Deep Composer: A Hash-Based Duplicative Neural Network For Generating Multi-Instrument Songs

Jacob Galajda
University of Central Florida
jegalajda13@knights.ucf.edu

Brandon Royal
University of Central Florida
Email: brandonroyal@knights.ucf.edu

Dr. Kien Hua
University of Central Florida
Email: kienhua@cs.ucf.edu

Abstract—**Music is one of the most appreciated forms of art, and generating songs has become a popular subject in the artificial intelligence community. There are various networks that can produce pleasant sounding music, but no model has been able to produce music that duplicates the style of a specific artist or artists. In this paper, we extend a previous single-instrument model: the Deep Composer -a model we believe to be capable of achieving this. Deep Composer originates from the Deep Segment Hash Learning (DSHL) single instrument model and is designed to learn how a specific artist would place individual segments of music together rather than create music similar to a specific genre. To the best of our knowledge, no other network has been designed to achieve this. For these reasons, we introduce a new field of study, Intelligence Duplication (ID). AI research generally focuses on developing techniques to mimic universal intelligence. Intelligence Duplication (ID) research focuses on techniques to artificially duplicate or clone a specific mind such as Mozart. Additionally, we present a new retrieval algorithm, Segment Barrier Retrieval (SBR), to improve retrieval accuracy within the hash-space as opposed to a more traditionally used feature-space. SBR prevents retrieval branches from entering areas of low-density within the hash-space, a phenomena we identify and label as segment sparsity. To test our Deep Composer and the effectiveness of SBR, we evaluate various models with different SBR threshold values and conduct qualitative surveys for each model. The survey results indicate that our Deep Composer model is capable of learning music generation from multiple composers. Our extended Deep Composer model provides a more suitable platform for Intelligence Duplication. Future work can apply this platform to duplicate great composers such as Mozart or allow them to collaborate in the virtual space.**

I. Introduction

Music is an unspoken language that can be universally interpreted. Music is difficult to describe, yet it is easy for humans to distinguish from regular sound. It can boost morale and be applied in a variety of medical applications. Keeping this in mind, it can be challenging to create music that people enjoy, let alone develop networks that can generate enjoyable songs. Most of these networks become composers themselves by analyzing samples and developing a unique style from the music, much like human composers. These networks tend to generate songs whose composition stray from familiar genres of music -as most artists' work are distinguishable from each others. Most networks do not have the capability of learning the style from specific artists; these networks must generate their own. If a network could do this, then it could generate unique songs that still sound familiar to people. Such a network could allow artists to collaborate with each other without ever meeting. It could even *revive* deceased composers and allow them to experiment with modern music.

Many artificial intelligence researchers have sought to make it easier for non-musicians to create music through various methods. Many of these methodologies focus on producing single instrument songs. Rule-based networks [1], [2], [3] use constraints inspired from music theory to generate music. These networks strive at producing structured music, but generated songs lack in creativity. To incorporate more creativity, deep learning networks [4], [5], [6], [7], [8] utilize recurrent neural networks (RNNs) to find the next timestep of a song rather than relying on predefined constraints. While these networks can produce pleasant sounding music, their songs do not match the style of the samples they were trained with. Concatenation-based models [9], [10], [11], [12], [13] break apart songs into small segments and learn how to concatenate these pieces together. Since all of the segments originate in the dataset, concatenation models have the potential to produce songs that sound similar to the dataset itself, however, this method is not common in multi-instrument music generation.

In summary, single instrument models have been able to produce pleasant sounding music, however, most genres of music contain more than one instrument. To achieve this, many researchers [14], [15], [16], [17] have extended previous single-instrument models utilizing generative adversarial networks (GANs) to produce songs with multiple instruments, one instrument at a time. Multiple instrument GAN networks can produce a wide variety of songs, but these networks still generate their own style. To overcome this, we extend a previous, single instrument, DSHL-like, concatenation model: Deep Composer. Our modified Deep Composer model learns the placement patterns of various song segments using hash pair encodings -unlike most models to date. We believe that the Deep Composer is the next step towards duplicating the minds of great composers. The contributions of this work are as follows:

- We demonstrate that the current state of the art methods in music generation through listening is not sophisticated enough to generate multi-instrument music with Intelligence Duplication.
- We propose new solutions to overcome the above mentioned limitation; and we implemented modifications to the Deep Composer to allow for multi-instrument music generation.

- We identify segment sparsity and present a new hash-space retrieval algorithm as a solution.
- Extensive performance results presented in this paper demonstrate the effectiveness of Deep Composer and SBR.

The contributions of this work resonate much further than the scientific and music communities. We take the next leap in artificial intelligence, creating the first hash-based ID network. Our work is the first step towards continuing the legacy of great musicians for years to come.

II. Related Works

While there are no other multi-instrument ID networks to date, the Deep Composer is built upon existing concepts from other music generation research. One crucial component to music generation is song segmentation. The segmentation methodology will define how much information the network will store about the song at every iteration. Some works implemented a rule-based segmentation protocol [1], [8], preserving the structure of each song by producing longer segments. While this approach is ideal for generating music with a similar style to the training data, the size of each segment is too large to genuinely create unique songs. Other works segmented songs into measure-long segments [13], [18], resulting in much smaller segments. While the measure-long approach can ensure the integrity of generated songs, the smaller segments cannot store enough meaningful information. These segmentation methods have also been incorporated in various multi-instrument models [19], [14], [16], [17] as well. The Deep Composer utilizes both methodologies for a two-phase segmentation methodology to capitalize on the advantages of both schemes.

