# Cibo v2: Realtime Livecoding A.I. Agent

Jeremy Stewart
Rensselaer Polytechnic Institute, Troy, NY
stewaj5@rpi.edu

Shawn Lawson
Rensselaer Polytechnic Institute, Troy, NY
lawsos2@rpi.edu

Mike Hodnick
mike@kindohm.com

Ben Gold
bgold.cosmos@gmail.com

## Abstract

Cibo v2 is a live-coding artificial intelligence (AI) agent that performs TidalCycles and is trained on recorded performances by several TidalCycles performers. This paper presents an entirely new architecture from the original Cibo agent for realizing autonomous performing agents.

## Introduction

Cibo v2 represents an entirely new architecture from the previously presented Cibo live-coding agent (Stewart and Lawson, 2019; Stewart, 2019, Cibo: Safeguard II). Cibo v2 performs TidalCycles (McLean and Wiggins, 2010; McLean et al, 2019) code in a live-coding setting for musical and sound performance. Cibo v2 is constructed with autoencoder and variational autoencoder architectures as its foundations, with additional neural network modules governing performance progression and variable production. Cibo v2 trains faster and is trained in a number of steps, allowing for a greater degree of flexibility during the development process. The resulting performance agent produces TidalCycles code that is highly reminiscent of the provided training material, while offering a unique, non-human interpretation of TidalCycles performances. Cibo v2 is trained using recordings of performances by several human performers, including Mike Hodnick, Ben Gold, and Jeremy Stewart. For the first time, the agent in performance hints at each of these influences. Furthermore, the manner by which the agent is constructed allows for visualizing the organization of training material, allowing us to peer into the learned AI-representation of these recordings. Finally, because each human contributor performs with a unique sample corpus, a sample analysis tool set is offered to allow the Cibo v2 agent to effectively substitute samples based on sonic features.

## Pre-Agent Technical Overview

The first step to creating the Cibo agent include recording TidalCycles performances by human performers and tokenizing the TidalCycles code to prepare it for training purposes. The following section outlines the metholodies and software developed to satisfy these functions.

## Recording Tidalcycles(Sublime JENSAARAI)

Jensaarai, a custom text editor written in NodeJS with Electron, was integral to the initial version of Cibo. Jensaarai created text recordings of all edits and code executions that a human performer would make. These recorders were fed into Cibo for its training process. When Cibo performs, Cibo's edits and code executions are made visible in the Jensaarai editor. Cibo v2 uses exactly the same process, with several differences in the text editor. Human performers making text recordings and Cibo v2 performances use Sublime Text with the Sublime Jensaarai plugin (Lawson, 2019). Using this plugin format reduced numerous issues that were occurring when the userbase of Jensaarai grew beyond the primary developers and Cibo. The functionalities of the Sublime Jensaarai plugin are nearly identical to the standalone Jensaarai application.

## LEXER/TOKENIZATION

Tokenization of TidalCycles code is carried out using the PLY library (Beazley, 2018), much like was described in "Cibo: An Autonomous TidalCycles Performer" (Stewart and Lawson, 2019). The lexer dictionary contains 248 discrete tokens, including all of those available in the TidalCycles documentation (All the Functions - TidalCycles), as well as additional tokens for custom function definitions. Variable values are stored in a second vector (represented as the "values" vector in Figure 1), and are replaced with INTEGER, FLOAT, and STRING tokens.

We then convert this single token representation into an n-gram encoding, combining up to four tokens into a single integer representation, as seen in Figure 2, where the tokens [0, 30, 17, 28] are combined into a single token: 40. Only n-gram tokens occurring in the training material are preserved, resulting in a dictionary of 11267 discrete tokens—much fewer than the possible 2484 tokens. These n-gram tokens are used as the input and output of the AutoEncoder architectures described below.

## Agent Architecture

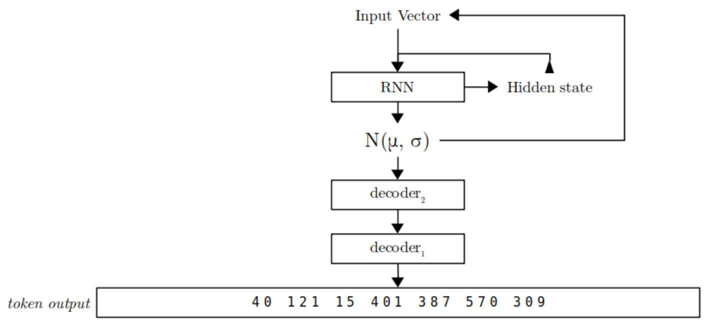


Figure 3: CiboV2 performance-ready architecture. Input vector is a three-dimensional vector denoting the latent space coordinates which are to be decoded by two stages of decoder module, resulting in n-gram output

The Cibo agent is trained, first, to construct compact encodings of TidalCycles code via an autoencoder and a variational autoencoder. The training of these neural networks produces a three-dimensional latent space: TidalCycles code can be converted into a point in this latent space, and, inversely, coordinates in this latent space can be decoded into usable TidalCycles code. The Cibo agent performs by traversing the three-dimensional latent space using a recurrent neural network (RNN), producing TidalCycles code at each step. The

