathcal{L} = \frac{1}{N} \sum_{i=1}^N \mathcal{L}_i \tag{1}$$

4 Experiments

Hyperparameter Tuning: We quantify alignment between learned representations with two similarity metrics. First, Canonical Correlation Analysis (CCA) measures the maximum linear correlation between two multivariate random variables after projecting them onto a shared subspace. Given code and audio embeddings C and A, CCA finds linear projections that maximize the correlation between Cw_c and Aw_a, with the resulting correlation score reflecting the extent a linear transformation can align the two modalities. Second, Centered Kernel Alignment (CKA) captures similarities between representations in a way that is invariant to orthogonal transformations and isotropic scaling. Unlike CCA, which measures linear alignment, CKA is sensitive to nonlinear structural similarities. CKA operates on kernel matrices K and L derived from embeddings, as shown in Equation 2, where K_c and L_c are centered kernel matrices, and $\langle \cdot, \cdot \rangle_F$ denotes the Frobenius inner product. CKA scores closer to 1 indicate higher structural similarity between representations.

$$\text{CKA}(K, L) = \frac{\langle K_c, L_c \rangle_F}{\|K_c\|_F \cdot \|L_c\|_F}, \tag{2}$$

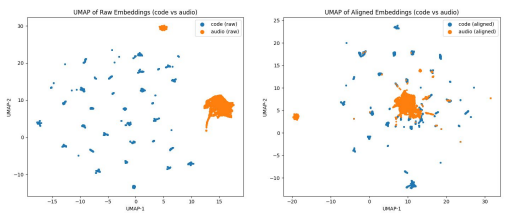
By combining InfoNCE-based contrastive training with post-hoc evaluation using CKA and normalized CCA, we assess the degree of alignment of learned embeddings at both linear and structural levels. We run 24 configurations varying hidden/output dimension, layers, and learning rate, with metrics averaged over five runs: Table A2 reports our results. The first row shows pre-alignment baselines, with low CKA (0.090) and CCA (0.140), indicating minimal correlation between raw embeddings. Post-alignment, the best configuration achieved a CKA of 0.590 (Config. 21) and a normalized CCA of 0.902 (Config. 24), representing over six-fold improvements in both metrics. These gains demonstrate that the model learns a meaningful shared embedding space, enabling reliable approximation of audio embeddings from code even without explicit supervision.

Evaluation: We are interested in examining ability to distinguish sonic similarity of code candidates. We evaluate our model against a raw code baseline (looking at code embedding distance) across three scenarios–**melody**, **drum**, and **bass**–where we simulate LLM-assisted Sonic-Pi code completion for melody, drum pattern, and bassline addition. Table A3 presents our experimental setup. We select the best model and evaluate alignment quality using neighborhood-based metrics such as Jaccard similarity and top-k overlap. These measures assess whether nearest neighbors in the code embedding space correspond to nearest neighbors in the audio embedding space. We report Jaccard, overlap@3, and rank correlations (Spearman, Pearson) in Table 1.

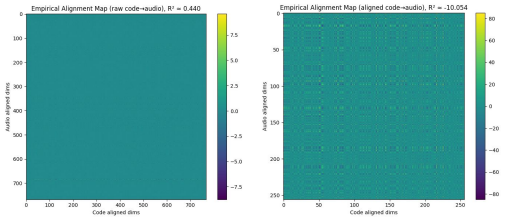
Across all three settings, our method consistently improves neighborhood-based metrics, with the largest gains on **s2-drum** (baseline fails entirely) and **s3-bass** (both selection and correlation improve markedly). Even in **s1-mel**, where fine-grained correlations dip slightly, top-k accuracy improves, highlighting complementary strengths of the evaluation metrics. Importantly, these gains are

Table 1: Comparison of raw baseline vs. our method across three scenarios. Standard deviations for our method are reported in parentheses.