Many researchers have utilized GANs to produce multiple instrument songs because single instrument GANs can be extended to produce multi-instrument songs. This approach has led to many successful multi-instrument models, however, the scope of each model is dependant upon the dataset. Yang et al. [15] indicated that GANs can be used to generate single instrument songs, and Dong et al.[19] extended this model to handle multiple instruments effectively by generating segments of notes for each instrument and layering the results together. Other works [14], [16], [17] implemented similar techniques to implement multi-instrument, GAN-driven networks. These networks are very effective but they lack the ability to generate music outside of the training dataset, because they are trained with globalized data from a specific genre of music.

The GAN approaches are quite flexible in producing generalized multi-instrument songs, but they cannot replicate the *style* of the dataset. In order to learn the style of a composer, the model must be able to learn *why* the composer placed each segment in its position so that the network can replicate this in the future. Previous works from the 20th century [20], [1] came close to achieving this, but were limited to the note-generative model and could only handle non-linear, linguistic-based compositions. Using the segmentation methodology, Deep Composer can learn the placement the composer's original music segments together -regardless of composition style. Additionally, these same segments can be used in testing, as they are too small to be considered for plagiarism [21]. Since Deep Composer uses hash-encoded segments, the style of music in the testing dataset *can* be completely independent from the training set. To our best knowledge, no network has achieved this. For these reasons, we introduce a new field of study in Artificial Intelligence: Intelligence Duplication.

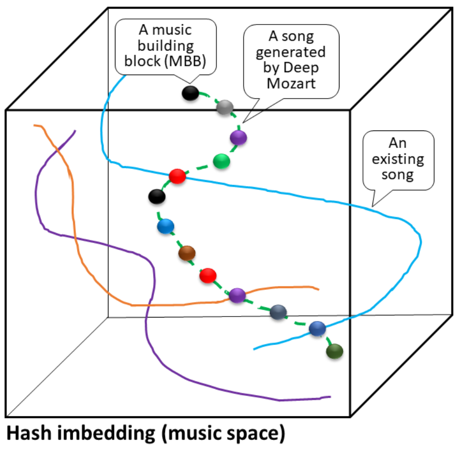

A music building block (MBB)
A song generated by Deep Mozart
An existing song
Hash imbedding (music space)

Fig. 1. Example feature curve within the hash embedding. In future works, a theoretical line can be drawn in the hash space to generate a song.

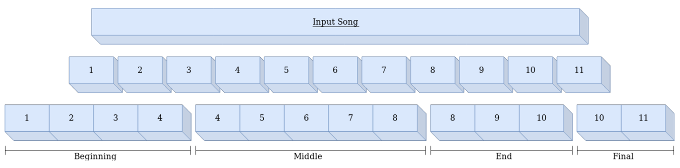


Fig. 2. Example two-phase song segmentation. Segments are categorized into four categories based on the position of the song: Beginning, Middle, End and Final. The number of segments per category depend on the length of the song.

III. Intelligence Duplication

The Deep Composer utilizes a music generation model based on database query processing. The music database may include hundreds of thousands, or more, of music building blocks (MBBs). These MBBs are tiny music segments obtained from existing songs, each a few seconds in duration. Although this is too short to convey any significant melody in terms of human comprehension, the Deep Composer is capable of piecing these MBBs together to construct new songs. This intelligence is acquired through the learning from the songs selected from the music database. Visually, we can view each MBB as a feature point in a music space (i.e., the hash embedding). Deep Composer uses the MBB selected for the current step as a query to retrieve **n** nearest MBBs in the music space, and then selects one of them for the next music segment of the constructed song. This iterative process is repeated until the composition of the song is complete.

Deep Composer is more than a general-purpose platform for music generation. Since it learns the music composition skills from the selected songs, the Deep Composer can also focus its learning on a specific music composer and thereby duplicate this composer in the composition styles. As an example, if the MBBs are derived from only music composed by Mozart, this music space may consist of tens of thousands of tiny Mozart musical building blocks. Moreover, the learned hash embedding capture the Mozart style of music composition. In fact, we can identify any of the Mozart music piece in this music space as a unique curve that pieces together the corresponding MBBs (feature points). In other words, Mozart had created many of such curves in this hash embedding. However, he did not live long enough to explore hundreds of thousands of other possible curves in this Mozart embedding. These are the new Mozart music pieces that the Deep Composer can artificially generates using the Mozart embedding, i.e., Intelligence Duplication. This is illustrated in Fig. 1. The solid curves represent music pieces composed by Mozart; and the dashed curve portrays a new music piece Mozart could have composed, and now generated by the Deep Composer.

Deep Composer also has the potential to offer great composers from different eras the opportunity to collaborate by using MBBs from their music. This is in fact the scenario selected for the performance study in this paper, except with many composers. In this framework, the music space is a hash embedding capable of capturing how these composers would compose their music. Furthermore, since some of these MBBs by the different composers are similar enough, the hash embedding also recognizes how these MBBs by the different composers could be musically combined in a song to facilitate collaboration. Visually, a newly constructed song would be a curve in the embedding, with a subsequent of MBBs belong to one composer transiting to another subsequent of MBBs belong to another composer. Their individual musical styles take turns to play the music in the constructed song. This capability can also be applied to learn and compose a truly worldwide fusion genre of music.