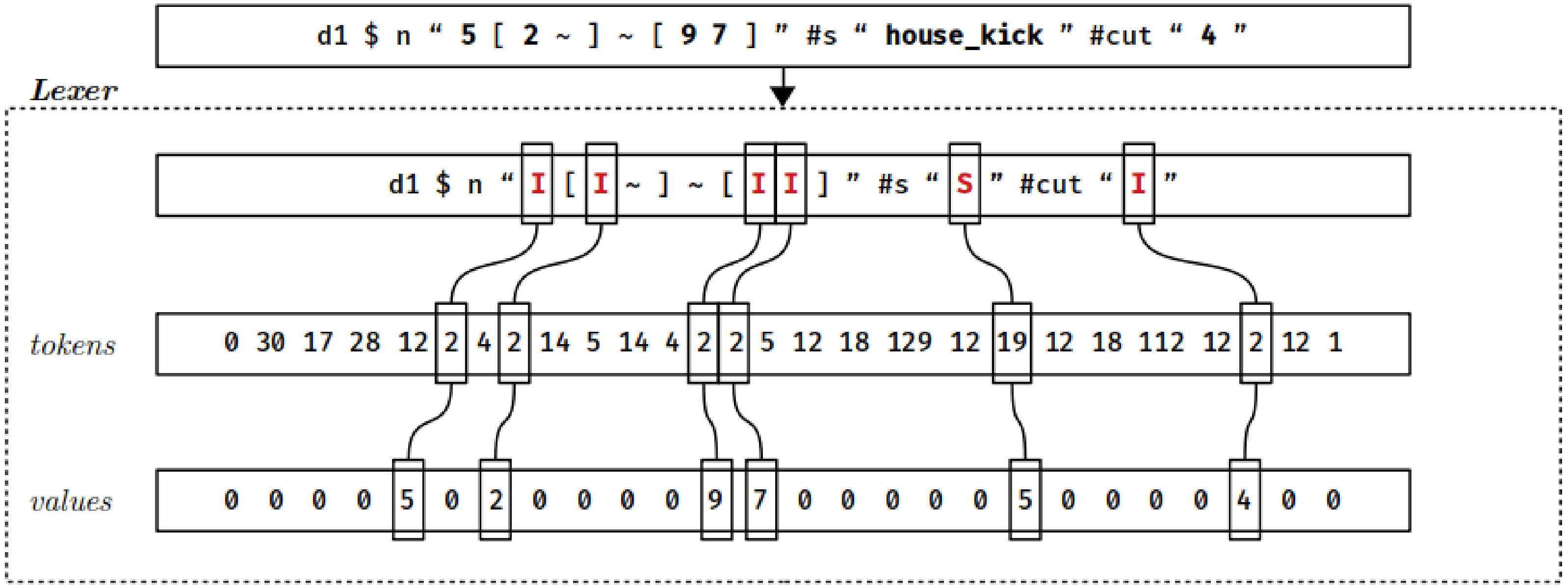


Figure 1: TidalCycles code is input into the lexer, which lexes the code in discrete tokens. Integer, Float, and String tokens are indicated as such, while preserving the values associated with these tokens in a second vector.

RNN module, seen at the top of Figure 3, is trained to traverse the resultant latent space in a manner consistent with the TidalCycles recordings used for training. Importantly, this module outputs normal distributions which are sampled, resulting in non-deterministic behavior. The basic performance-ready architecture is represented in Figure 3. “Decoder1”, seen at bottom, is the first component to be trained, followed by “decoder2”. These two modules together convert a 3-dimensional latent space vector into an n-gram token sequence which can be converted to TidalCycles code. The following sections will detail each of the neural network components of the Cibo agent.

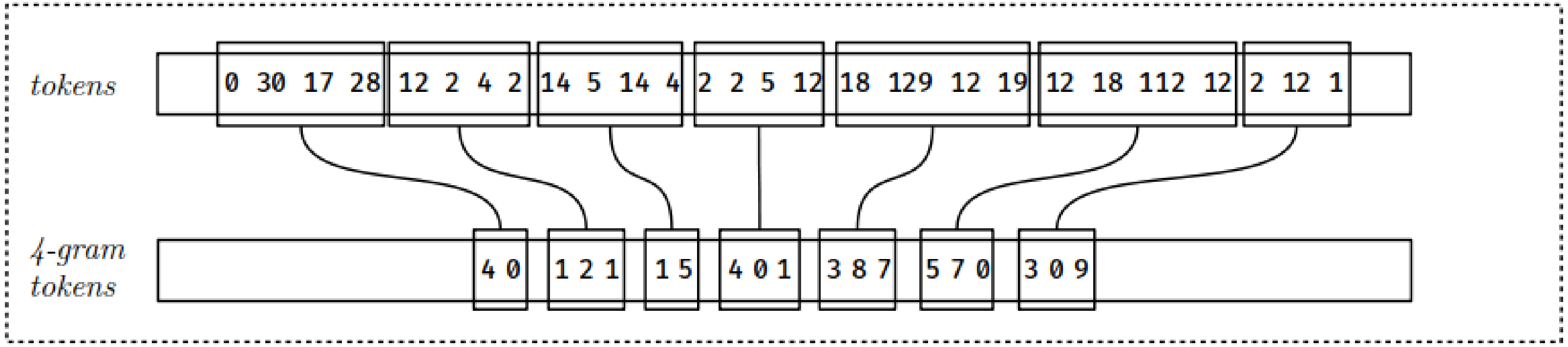


Figure 2: Tokens are combined into 4-gram representations.