Scenario	Method	Jaccard	Overlap@3	Spearman	Pearson
s1-mel	Raw	0.20	0.33	**0.21**	**0.18**
	Ours	**0.34** (0.21)	**0.47** (0.27)	0.16 (0.17)	0.07 (0.18)
s2-drum	Raw	0.00	0.00		-0.25
	Ours	**0.16** (0.08)	**0.27** (0.13)	-0.05 (0.18)	**-0.12** (0.20)
s3-bass	Raw	0.20	0.33	0.24	0.21
	Ours	**0.50** (0.00)	**0.67** (0.00)	**0.44** (0.05)	**0.46** (0.05)



(a) UMAP visualization of code and audio embeddings before and after alignment.



(b) Empirical alignment heat maps showing code-audio correspondence quality

Figure 2: UMAP and Empirical heat map visualizations demonstrate improved clustering and code-audio matching after alignment training.

achieved *directly from code embeddings*, without compiling audio or extracting audio embeddings, a process that is computationally expensive and time-consuming.

In Figure 2, we illustrate the effectiveness of our model in bridging the semantic gap between code and audio modalities. The UMAP visualizations in Figure 2a reveal that raw embeddings exhibit complete modal separation, with code (blue) and audio (orange) occupying entirely distinct regions of the embedding space. After alignment, we observe overlap between modalities, with audio embeddings clustering within code neighborhoods, demonstrating successful semantic bridging.

Interestingly, the empirical alignment maps in Figure 2b show that while raw embeddings achieve higher linear correlation ($R^2 = -10.054$ to 0.840), this linear fit fails to capture true cross-modal semantics, as evidenced by the poor clustering in UMAP space. Our aligned model sacrifices some linear correlation for semantically meaningful overlap, where related code-audio pairs now occupy shared embedding regions. This nonlinear alignment approach successfully maps semantically related content across modalities into proximate embedding neighborhoods, enabling effective cross-modal retrieval and generation despite reduced linear correlation measures.

5 Conclusion

In this work, we present a code–audio embedding alignment map to bridge the cross-modal semantic gap for code generation models (LLMs). Our preliminary analysis reveals minimal linear or rank-order relationships between the respective latent spaces, motivating the development of a non-linear alignment model. Leveraging Sonic Pi templates, we augment a curated dataset of code–audio embeddings and train a dual MLP model architecture to project these embeddings into a shared latent space. Model evaluation and hyperparameter tuning are quantified using CCA and CKA. Our final experiments, conducted on three distinct live-coding code assistance tasks, demonstrate that the proposed model effectively captures and distinguishes auditory differences among code candidates.

This study presents an initial step toward an integrated framework designed to support live coders with code generation models. Future work will focus on extending this framework by incorporating our alignment map into a generative code assistant environment to better realize artistic intent.

References

Samuel Aaron and Alan F. Blackwell. Sonic pi: A platform for real-time music coding. *Computer Music Journal*, 48(1), 2024. doi: 10.1162/comj_a_00512.

A. Baevski, H. Zhou, A. Mohamed, and M. Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. In *Advances in Neural Information Processing Systems (NeurIPS 2020)*, volume 33, 2020a.

A. Baevski, H. Zhou, A. Mohamed, and M. Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in Neural Information Processing Systems (NeurIPS)*, 33, 2020b. URL `https://tidalcycles.org`.

Russa Biswas, Mehwish Alam, and Harald Sack. Is aligning embedding spaces a challenging task? an analysis of the existing methods. *CoRR*, 2020. URL `arxiv.org/abs/2002.09247`.

Aurora Cramer, Ho-Hsiang Wu, Justin Salamon, and Juan Pablo Bello. Look, listen and learn more: Design choices for deep audio embeddings. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019.

Bhavika Devnani, Skyler Seto, Zakaria Aldeneh, Alessandro Toso, Elena Menyaylenko, Barry-John Theobald, Jonathan Sheaffer, and Miguel Sarabia. Learning spatially-aware language and audio embeddings, 2024. URL `arxiv.org/abs/2409.11369`.

Benjamin Elizalde, Shuayb Zarar, and Bhiksha Raj. Cross modal audio search and retrieval with joint embeddings based on text and audio. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019. doi: 10.1109/ICASSP.2019.8682632.

Aysegul Ozkaya Eren and Mustafa Sert. Audio captioning based on combined audio and semantic embeddings. In *2020 IEEE International Symposium on Multimedia (ISM)*, 2020. doi: 10.1109/ISM.2020.00014.