IV. Deep Composer: Hash-Based Music Generation

A. Overview

The Deep Composer's network architecture [21] is inspired by the DSHL network [22]. Our extension of this network allows the network to incorporate multiple instruments much like the previous GAN models [19], [14], [17] while still preserving the structural integrity of each generated song. We incorporate a two-phase song segmentation process that breaks songs into small, unique segments like the Deep Composer network. Our network then isolates each instrument, and concatenates the features from each instrument into one segment. A series of long short-term memory networks (LSTMs) [23], a type of RNN, are tasked with creating a different structural component of a song. Each LSTM learns a forwards and backwards hash encoding from these segments. Deep Composer performs concatenations by retrieving hash pairs within the network's generated embedding or *hash embedding*

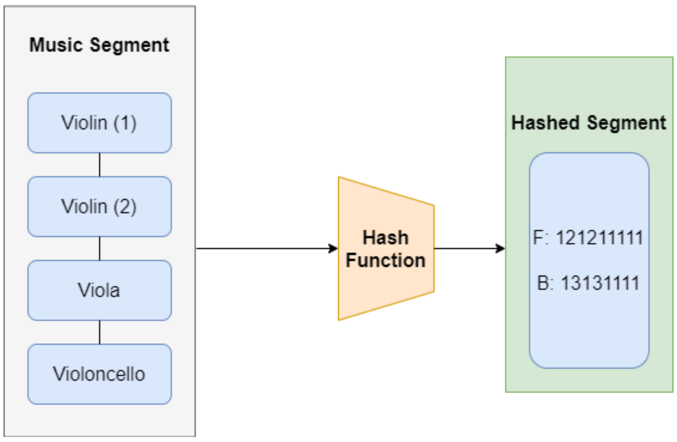


Fig. 3. The feature arrays for each instrument are concatenated together before they are hashed. The result from the hash function is a forward and backward hash-encoded segment.

with a minimal Hamming distance. The following subsections describe the general architecture of Deep Composer and our modifications to this network.

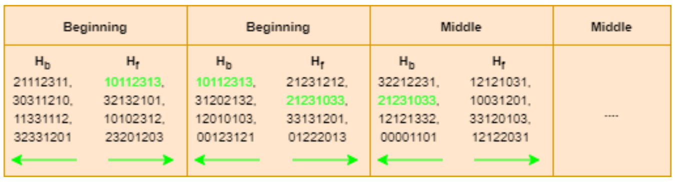
B. Song Segmentation

Songs contain many features that can be considered for segmentation, such as individual notes, chords and pitch. Some researchers [13], [22] divide each song by one or two measures per segment containing just enough information to produce songs without copying the original songs. This is ideal for generating unique songs, but there is not enough data present for the Deep Composer to learn the style of each song. Other networks use structure-based segmentation [24], [25], [26] to break apart songs into much larger segments, preserving the structure of each song. With such large segments, this limits the uniqueness and creativity of the generated songs. One proven workaround for this would be to synthesize both segmentation methodologies. To alleviate the disadvantages from both of these methodologies, the Deep Composer [21] implements a two-phase song segmentation process.

To briefly describe this process, the first phase of segmentation focuses on obtaining segments of equal length. Bretan et al. [13] conclude that the ideal segment length is one or two measures long. The Deep Composer [21] uses the song's time signature to calculate the length of its measures. For example, every measure written in $\frac{4}{4}$ time will have four beats (numerator), each lasting a quarter note (denominator) which will result in 1 second segments. However, not all songs are composed in $\frac{4}{4}$ time. In order to utilize songs with various time signatures, the following formula is used to calculate the segment length for each song:

$$SegmentLength = \sum \frac{TotalBeats}{BeatsperMeasure}$$

The second phase of the segmentation assigns structure to each of these segments, depending on their position and the length of the song. The network considers four positions within a song: beginning, middle, end and final. The length



Beginning		Beginning		Middle		Middle
H_b	H_f	H_b	H_f	H_b	H_f	
21112311,	10112313,	10112313,	21231212,	32212231,	12121031,	
30311210,	32132101,	31202132,	21231033,	21231033,	10031201,	....
11331112,	10102312,	12010103,	33131201,	12121332,	33120103,	
32331201	23201203	00123121	01222013	00001101	12122031	
←	→	←	→	←	→	

Fig. 4. Example hash-encoding of a segmented song. There are four pairs of hash-encodings for each segment which correspond to the four positional categories of the song. Each hash code is 8 positions long and can have 4 values (0, 1, 2, 3) in each position.

of the first three groups are calculated using a series of formulas dependant upon the length of the song and number of segments, whereas the final group's length is arbitrarily defined. Additional overlap variables are introduced to ease the transition from each group. The results of the two-phase segmentation are depicted in Fig. 2.

This two-phase methodology was initially designed for single-instrument music generation. We use this same two-phase segmentation as the Deep Composer, but have extended it to collect individual information about each instrument within a given segment, concatenate their feature arrays into one segment and hash-encode the entire segment. Fig. 3 illustrates this concept.

C. Segment Hash-Pair Encoding

For many information retrieval tasks, hash-based approaches [27], [28], [29], [30], [11] have proven to be quite efficient. The Deep Composer consists of forwards and backwards LSTM pairs working together to learn the hash encodings of each segment. Forward LSTMs take a pair of segments in the forward direction, and the backward LSTMs take the same pair of segments but in the opposite direction. The inner product loss is calculated from both the forward and backward LSTMs, which updates the network's weights.