## Autoencoder

Once n-gram token sequences are produced by the lexer, the first stage of the Cibo agent must convert these variable-length sequences into a fixed-length representation. In order to accomplish these, we employ an autoencoder architecture, seen in Figure 4.

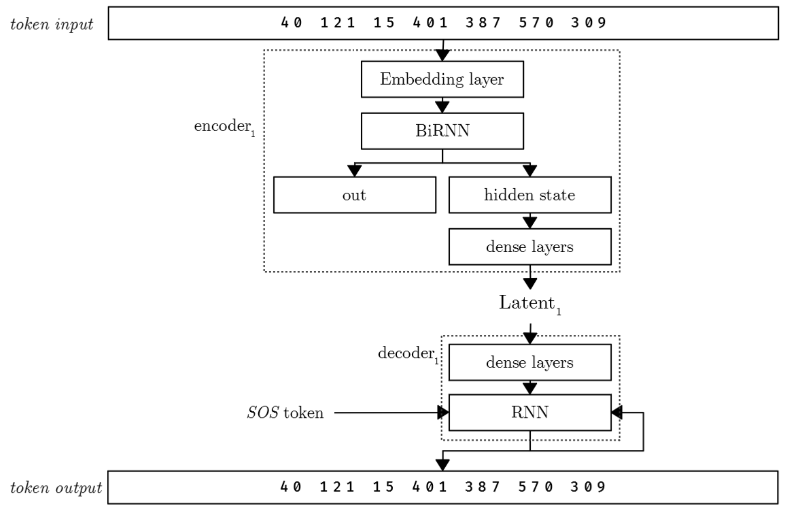


Figure 4: Training the first stage autoencoder neural network. Variable length input (n-gram token input, at top) is converted to a fixed-length latent representation (Latent1), and then decoded back to a variable length output (token output, bottom).

The token input passes, first, through an embedding layer, which converts the discrete, integer token notation into a floating-point vector in a higher-dimensional space. The output from the embedding layer is sent to a bidirectional recurrent neural network (BiRNN) which reads the sequence both forward and backward, producing two outputs: a variable-length out vector and a fixed-length hidden state vector. The out vector is discarded. The hidden state vector is passed through a fully connected (dense) layer in order compress the encoding further, resulting in a vector that contains 500 values: $\text{Latent}_1$ (This $\text{Latent}_1$ vector will be used to train the subsequent VAE module, discussed in the next section). The latent vector is then fed into the decoder module, which is trained to convert it back to the original input sequence, thus token input and token output are the same sequence.

## Variational Autoencoder

With the first stage autoencoder complete and trained, we construct a variational autoencoder (Kingma et al, 2013) to further compress the latent representations of TidalCycles code. The variational autoencoder (VAE) is constructed by alternating linear (fully connected) layers with ReLU (Rectified Linear Unit) activation functions. The encoder, labeled $\text{encoder}_2$ in Figure 5, outputs two vectors which are used as the mean and standard deviation of a normal distribution (seen at center of Figure 5). This normal distribution is then sampled and input into the decoder (decoder2) which reconstructs the $\text{Latent}_1$ vector.

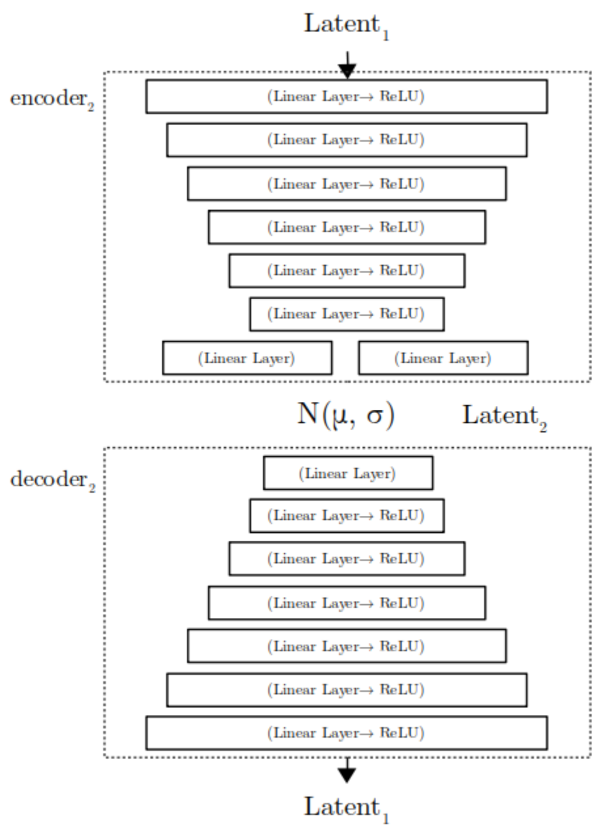


Figure 5: Training the second stage variational autoencoder (VAE). The learned latent representation from stage 1 (seen in Figure 4) is used as input and output here. The encoder module produces a normal distribution which is sampled before being sent to the decoder, resulting in non-deterministic behavior.

