JAZZMASTER: AN AUTOMATED ACCOMPANIMENT SYSTEM FOR JAZZ MUSIC

A senior design project submitted in partial fulfillment of
the requirements for the degree of Bachelor of Science
at Harvard University

ALEXANDER G. MARIONA

S.B. Degree Candidate in Electrical Engineering

Faculty Advisor: Flavio du Pin Calmon

Harvard University School of Engineering and Applied Sciences

Cambridge, MA— April 2021

ACKNOWLEDGMENTS

I must give my thanks to three groups of people without whom this project would never have been possible.

First and foremost, I would like to thank Flavio du Pin Calmon for his support as my advisor. He has given me meaningful insight, actionable feedback, and helpful encouragment at all points, both with respect to this project and beyond. I owe much of what I am now as a graduating engineer to him.

Secondly, I give my thanks to the ES100 teaching staff, most especially to my section leader Peter Zoogman. Weekly sections with Peter kept this project moving forward even when I was being pulled in many different directions. It would also be a mistake for me to forget my fellow section members, Mark, Eva, Charlie, and Araceli. I truly wish that we could have been on campus together for our senior year. The conversations and commiseration would have been all the more meaningful.

Last, but most certainly not least, I must thank my roommates at Fellsway—Alex, Isaac, and Ben. You dragged me out of my room and made sure that I came out of the other side of all of this with my sanity. Let the boys play.

ABSTRACT

Jazz is a musical genre which focuses heavily on improvisation in a group setting. In order to practice and improve, musicians must frequently play a wide variety of songs with others. Beginners or other musicians who do not have access to a band or practice group need a system which provides a suitable replacement experience. We design and implement a system which can accompany a soloist playing an arbitrary melody in real-time. If the melody is recognized to be that of a known jazz standard, the device will follow along to that standard. If the melody is not recognized, or if the player deviates from a known melody, the device will comp with a suitable, conventional chord progression.

The project consists of three main tasks: identifying a melody, determining a suitable accompaniment, and playing the accompaniment. All of these tasks present primarily signal processing challenges. Identifying the melody involves parsing a raw audio waveform into a programmatic representation. Such a representation can then be used to produce an accompaniment by matching to a database of known melodies and performing musical transformations to generate an appropriate harmony. Finally, the device must actually play along with the musician, following the musician's tempo and phrasing well.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 BACKGROUND AND MOTIVATION

Jazz is a musical genre defined by its improvisational nature and its band- or ensemble-oriented instrumentation. The simplest structure for a jazz tune consists of a melody played over a chord progression. This combination of melody and rhythm is called the "head" and is traditionally played by all of the instruments together at the start of the song. After that, a subset of the instruments take turns improvising, or "soloing", over the head, with the rest of the band providing accompaniment, or "comping", by playing the chords and possibly improvising their own countermelodies to complement the soloist. Finally, the song usually concludes by having all the instruments "go back to the head" to play the main melody together, as in the beginning.

Because of the group dynamic of jazz, every musician must be comfortable both soloing and comping to jazz tunes. There is an ill-defined body of songs called "standards" which most proficient or experienced musicians can play from memory, or nearly from memory. One part of the learning process for new musicians is to become familiar with playing these standards in a band. Not all beginner or even intermediate musicians will be able to have consistent or quality practice time with a group, but any serious student will spend a significant amount of time practicing individually. These students need a musical system that can simulate the experience of playing with a band. Seasoned musicians could also benefit from the convenience of such a system, even if the overall experience is inferior to playing with a live group.

1.2 PROBLEM STATEMENT

Jazz musicians must be skilled improvisors capable of soloing over a large library of standards. Beginners or other musicians who do not have access to a band or practice group need a system which provides a suitable replacement experience, but most existing methods for providing accompaniment either do not respond to what the musician is playing or are not designed to handle jazz in particular. Musicians need a system which can produce an appropriate harmony in response to a jazz melody in real-time.

1.3 IMPACT ON THE END USER

Any jazz musician can benefit from the opportunity to practice and improvise on-demand in an band setting, but beginners who may not be part of a jazz community or musicians who live far away from a thriving jazz scene can have difficultly ever finding a band to play with. Furthermore, these musicians may be discouraged from seeking out or attempting to create a band due to their own lack of experience. As a result, these players can often stagnate, give up on the genre, and ultimately may never experience the excitement and energy of playing in a jazz band.

For musicians without access to a band, a personal practice system that can emulate the basic role of an accompanying band member can help bridge that gap for a novice or intermediate musician, as any sort of "live" playing is dramatically better than no live playing at all. Playing over a backing track or a recording is decidedly not the same. There is no dynamism or uniqueness in a recording, whereas a band will never play the same song twice in exactly the same way. That freedom and uncertainty is challenging, and having a system that can emulate that changing play style and do so in a way that is responsive to the player is a decided improvement over simpler practice methods.

1.4 PROJECT OVERVIEW

The project consists of three main tasks:

1. Identify a melody.
2. Determine a suitable accompanying harmony.
3. Play the accompaniment.

All three must be completed in real-time for a wide range of input melodies which can be characterized as being drawn from the input space of all possible jazz melodies. These tasks present primarily signal processing challenges. Identifying the melody involves parsing a raw audio waveform into a programmatic representation. Such a representation can then be used to produce an accompaniment either by drawing on pre-existing knowledge of jazz melodies and how they are harmonized, or by starting from musical first principles to harmonize the melody from scratch. Finally, the system must actually comp with the musician, which means that when the system plays its harmony, it must follow the musician's tempo and phrasing reasonably well. Thus, these qualities of the melody must also be extracted from the audio input signal.