Instead of directly training with feature data, the Deep Composer learns hash-pair encodings. Previous works [11], [22] have shown that segment hashing can be used to optimize song generation. This method allows for quick segment retrieval, however, the generated song's structure cannot be guaranteed. The Deep Composer network solves this issue by incorporating previous structure components from songs into the hash-encoding generation process [21]. This scheme seeks to minimize the Hamming distance between the forwards and backwards hash codes of neighboring segments or composable segments. The network defines composable segments as adjacent segments within the same structure group of a song in which the Hamming distance between the forwards hash code of the first segment and the backwards hash code of the second segment is minimized. To extend this concept into our own implementation, the hash-code pairs are simply increased to account for the additional instruments. These hash-code pairs are then used like a feature space to train the network which is illustrated in Fig. 4.

In conclusion, our modifications to this network allow the Deep Composer to generate multi-instrument songs. Given that our modifications do not stray far from the original design, we trained our extended model using a classical string-quartet dataset (described in detail in the Experimentation Section) using the recommended parameters as described in [21].

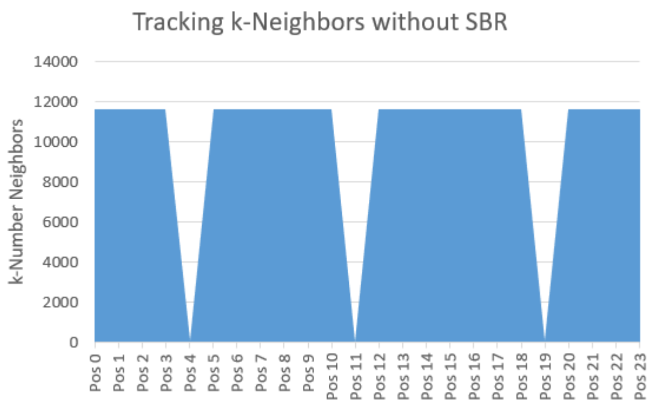


Fig. 5. Segment retrieval with no resolution policies. Segment sparsity in the hash-space is demonstrated by the dips in the graph. Notice how some positions have close to 0 valid neighbors.

V. Hashed Segment Retrieval

A. Overview

When generating music, the network uses this feature space filled with hash pairs or *hash space* to make segment retrievals. Deep Composer's retrieval algorithm works to reduce the Hamming distance between the hash-code pairs of each segment and selects the best next segment whose backwards hash code is very similar to the previous segment's forward hash code (illustrated in Fig. 4). Initially, we applied this retrieval algorithm to the Deep Composer network, but noticed an interesting phenomena: the network would select uncharacteristic segments. A similar phenomena occurs often in other segment concatenation-based networks [9], [10], [11] where two very different segments of music are put together to form an abrupt change in the song. Unlike previous works, our segments are pulled from a hash embedding whose space is vast. Since the Deep Composer works to minimize Hamming distance only, the *direction* of the retrieval is unguided and can lead the network into undesirable regions within the hash-space, such as low-density areas. We believe this phenomena, or segment sparsity, occurs when our network does not have enough close to produce a smooth transition.

To validate this theory, we conducted a test to determine the number of neighbors from each segment within a Hamming distance threshold of **k = 5** from the origin with no resolution policy. The results of our test (Fig. 5) confirm that the network contains some instances of segment sparsity, as characterized by the dips in the graph. These results show that segment sparsity can take place whenever the network retrieves segments from low-density regions within the hash embedding. When this occurs, the Hamming distance between the segment and its nearest neighbors can increase greatly which results in segments with low composability to be

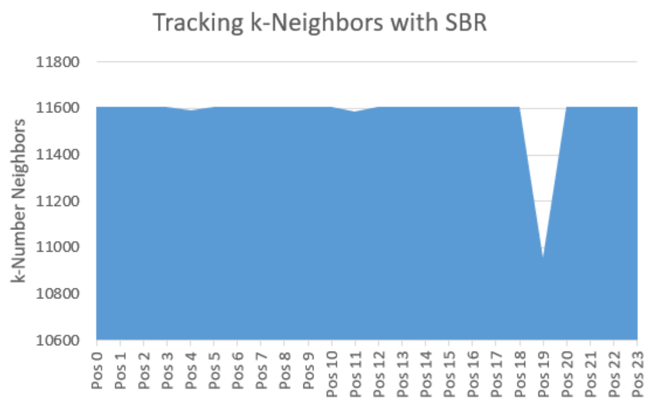


Fig. 6. Segment retrieval with resolution policies. Compared to Fig. 5, the dips are much less dramatic with a resolution policy, still maintaining over 10800 valid neighbors at every position.

selected out of necessity. To prevent this from happening, we propose Segment Barrier Retrieval (SBR): an algorithm designed to detect when segment sparsity occurs by comparing the Hamming distances of the next segment with a segment in a high-density region.

B. *Segment Barrier Retrieval*

SBR is a simple and efficient algorithm that prevents retrievals from branching too far away from areas of high density in the hash-space. The algorithm's ability to detect segment sparsity is dependant upon the set Hamming distance threshold denoted by **k**. We define an origin segment at the beginning of each retrieval branch. The Hamming distance between the origin and any next segment are used to detect segment sparsity. If segment sparsity is not detected, the next segment is added to the current retrieval branch. If segment sparsity is detected, we resolve this by choosing a new next segment close to the previous origin. To encourage creativity, this newly selected next segment is also designated as the new origin. We believe this will also reduce the volatility of generated songs, as these retrieval branches must gradually shift away from the initial origin. To summarize, the retrieval branch is closed off and another branch is created near the previous origin when segment sparsity is detected.