During the development of the Cibo agent and many training experiments, many different hyperparameter settings we tested, including different numbers of layers, different numbers of hidden features between each layer, and different activation functions. Figure 5 accurately represents the architecture as it is currently in use, with 7 layers in each the encoder and decoder. The Latent1 input/target vector contains 500 features, while the innermost latent representation—the result of sampling the normal distribution (and the input into decoder2)—contains 3 features.

The variational autoencoder provides two strengths in this implementation. First, because the encoder (encoder2) produces a normal distribution which is then sampled, the VAE is non-deterministic and retains a degree of stochasticity in its final execution. Additionally, the resulting latent space produces gradual changes in the output, i.e., the latent space, while highly complex and knotted, can be smoothly traversed for continuous subtle changes in output.

## Latent Space Traversal

*

Once the latent space has been constructed by training the autoencoder and subsequent VAE (discussed above), all training recordings are passed through the encoders, producing a normal distribution for each training sample. Once these latent normal distributions are generated, we can visualize the latent space (see Figure 6). The three TidalCycles contributors—blindelephants (Jeremy Stewart), kindohm (Mike Hodnick), and bgold (Ben Gold)—are signified through different colors, with each point representing an execution of a line or block of TidalCycles code. Figure 6 was produced via t-SNE using the latent-space vectors produced by the encoders (Maaten et al, 2008). There are clear regions in this visualization, with groupings of the same color in some areas, and mixing in others.

With this latent space constructed by the two encoder neural networks, all training recordings were encoded, resulting in a sequence of normal distributions for each recording. More simply put, with the latent space constructed as in Figure 6, it is possible to trace pathways through this space which represent a given performance. These sequences were used to train a recurrent neural network (using a LSTM architecture; Hochreiter, 1997) in an autoregressive fashion. The resulting model will trace a pathway through the latent space (in Figure 6) in a manner based on the training recordings, at each step producing a vector which can be sent to the two decoder modules and output as TidalCycles code.

## Generating Token Values

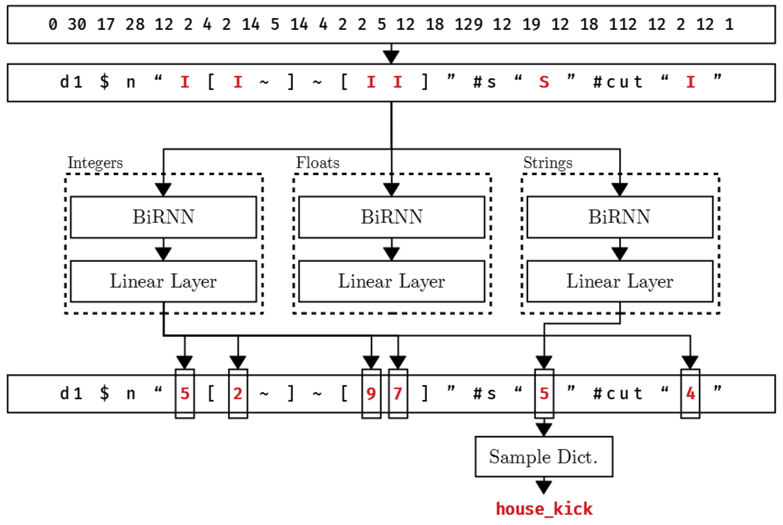


Figure 7: Generation of token values takes place by 3 neural networks, one for each variable type: Integer, Float, and String

The token sequence that is generated by the agent modules discussed above will contain Integer, Float, and String tokens, without yet specifying their values. The last step in the Cibo agent's generative process is to produce these variables. It does so with three neural networks, each dedicated to one of the variable types. The