GitHubCopilot. Github copilot. `github.com/features/copilot`, 2024. Accessed: 2024-12-26.

Qingqing Huang, Aren Jansen, Joonseok Lee, Ravi Ganti, Judith Yue Li, and Daniel P. W. Ellis. Mulan: A joint embedding of music audio and natural language, 2022. URL `arxiv.org/abs/2208.12415`.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*, 2024.

Yu-Chen Lin, Akhilesh Kumar, Norman Chang, Wenliang Zhang, Muhammad Zakir, Rucha Apte, Haiyang He, Chao Wang, and Jyh-Shing Roger Jang. Novel preprocessing technique for data embedding in engineering code generation using llm. In *2024 IEEE LLM Aided Design Workshop (LAD)*, 2024. doi: 10.1109/LAD62341.2024.10691715.

et al. Mark Chen. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

James McCartney. Rethinking the computer music language: Super collider. *Computer Music Journal*, 26(4), 2002. ISSN 01489267, 15315169. URL `http://www.jstor.org/stable/3681770`.

Alex McLean. Tidalcycles: A language for live coding pattern-based music. In *Proceedings of the International Computer Music Conference*, 2012.

Alex McLean. Strudel repl, 2022. URL `https://strudel.cc/`. [Online; accessed 29-August-2025].

Yann Orlarey, Dominique Fober, and Stéphane Letz. FAUST : an Efficient Functional Approach to DSP Programming. In Editions DELATOUR FRANCE, editor, *NEW COMPUTATIONAL PARADIGMS FOR COMPUTER MUSIC*. 2009. URL `https://hal.science/hal-02159014`.

Shuyin OuYang, Jie M. Zhang, Mark Harman, and Meng Wang. The impact of non-determinism in large language models on code generation and software engineering tasks. *arXiv:2308.02828*, 2024. URL arxiv.org/abs/2308.02828.

Armin Ronacher. Jinja, 2008. URL https://jinja.palletsprojects.com.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108, 2019.

Minju Seo, Jinheon Baek, Seongyun Lee, and Sung Ju Hwang. Paper2code: Automating code generation from scientific papers in machine learning. *arXiv preprint arXiv:2504.17192*, 2025.

Weixi Tong and Tianyi Zhang. Codejudge:evaluating code generation with large language models, October 2024.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. In *Advances in Neural Information Processing Systems*, 2018.

Yannis Vasilakis, Rachel Bittner, and Johan Pauwels. Evaluation of pretrained language models on music understanding, 2024. URL https://arxiv.org/abs/2409.11449.

Josch Wang. A case study on llm code generation in sonic pi and its impact on student attitudes towards computer science. *International Journal of Secondary Computing and Applications Research*, 2024. doi: 10.5281/zenodo.14279494.

A Related Work

A.1 Code Generation Models

With the rise of large language models (LLMs), systems such as GitHub's Copilot GitHubCopilot and OpenAI's Codex Mark Chen [2021] have demonstrated strong code generation capabilities for complex programming tasks, having been trained on extensive code repositories and fine-tuned to capture code semantics. Code generation models are typically evaluated on competitive programming tasks, with the HumanEval dataset Mark Chen [2021] serving as one of the predominant benchmarks. Recent research has expanded these efforts to domains such as music computing and the study of coding-related sentiments Wang [2024]. However, given the inherent non-determinism of LLMs, ensuring code correctness and consistency remains a persistent challenge OuYang et al. [2024]. These challenges are further exacerbated in creative coding contexts, where limited training data exist for certain domain-specific languages (DSLs). Moreover, LLMs have been shown to struggle with music comprehension Vasilakis et al. [2024], suggesting corresponding difficulties in evaluating code for computer music generation.