To account for the structure-component of the hash encodings from the Deep Composer, a new branch is started at the beginning of every structural group. When a new structure group is introduced, the current segment becomes the new origin and a new branch is started from this position -unlike retrieval branches completely contained in one structure group. Since the Deep Composer's hash encodings are designed with overlap [21], the origin has an increased probability of falling in a high-density region. If this is not the case, the newly calculated origin will be far away from the previous low-density region. However, this becomes an extreme edge case as more testing data is provided. Overall, this design increases the creativity of the network by allowing the origin to naturally

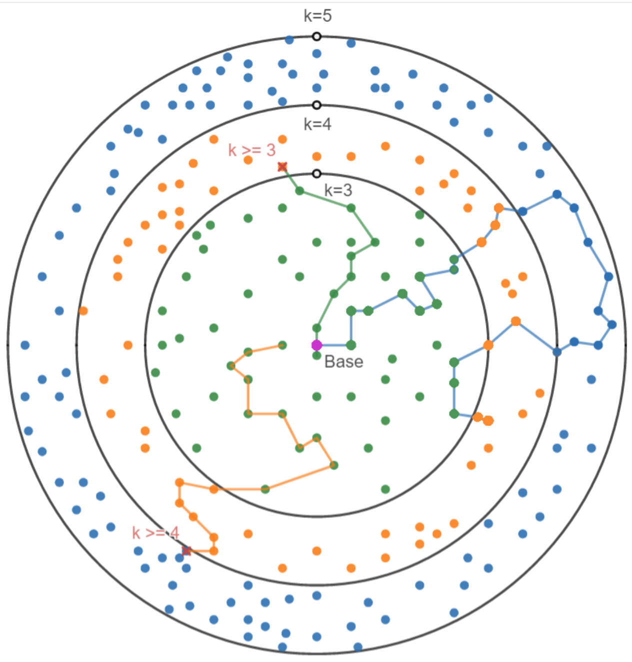


Fig. 7. Example retrieval branches using SBR with various k-Distance thresholds. Retrieval branches are terminated as soon as the Hamming distance from the origin and retrieval segment is greater than the threshold.

shift across the hash-space across structure group transitions rather than within a structure group.

We tested our SBR algorithm with a **k** threshold of 3 and implemented the resolution policy previously described. We tracked the total amount of valid neighbors within the threshold after resolving segment sparsity occurrences. Compared to Fig. 5, there were significantly more valid neighbors (shown in Fig. 6). These results conclude that a resolution policy such as SBR can be useful for alleviating segment sparsity in the hash-space. The dips from both of the previous figures occur in the same positions, which we believe are the song's shifts between the song structure groups (beginning to middle, middle to end, end to final). These results show that segment barrier retrieval is capable of handling structure group transitions effectively.

In summary, SBR is much like a k-Nearest Neighbor algorithm (kNN) that checks and resolves cases of segment sparsity. As mentioned previously, the SBR algorithm is dependant on the Hamming distance threshold **k**. When this value becomes more precise, more cases of segment sparsity will be resolved. Thresholds too high could cause more cases of segment sparsity, whereas lower thresholds can potentially limit creativity or inadvertently cause abrupt changes in the song. In Fig. 7, we illustrate various retrieval branches examples with different threshold values. We believe this optimization will allow the Deep Composer to take another step towards demonstrating Intelligence Duplication.

VI. Experimentation and Survey Results

A. Dataset

In order to demonstrate that the Deep Composer is capable of these things, we must first prove that it can produce pleasant, collaborative music. A comparison between other current multi-instrument models will not suffice as there is yet to be another multi-instrument model that can learn the fused style of multiple composers. Thus, we compare the extended Deep Composer to various versions of itself as in [31]. We utilize a string quartet dataset to train the Deep Composer. The dataset contains 176 classically composed string quartet songs (as MIDI files) for a total of 3.62 MB worth of data. The quartet consists of two violins (one lead, one accompaniment), one viola and one violincello or cello. The time signatures vary across each song, with $\frac{4}{4}$ (common time) the most occurring, $\frac{3}{4}$ (Waltz time) the second most and $\frac{6}{8}$ following. The first two time signatures are the most frequently occurring simple time signatures, whereas $\frac{6}{8}$ time is the most frequently used compound time signature. We specifically chose songs with conflicting time signatures to create various high density regions within the hash-space to increase the chance of incurring segment sparsity. When analyzing segments from varying time signatures, there are still enough composable segments to produce unique songs. We believe these conditions will be ideal for testing SBR.

B. Survey Overview

The previous dataset was used to train the Deep Composer model. To test the ability of the Deep Composer model as well as the SBR implementation, we developed four individual groups with various threshold values for the SBR algorithm:

- Group 1: Deep Composer without SBR
- Group 2: Deep Composer with SBR (k-Distance of 3)
- Group 3: Deep Composer with SBR (k-Distance of 4)
- Group 4: Deep Composer with SBR (k-Distance of 5)

To reduce the complexity of each survey, each model generated five songs using the same testing dataset as the training dataset. The best song from each group was used in both the individual and comparison surveys to reduce the overall complexity -determined by the song with the highest average number of neighbors at each iteration. We used the following formula to be used in both the individual and comparison surveys to generate the average of all valid neighbors, $\mathbf{N} = \sum \frac{n_i}{s}$ where $\mathbf{n_i}$ within a song whose Hamming distance from the origin is less than $\mathbf{k}$. In this experiment, each generated 5 songs per group (24 segments in length) and used the best one from each group.