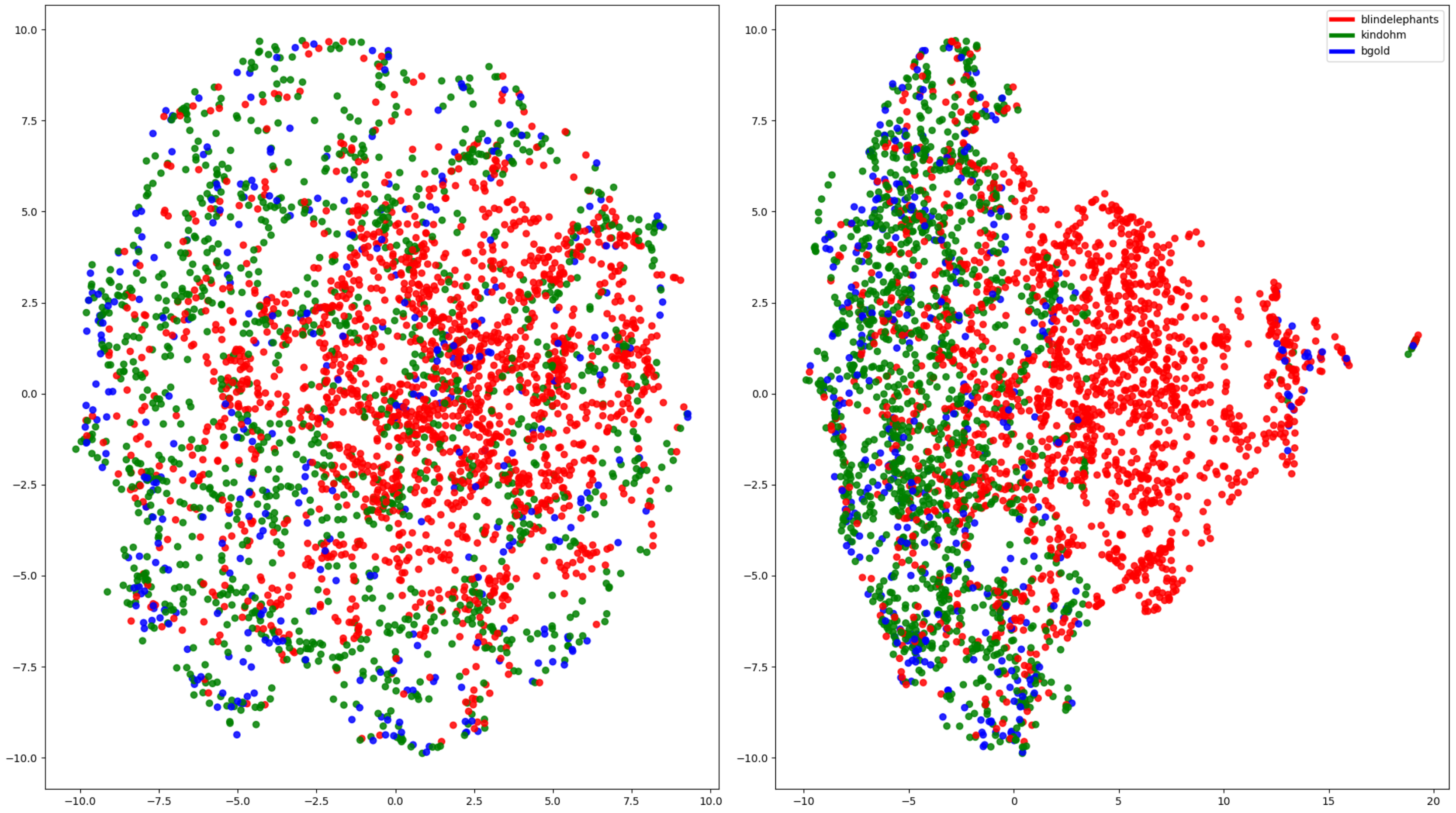


Figure 6: The latent space produced by training, visualization aided by t-SNE processing. Each point represents a single execution of a TidalCycles line or block. The two subfigures represent the same space, rotated by 90 degrees around the y-axis.

TidalCycles token sequence is read into a bidirectional recurrent neural network (BiRNN) which generates a variable length output (equal to the sequence length). Indices of the given variable type are then passed through a linear layer which outputs a single value. If the variable type is float, the output is used directly; integer type will result in the output being truncated and used. If the variable is of string type, the value will be used to look up a sample name from an available dictionary which contains all currently available samples.

## Sound Analysis-Based Substitution

TidalCycles and SuperDirt make using one's own sample library very straightforward. While recording performances to use for training the Cibo agent, each contributor used a unique sample library—potentially containing a combination of publicly available sound samples (such as those included with SuperDirt) and samples created by each individual artist. While the agent is trained in a manner that preserves all of these unique sample names and possibilities, there is no guarantee that all of these samples will be available at the time of performance. Therefore, we developed a small tool set (Stewart, 2019, Sample Corpus Sound Analysis) to analyze each sample, producing a fixed-size vector that represents various spectral features, including: centroid, bandwidth, contrast, flatness, and rolloff; as well as a full STFT (short time fourier transform) and CQT (constant Q transform). This analysis tool uses the Librosa audio analysis library. From each of these analyses (the length of which would be based on the duration of the sample) a mean, standard deviation, and sum vector was calculated. Thus, each sample is represented by a vector containing 3360 values.