A.2 Music Programming Languages

Many modern music programming languages take the form of DSLs (domain specific languages) - languages that are designed for a specific application domain. One such example is SuperCollider McCartney [2002], an audio programming language and environment for real-time audio synthesis and algorithmic composition. Conversely, FAUST (Functional AUdio STream) Orlarey et al. [2009] is a purely functional programming language for real-time signal processing. Live coding languages are a subset of music programming languages tailored for live music performance. One example is Sonic Pi Aaron and Blackwell [2024], a live coding language built on Ruby that has been prominently adopted by the community. Sonic Pi has shown promise in educational settings, introducing students to computer science concepts through real-time music coding. Tidal Cycles McLean [2012] is another functional alternative built on the Haskell functional programming language; it offers programmers a declarative approach to live coding. Strudel McLean [2022] is a variant of Tidal Cycles built on JavaScript garnering notable community adoption.

A.3 Embedding Space Alignment

Embedding models attempt to produce learned dimensional representations of data in a hyperplane. Pre-trained embedding models map data - text, images, audio, programs - into vector spaces that encode relational semantics. Program embedding models have been applied to augment code-classification and auto-completion Lin et al. [2024]. Similarly, audio embedding models have been considered for speech recognition, music generation, and audio classification Eren and Sert [2020]. Audio embedding models capture acoustic features and temporal dependencies, with different models highlighting different auditory features Baevski et al. [2020a], Cramer et al. [2019].

Given two distinct embedding latent spaces, one may inquire about an alignment mapping relationship. Early alignment methods applied to language and knowledge graphs analytically established mappings between these spaces; however, recent efforts on more complex maps have employed unsupervised methods Biswas et al. [2020]. With the rise of multi-modal LLMs, cross-modal alignment has begun to appear in audiovisual domains Elizalde et al. [2019]. Novel works have explored ways of encoding linguistic semantics in audio embeddings, and have even presented joint embedding spaces between the two fields Devnani et al. [2024], Huang et al. [2022].

B Dataset Augmentation

B.1 Templating Parameters

Table A1: Parameters used for templating

Parameters	Example Values
samples	ambi_choir, bd_haus
synths	beep, rodeo
character	major, minor
attack/release_range	[0, 10]
amp_range	[0, 10]
sleep_range	[0.1, 5.0]
effects	echo, compressor
notes	C2, Db2, . . ., C6

B.2 Sample Templated Dataset Entry

Listing 1: Jinja Template used for parametrising Compus Beats dataset entry

```
# Compus Beats

# Coded by Sam Aaron

use_sample_bpm :{{samples_bpm[0]}}, num_beats: {{repeat_small_ints
    [0]}}

live_loop :loopr do
  sample :{{samples_bpm[0]}}, rate: [0.5, 1, 1, 1, 1, 2].choose unless
       one_in(10)
  sleep {{sleep_values[0]}}
end

live_loop :bass do
  sample :{{sample_values[0]}}, amp: rrand(0.1, 0.2), rate: [0.5, 0.5,
       1, 1,2,4].choose if one_in(4)
  use_synth :{{synth_values[0]}}
  use_synth_defaults mod_invert_wave: 1
  play :{{note_values[0]}}, mod_range: 12, amp: rrand(0.5, 1),
      mod_phase: [0.25, 0.5, 1].choose, release: {{release_values[(1)%
       release_values|length]}}, cutoff: rrand(50, 90)
  play :{{note_values[(1) % note_values|length]}}, mod_range: [24, 36,
       34].choose, amp: {{amp_values[0]}}, mod_phase: 0.25, release:
      {{release_values[0]}}, cutoff: {{repeat_large_ints[0]}},
      pulse_width: rand
  sleep {{sleep_values[(1)% sleep_values|length]}}
end
```

C Hyperparameter Tuning

Table A2: CCA and CKA comparison across hyperparameters. First row shows pre-alignment metrics. Best post-alignment results are in **bold** (1st) and underlined (2nd).