We randomly selected 20 individuals from a nearby accredited university to participate in both surveys. Participants were selected on the basis that they enjoyed music genres similar to the classical dataset. If participants were not familiar with the genre/style of music, asking them to judge the songs would yield potentially biased results. Additionally, participants were required to have a minimum understanding of music theory whether they learned this through playing an instrument or from reading music theory articles. We imposed these limitations so that our participants would potentially have a more inclined opinion from each generated song.

Our first survey was designed to assess the Deep Composer's ability to produce pleasant sounding, collaborative music. Participants received an electronic copy of each song titled by their group number only, along with a randomly generated order in which the songs should be listen to. At the end of each song, participants were asked to evaluate the song's pleasantness. For simplicity, we define a pleasant song as one that contains little to no abnormalities, such as unpleasant changes in key and shifts in tempo. Participants then evaluated the pleasantness of each group's song on a scale of 1 (very unpleasant) to 10 (very pleasant).

Our second survey asked participants which song of the four groups was the most pleasant. Since song pleasantness is subjectively measured, this survey better refines our findings from the initial survey. This additional survey also provides us more information about the **k** threshold for the segment barrier retrieval algorithm. The songs generated by each group can be found in supplementary resources.

C. Survey Results

1) Individual Song Performance Survey: The results of the first survey exhibit interesting trends. Collectively, the pleasantness scores for all models averaged 5.85 with a median pleasantness score of 6. This negative skew indicates a noticeable performance difference across each model. Despite this, the Deep Composer model without SBR (Group 1) was the second best performing group of the experiment scoring an average of 6.6 and a median score of 7.0 in song pleasantness. This shows that this network, at the very least, is capable of producing somewhat pleasant, collaborative multi-instrument music with or without SBR. However, this may not be the case for Group 1 as the amount of testing data decreases.

The survey results also suggest that the value of **k** is inversely related to the overall pleasantness of the song. Group 2 (**k = 3**) performed better than the other three groups, with an average score 11.36% higher than the next best model, Group 1. Additionally, Group 2's median scores were 7.14% greater than Group 1's. We believe that larger values of **k** lead to larger distances needed to travel back to the origin (This can be seen in Fig. 7). These larger leaps can lead to unpleasant shifts, however, shorter leaps are less detectable. In Group 1's case, we speculate that the singular retrieval branch was relatively close to the origin at most positions. This theory would also explain why Group 1 had received such high pleasantness scores.

We also hypothesize that average valid neighbors **N** is another contributing factor in song pleasantness. To demonstrate this, we collected the values of **N** both before and after applying SBR in every group and found that different values of **k** also result in different **N** increases when using SBR. Group 2 experienced the most significant increase of 5.59%, whereas Groups 3 and 4 increased by 0.40% and 2.61%, respectively. With these values we conclude that along with larger **k** values,

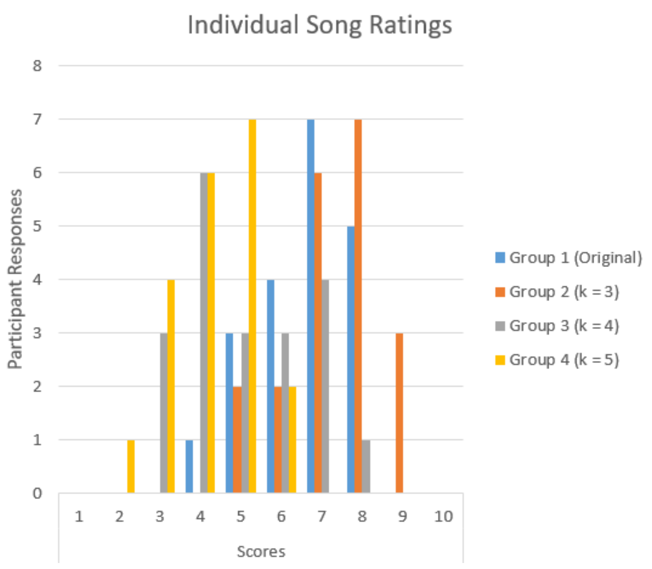


Fig. 8. The raw scores from the first survey indicate a favoring towards Group 2.

Results	Avg. Score	Med. Score	N Before	N After
No SBR	6.6	7.0	11,579	—
k = 3	7.4	7.5	10,615	11,208
k = 4	5.1	5.0	10,973	11,017
k = 5	4.3	4.0	10,948	11,234

TABLE I
INDIVIDUAL SONG PERFORMANCE RESULTS SHOW A CONSIDERABLE N AVERAGE INCREASE FOR LOWER VALUES OF **K**.

larger values of **N** can potentially reduce detected incidents of segment sparsity. It appears that lower **k** values and more frequently corrected segment sparsity events result in a greater value of **N**. The raw data from this experiment is shown in Table I.