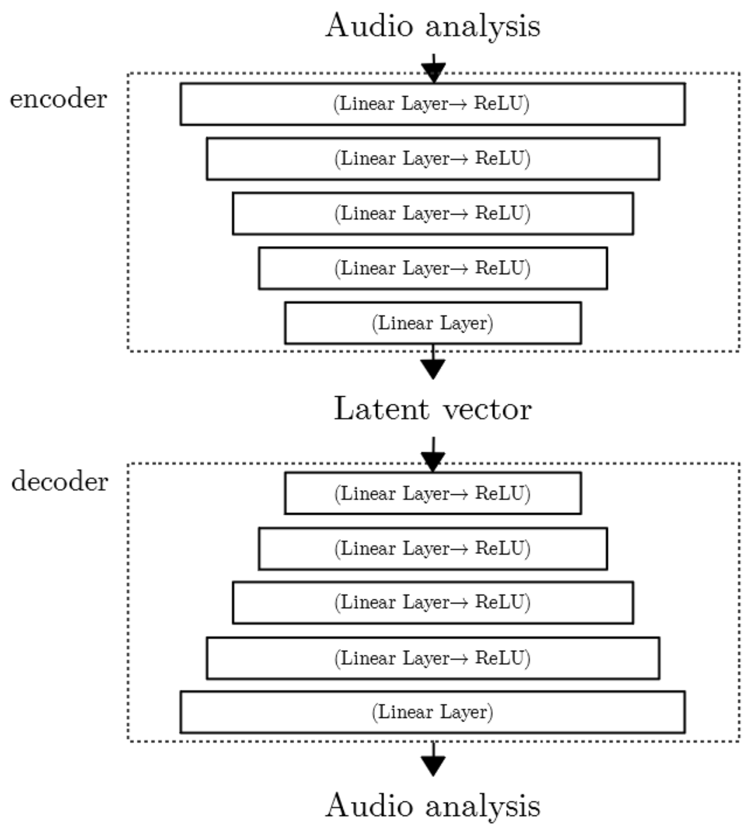


Figure 8: Dimensionality reduction performed on sample audio analyses with an autoencoder.

An autoencoder neural network was constructed for the sake of dimensionality reduction, allowing us to effectively reduce each sample's representation from 3360 values to 300 values. This neural network, seen in Figure 8, contains five layers in each the encoder and decoder.