Config	d_{hidden}	d_{out}	**L**	**LR**	**CKA**	**CCA**
–	–	–	–	Before training	0.090 ± 0.001	0.145 ± 0.003
1	256	128	5	1e-4	0.420 ± 0.019	0.523 ± 0.021
2	128	64	1	1e-3	0.455 ± 0.004	0.480 ± 0.004
3	128	64	1	1e-4	0.463 ± 0.011	0.378 ± 0.011
4	128	64	3	1e-3	0.424 ± 0.024	0.510 ± 0.019
5	128	64	3	1e-4	0.398 ± 0.021	0.372 ± 0.010
6	128	64	5	1e-3	0.422 ± 0.014	0.552 ± 0.009
7	128	64	5	1e-4	0.357 ± 0.040	0.396 ± 0.011
8	128	128	1	1e-3	0.472 ± 0.057	0.660 ± 0.021
9	128	128	1	1e-4	0.490 ± 0.033	0.522 ± 0.010
10	128	128	3	1e-3	0.494 ± 0.017	0.691 ± 0.010
11	128	128	3	1e-4	0.459 ± 0.031	0.547 ± 0.003
12	128	128	5	1e-3	0.410 ± 0.023	0.735 ± 0.013
13	128	128	5	1e-4	0.407 ± 0.055	0.571 ± 0.013
14	256	64	1	1e-3	0.468 ± 0.063	0.499 ± 0.011
15	256	64	1	1e-4	0.514 ± 0.014	0.357 ± 0.005
16	256	64	3	1e-3	0.461 ± 0.035	0.556 ± 0.007
17	256	64	3	1e-4	0.454 ± 0.027	0.366 ± 0.006
18	256	64	5	1e-3	0.432 ± 0.005	0.644 ± 0.013
19	256	64	5	1e-4	0.386 ± 0.014	0.372 ± 0.010
20	256	128	1	1e-3	0.444 ± 0.042	0.736 ± 0.034
21	256	128	1	1e-4	$\mathbf{0.590 \pm 0.044}$	0.486 ± 0.007
22	256	128	3	1e-3	0.444 ± 0.021	$\underline{0.743 \pm 0.045}$
23	256	128	3	1e-4	$\underline{0.548 \pm 0.033}$	0.493 ± 0.005
24	256	128	5	1e-3	0.466 ± 0.007	$\mathbf{0.902 \pm 0.007}$

D Experimental Setup

Table A3: Code snippets and prompts used for results. For each snippet, GPT-5 generated 3 candidate and 10 candidate completions. The 10 candidates experienced a wider auditory variance, motivating the value of greater candidate generation and subsequent pruning with an embedding model. Our model successfully predicts the most sonically distinct entries with just the code embeddings.

	s1-mel	**s2-drum**	**s3-bass**

Code	`use_bpm 100`
	`# Drums`
	`live_loop :drums do`
	`  sample :bd_haus`
	`  sleep 0.5`
	`  sample :sn_dolf`
	`  sleep 0.5`
	`end`
	`live_loop :hats do`
	`  sleep 0.25`
	`  sample`
	`  :drum_cymbal`
	`end`
	`# Bassline`
	`live_loop :bass do`
	`  use_synth :fm`
	`  play_pattern_timed`
	`  [:e2, :g2,`
	`  :a2, :g2],`
	`  [0.5, 0.5,`
	`  0.5, 0.5],`
	`  release: 0.25`
	`end`
		`use_bpm 90`
		`# Melody`
		`live_loop :melody do`
		`  use_synth :prophet`
		`  play_pattern_timed`
		`  [:c4, :e4,`
		`  :g4, :e4],`
		`  [0.5, 0.5,`
		`  0.5, 0.5],`
		`  release: 0.3`
		`end`
		`# Bassline`
		`live_loop :bass do`
		`  use_synth :fm`
		`  play_pattern`
		`  [:g3, :g3,`
		`  :d4, :g3],`
		`  release: 0.5`
		`end`
			`use_bpm 100`
			`# Drums`
			`live_loop :drums do`
			`  sample :bd_haus`
			`  sleep 1`
			`  sample :sn_dolf`
			`  sleep 1`
			`end`
			`# Melody`
			`melody = [:e4,`
			`  :g4, :a4, :b4,`
			`  :a4, :g4, :e4,`
			`  :d4].ring`
			`live_loop :melody`
			`  do`
			`  use_synth :pulse`
			`  play`
			`  melody.tick,`
			`  release: 0.3`
			`  sleep 0.5`
			`end`

Prompt	Propose $\{3, 10\}$ melodies to accompany this code	Propose $\{3, 10\}$ drum patterns to accompany this code	Propose $\{3, 10\}$ bass lines to accompany this code