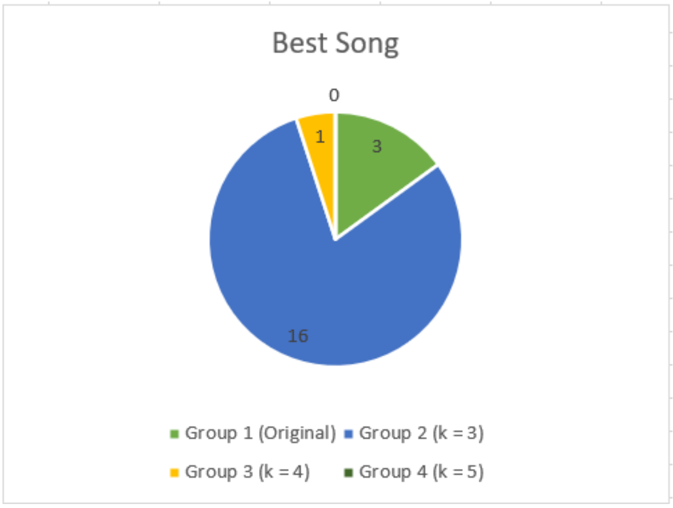


Fig. 9. The results of the comparison survey show that Group 2 was the most pleasant song.

2) *Song Comparison Survey:* Utilizing the same groups and participants of the first survey, the second survey was designed to see which song performed the best. To alleviate potential biases from the previous survey, participants received renamed electronic copies of each song along with a new listening order. Participants were instructed to listen to each song and asked to subjectively select the song that performed the best. We also included the labeled sheet music of each song. The results of this survey can be found in Fig. 9.

The comparison survey supports the individual surveys in that lower **k** values are ideal for producing pleasant songs. Out of the four groups, Group 2 generated the best song out of the experiment, depicted in Fig. 9. Groups with larger **k** values did not perform nearly as well, resulting in a non-proportionate distribution in participant responses -indicating a clear trend towards Group 2.

D. *Survey Summary*

The results from the second survey show that Group 2 performed significantly better than the other groups. The scores from the individual scores would also agree with this verdict, concurring that the best Deep Composer model from this experiment uses a **k** distance of 3. An explanation as to why this model performed so well can be derived from the SBR algorithm. Lower values of **k** ensure that segments retrieved are from within the same cluster in the hash-space, however, a direct correlation between decreased **k** values and increased **N** cannot be determined with the current data.

Interestingly, Group 1 performed much better than we initially anticipated in both surveys. Group 1's performance can be attributed to an ideal start location in the hash-space and inadvertently selecting good next segments thourghout the entire retrieval branch. The large value of **N** shown in Table I would suggest that this model was in a very dense region for the majority of its retrieval. Another possibility to consider is that this model did not have to make any jumps from the hash-space back to the origin. These large jumps in SBR could have remedied segment sparsity but increased the likelihood of the song having abrupt and unpleasant jumps as the Hamming distance is minimized in this situation. One modification to the SBR algorithm would be to move the origin throughout a branch using $\mathbf{origin} = \mathbf{segments}[\mathbf{i} - \mathbf{L}]$ where $\mathbf{L}$ is the number of segments separating the origin and the current segment. In this case, there would be a much lesser jump.

In both the individual and comparison surveys, groups with larger **k** values (Groups 3 and 4) did not perform as well as Group 2 with the smallest **k** value in the experiment. The **N** increases from Groups 3 and 4 are much smaller than Group 2, indicating Groups 3 and 4 had less frequent jumps in the hash-space. These jumps, however, are greater in length, as the threshold limit is greater. The performance results of Groups 3 and 4 would support the previous claim that the current SBR algorithm is not optimized for larger threshold values.

VII. Conclusion and Future Works

In conclusion, we have extended the previous Deep Composer model to incorporate multi-instrument song generation. Unlike most networks, Deep Composer generates a unique, *custom* embedding filled with hash-pair encodings instead of features or single hash values. This hash embedding generated by the network extends Deep Composer's capacity, giving it the ability to generate a potentially infinite number of songs without additional training. Deep Composer places segments based on their composability in which the network learns how to duplicate the actual composer(s) rather than replicate the globalized genre of music. To our knowledge, this implementation is unlike any other network that has been studied.

For these reasons, we introduce a new field of study, Intelligence Duplication, to explore this subject further. AI research generally focuses on developing techniques to mimic universal intelligence. Intelligence Duplication (ID) research focus on techniques to artificially duplicate/clone a specific mind such as Mozart to perform tasks only Mozart could. AI research generally focus on mimicking universal intelligence such as face recognition or human action recognition. In this research, we attempt Intelligence Duplication, in which the learning is not to capture general knowledge to solve general problems, but to duplicate/clone a specific mind such as "recreating" Mozart. More specifically, we use music as a model to illustrate ID. It takes two steps to achieve this. Step 1 is demonstrated in this paper showing an artificial way to learn music through listening much like how toddlers are learning a native language. Our future work in Step 2 will apply this artificial learning technique to demonstrate ID using Mozart as an example. It is ready for our next research goal to illustrate Intelligence Duplication which will be the key to reviving great composers, such as Mozart, from the dead.

References

[1] D. Cope and M. J. Mayer, *Experiments in musical intelligence*. AR editions Madison, 1996, vol. 12.

[2] R. P. Whorley and D. Conklin, "Music generation from statistical models of harmony," *Journal of New Music Research*, vol. 45, no. 2, pp. 160–183, 2016.

[3] D. Wells and H. ElAarag, "A novel approach for automated music composition using memetic algorithms," in *Proceedings of the 49th Annual Southeast Regional Conference*. ACM, 2011, pp. 155–159.