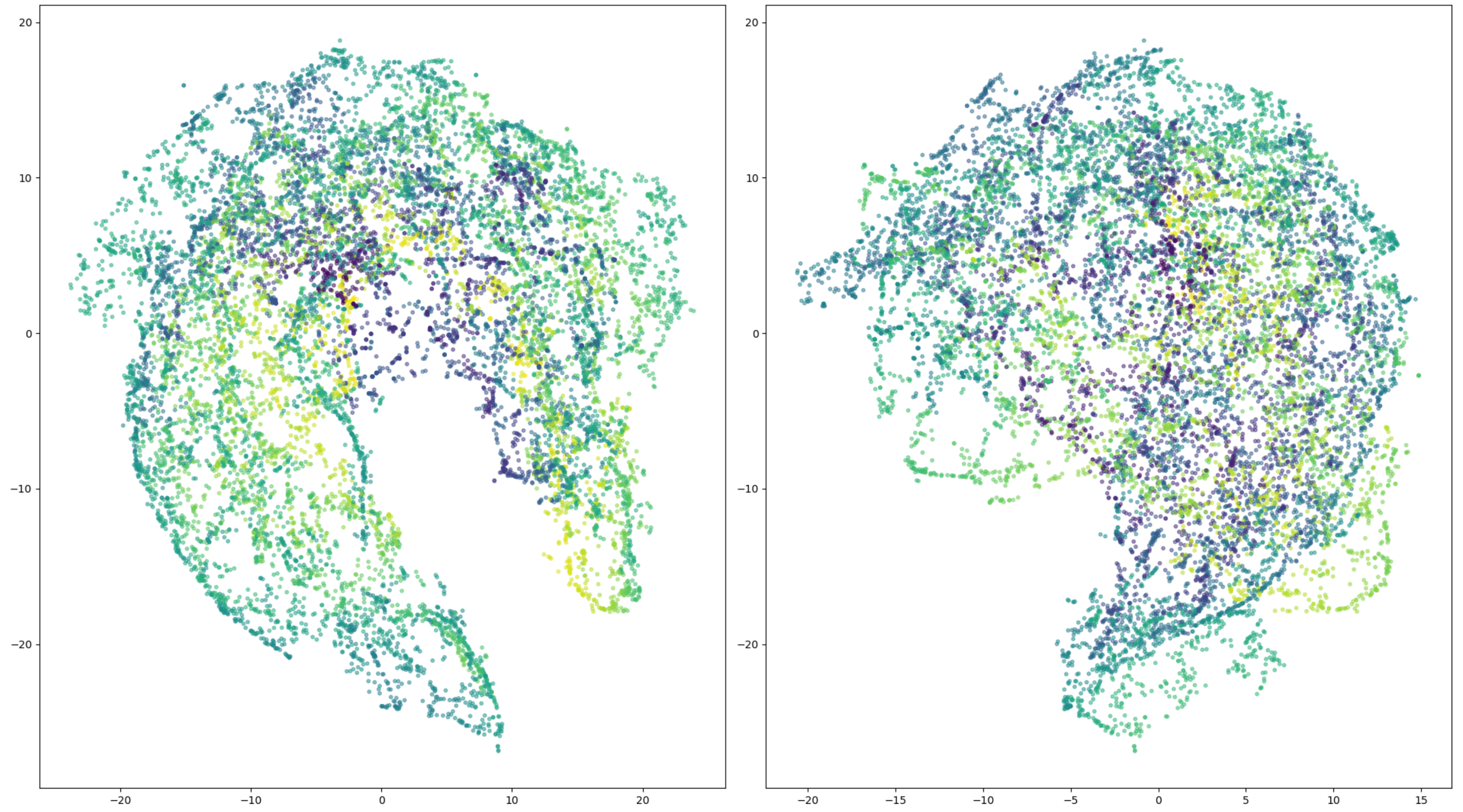
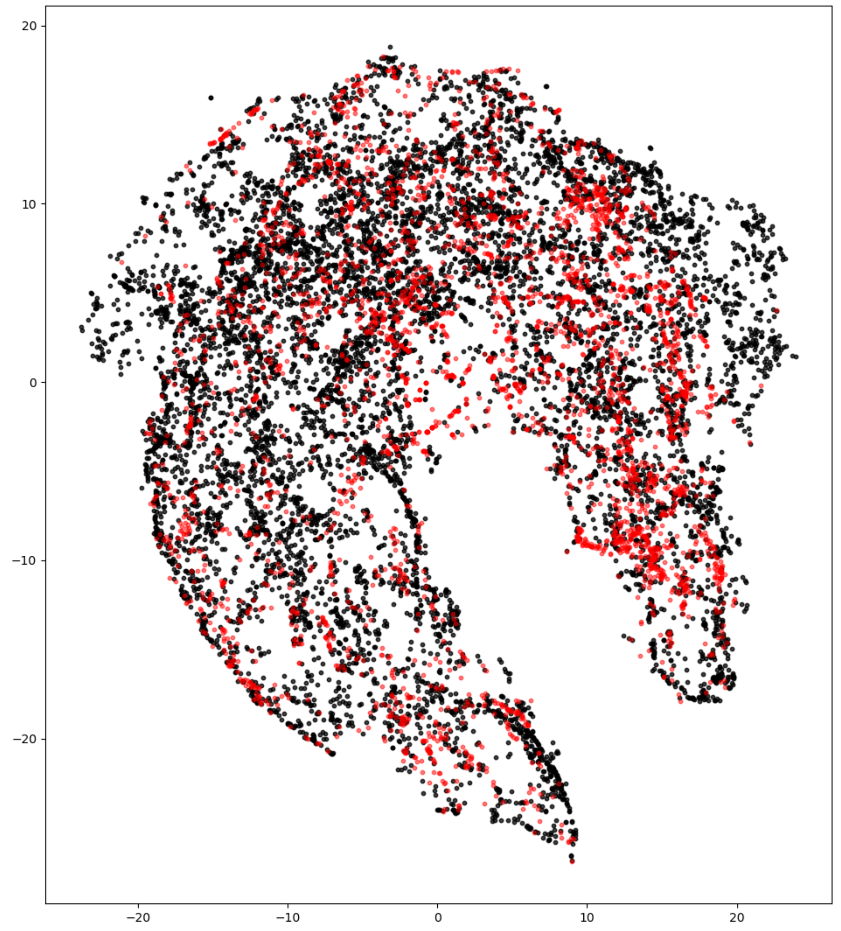


Figure 9: t-SNE visualization of the latent space produced by the autoencoder. Each point represents a single sample, with 13340 in all. The two subfigures are of the same space, rotated 90 degrees on the y-axis. Z-depth is represented by color.