[4] K. Chen, W. Zhang, S. Dubnov, G. Xia, and W. Li, "The effect of explicit structure encoding of deep neural networks for symbolic music generation," in *2019 International Workshop on Multilayer Music Representation and Processing (MMRP)*. IEEE, 2019, pp. 77–84.

[5] G. Medeot, S. Cherla, K. Kosta, M. McVicar, S. Abdalla, M. Selvi, E. Rex, and K. Webster, "Structure net: Inducing structure in generated melodies," in *The 19th International Society for Music Information Retrieval Conference*, 2018.

[6] H. Chu, R. Urtasun, and S. Fidler, "Song from pi: A musically plausible network for pop music generation," *arXiv preprint arXiv:1611.03477*, 2016.

[7] D. D. Johnson, "Generating polyphonic music using tied parallel networks," in *International conference on evolutionary and biologically inspired music and art*. Springer, 2017, pp. 128–143.

[8] M. Chan, J. Potter, and E. Schubert, "Improving algorithmic music composition with machine learning," in *Proceedings of the 9th International Conference on Music Perception and Cognition, ICMPC*, 2006.

[9] J. Nika, M. Chemillier, and G. Assayag, "Improtek: introducing scenarios into human-computer music improvisation," *Computers in Entertainment (CIE)*, vol. 14, no. 2, p. 4, 2016.

[10] F. Pachet, "The continuator: Musical interaction with style," *Journal of New Music Research*, vol. 32, no. 3, pp. 333–341, 2003.

[11] A. Charapko and C.-H. Chuan, "Indexing and retrieving continuations in musical time series data using relational databases," in *2014 IEEE International Symposium on Multimedia*. IEEE, 2014, pp. 341–346.

[12] H.-Y. Lin, Y.-T. Lin, M.-C. Tien, and J.-L. Wu, "Music paste: Concatenating music clips based on chroma and rhythm features." in *ISMIR*. Citeseer, 2009, pp. 213–218.

[13] M. Bretan, G. Weinberg, and L. Heck, "A unit selection methodology for music generation using deep neural networks," *arXiv preprint arXiv:1612.03789*, 2016.

[14] X. Liang, J. Wu, and J. Cao, "Midi-sandwich2: Rnn-based hierarchical multi-modal fusion generation vae networks for multi-track symbolic music generation." 2019.

[15] L.-C. Yang, S.-Y. Chou, and Y.-H. Yang, "Midinet: A convolutional generative adversarial network for symbolic-domain music generation." 2017.

[16] H.-M. Liu and Y.-H. Yang, "Lead sheet generation and arrangement by conditional generative adversarial network." *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), Machine Learning and Applications (ICMLA), 2018 17th IEEE International Conference on, ICMLA*, pp. 722 – 727, 2018.

[17] F. Guan, C. Yu, and S. Yang, "A gan model with self-attention mechanism to generate multi-instruments symbolic music." *2019 International Joint Conference on Neural Networks (IJCNN), Neural Networks (IJCNN), 2019 International Joint Conference on*, pp. 1 – 6, 2019.

[18] D. Eck and J. Schmidhuber, "Finding temporal structure in music: Blues improvisation with lstm recurrent networks," in *Proceedings of the 12th IEEE workshop on neural networks for signal processing*. IEEE, 2002, pp. 747–756.

[19] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang, and Y.-H. Yang, "Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment." 2017.

[20] D. Cope, "Experiments in musical intelligence (emi): Non-linear linguistic-based composition," *Interface*, vol. 18, no. 1-2, pp. 117–139, 1989.

[21] B. Royal, B. Zhang, and K. A. Hua, "Deep composer: Deep neural hashing and retrieval approach to automatic music generation," in *to appear in Proc. of IEEE International Conference on Multimedia and Expo*. ICME, 2020.

[22] K. Joslyn, N. Zhuang, and K. A. Hua, "Deep segment hash learning for music generation," *arXiv preprint arXiv:1805.12176*, 2018.

[23] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[24] R. Sun, J. Zhang, W. Jiang, and Y. Hu, "Segmentation of pop music based on histogram clustering," in *2018 11th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*. IEEE, 2018, pp. 1–5.

[25] C.-i. Wang, G. J. Mysore, and S. Dubnov, "Re-visiting the music segmentation problem with crowdsourcing." in *ISMIR*, 2017, pp. 738–744.

[26] G. Sargent, F. Bimbot, and E. Vincent, "Estimating the structural segmentation of popular music pieces under regularity constraints," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 2, pp. 344–358, 2017.

[27] K. Li, G.-J. Qi, J. Ye, and K. A. Hua, "Linear subspace ranking hashing for cross-modal retrieval," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 9, pp. 1825–1838, 2017.

[28] Q.-Y. Jiang and W.-J. Li, "Deep cross-modal hashing," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 3232–3240.

[29] K. Joslyn, K. Li, and K. A. Hua, "Cross-modal retrieval using deep de-correlated subspace ranking hashing," in *Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval*. ACM, 2018, pp. 55–63.

[30] F. Zhao, Y. Huang, L. Wang, and T. Tan, "Deep semantic ranking based hashing for multi-label image retrieval," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1556–1564.

[31] K. H. Cheah, H. Nisar, V. V. Yap, and C.-Y. Lee, "Convolutional neural networks for classification of music-listening eeg: comparing 1d convolutional kernels with 2d kernels and cerebral laterality of musical influence." *Neural Computing & Applications*, vol. 32, no. 13, pp. 8867 – 8891, 2020.