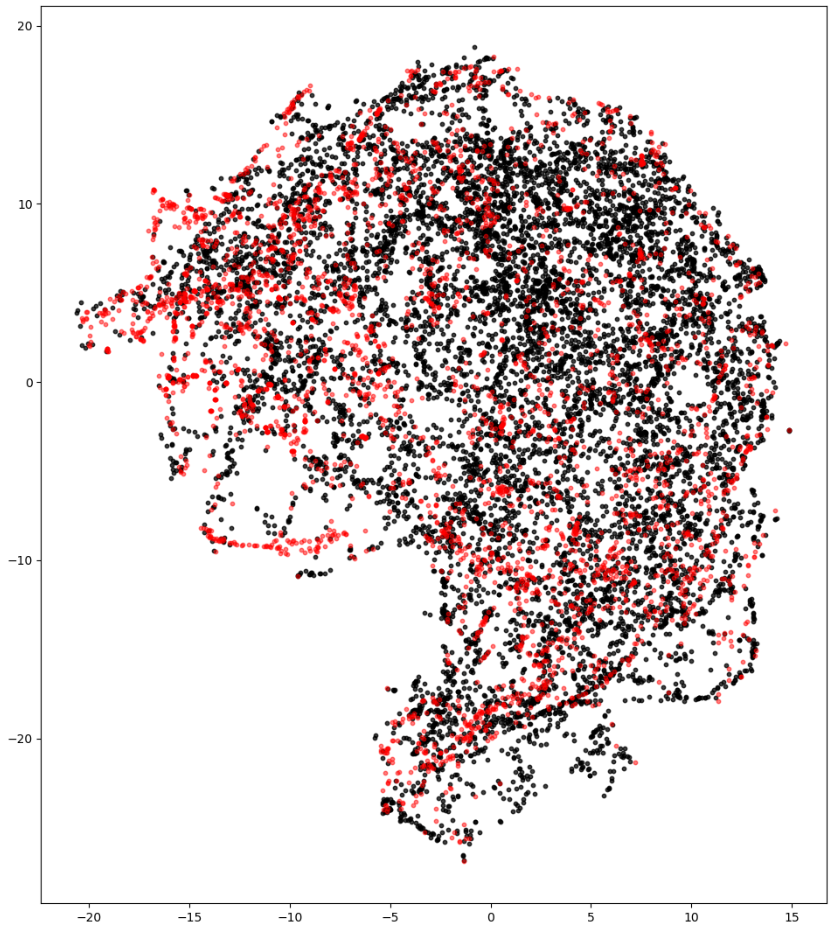


Figure 10: Here, the mapping is made more apparent. Red points indicate currently unavailable samples, while black points are those sample currently available. The substitution method described here will map each red point to the nearest black point.

*

From here, a table is produced that maps all unavailable samples to available samples, based on proximity of latent vectors. This table is stored for use by the agent during performance. If the agent produces a sample name which is not in the currently available sample library, this table is used for a simple name substitution.

*

All sample analyses (those containing 3360 values each) are preserved, allowing for recalculation of the substitution tables if the available sample library changes or if new training recordings, referencing additional samples, become available.

## Results

The Cibo v2 agent, as described above, performs as a live-coding performer using TidalCycles code in a self-directing manner based on learned characteristics from training material. Video documentation of the agent's performance (Stewart, 2019, Cibo V2 Demo) demonstrates Cibo v2's ability to produce valid TidalCycles code in way that is highly reminiscent of the contributing performers, while also producing novel output that effectively blends between these influences.

## Conclusions

Cibo v2 represents a novel implementation of artificial intelligence (AI) and machine learning (ML) techniques to create an autonomous live-coding performance agent. The construction of Cibo v2 around autoencoder and variational autoencoder modules results in a faster training process (than previous versions of the Cibo agent, specifically) and more robust sequence generation. Additionally, the ability to visualize the learned latent space (Figure 6) allows us to begin to develop a better understanding and conceptualization of how TidalCycles performances might be organized and correlated.

## References

All the functions – TidalCycles, viewed 25 September 2019, `https://tidalcycles.org/index.php/All_the_functions`

Beazley, D., 2018. PLY (Python-Lex-Yacc), viewed 25 September 2019, `http://www.dabeaz.com/ply/`

Bingham, E., Chen, J.P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R.,

Szerlip, P., Horsfall, P. and Goodman, N.D., 2019. Pyro: Deep universal probabilistic programming. The Journal of Machine Learning Research, 20(1), pp.973-978.

Hochreiter, S. and Schmidhuber, J., 1997. Long short-term memory. Neural computation, 9(8), pp.1735-1780.

Kingma, D.P. and Welling, M., 2013. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114.

Lawson, S., 2019. Sublime Jensaarai, viewed 26 September 2019, `https://github.com/shawnlawson/SublimeJensaarai`

LibROSA, viewed 25 September 2019, `https://librosa.github.io/librosa/`

Loshchilov, I. and Hutter, F., 2017. Fixing weight decay regularization in adam. arXiv preprint arXiv:1711.05101.

Maaten, L.V.D. and Hinton, G., 2008. Visualizing data using t-SNE. Journal of machine learning research, 9(Nov), pp.2579-2605.

McLean, A., et al, 2019. Tidal, viewed 6 December, 2019, `https://github.com/tidalcycles/Tidal`

McLean, A., and Wiggins, G., 2010. Tidal—pattern language for the live coding of music. Proceedings of the 7th Sound and Music Computing Conference.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A.,

Antiga, L. and Lerer, A., 2017. Automatic differentiation in pytorch.

Stewart, J., 2019. Cibo: Safeguard II, viewed 26 September 2019, https://vimeo.com/288889990/6d92fd1a55

Stewart, J., 2019. Cibo V2 Demo, viewed 26 September 2019, https://vimeo.com/361567860/abac29e10f

Stewart, J., 2019. Sample Corpus Sound Analysis, viewed 26 September 2019, https://github.com/BlindElephants/SampleCorpus_SoundAnalysis

Stewart, J. and Lawson, S., 2019. Cibo: An Autonomous TidalCyles Performer.

Williams, R.J. and Zipser, D., 1989. A learning algorithm for continually running fully recurrent neural networks. Neural computation, 1(2), pp.270-280.