Placing this project within the context of problems from electrical engineering, multiple strategies from signal processing and data analysis are relevant. The general approach of transforming time domain

signals into the frequency domain (or a similar transform domain) is widely applicable, especially for audio signals. When working with a large corpus of information based on audio, such as a database containing knowledge of jazz melodies, these structures will often store raw amplitude waveforms [19]. In these systems, it is extremely difficult to perform computationally-viable searches or even to load the entire databases into memory at a single time. Finding a meaningful and useful representation of a high-dimensional signal requires a precise definition of the problem to be solved as well as an analysis of how the information can be extracted and represented in a sparse basis.

2

BACKGROUND RESEARCH

The problem of automating a musical accompaniment is not new. There have been a number of attempts to develop systems to address a problem very similar to the one we have presented, including ones that approach jazz improvisations in particular. We offer a brief summary here of existing systems categorized by the way they produce the output accompaniment given an input melody and by the musical style the system is primarily designed for. There are three main categories of systems present in the literature:

- those using machine learning models [11], [15], [16], [3],
- those using classical statistical correlation methods [12],
- and those using pure musical analysis [6].

The defining feature of machine learning based systems is the use of large neural networks, hidden Markov models, or other architectures which are trained on a great number of existing recordings (which may be pre-processed into a symbolic representation as opposed to a simple audio waveform) in order to learn what chord progression might be used to accompany a given melody. The main benefit of such systems, as with any machine learning application, is their capacity to capture complex relationships and structures which would be too numerous to encode directly or simply unknown to the designers a priori. The main obstacle in producing such systems is finding or creating a sufficiently large, well-labeled, and meaningfully representative data set which captures the style of music for which the system is designed. Of the three classes of systems, those based on machine learning are the most numerous in the literature and seem to be the most promising for future endeavours.

Systems which rely on statistical analysis approach the problem of generating an accompaniment by tackling the related problem of determining what chord is most likely to be played at any arbitrary point in the melody. In theory, this is not too dissimilar from the general goal of the machine learning approach, but these systems operate without the infrastructure and baggage of machine learning, using simpler probability distributions and inference equations. Compared to machine learning systems, significantly smaller data sets are required to create a reasonable model, but their bounded complexity can result in limitations on the system's ability to effectively adapt to the entire input space.

Systems which solely make use of musical analysis do not perform any inference, but instead approach the problem from the perspective

of music theory. These systems attempt to produce an accompaniment that makes musical sense, given a list of rules that are pre-defined and are generally not subject to change based on the input. Of the three categories, these systems are the most inflexible and the most susceptible to unexpected irregularities in the input. When considering the complexity of the harmonies produced, systems which rely on musical analysis are limited by the extent of the rules they are working with, whereas systems which make use of machine learning, for example, are limited only by the extent of the data they have been trained on. The benefit to using a system predicated on musical analysis is a stronger guarantee on correctness, since it will only produce harmonies that it knows to be musically viable, even if those harmonies are dramatically simpler than those that a human might produce.

In practice, more complex systems which rely on machine learning or statisical processing often do incorporate some kind of musical analysis, either as a check on the final output of the model or as a pre-processing stage in forming the training data, but they do not rely on it as their primary method because of its limited scope.

The main gap in the landscape of existing automated accompaniment systems is that there are few which can handle improvised jazz melodies well. Because of the inherent variety of music, no system can be universal. Machine learning systems are limited by the data they are trained on and musical analysis systems are limited by their rules of analysis. Our system seeks to fill this gap.

3

DESIGN REQUIREMENTS

3.1 QUALITATIVE GOALS

To begin formalizing the system requirements, we return to the three main tasks identified in Section 1.4:

1. Identify a melody.
2. Determine a suitable accompanying harmony.
3. Play the accompaniment.

The first general consideration is the kind of inputs the system should be able to handle. Since we are concerned with jazz melodies, it is natural to consider traditional jazz instruments as potential input sources. Because of its near-universal presence in jazz and many other musical genres, we will take the piano as the main instrument with which to concern ourselves. This choice is signficant because the different acoustic properties of instruments can affect the design of signal processing techinques, algorithms, or filters which may be used in identifying the melody.

Furthermore, we will make the significant simplifying assumption that the melody will be monophonic, that is, only one note will be played at a time. This dramatically reduces the complexity of melody identification [2] without dramatically reducing the applicability and practicality of our system. Even though the piano is not an inherently monophonic instrument, many others which are commonly found in jazz are, such as those in the brass and woodwind families. For this reason, a large subset of jazz melodies and improvisation found in real music is also monophonic.

A second crucial factor to consider is how fast the system must be able to operate. Because the system must be able to determine and produce a harmony in real-time, we are strictly limited in computation time by the tempo of the music we are accompanying. Qualitatively speaking, we wish to be able to keep up with the majority of the standard jazz repertoire, accepting that there will likely be outlier songs that are played abnormally quickly and which we are willing to ignore.

A third and final qualitative goal is that our system should produce musically "correct" harmonies. This is both an obvious necessity and a slippery formulation. Without some guarantee or expectation of correctness, the system is effectively worthless. Given the massive input space of jazz melodies, however, it is also unrealistic to expect

perfect performance on every possible input. We will be satisfied with generating a correct harmony most of the time and with playing that harmony correctly (i.e. rhythmically aligned with the musician) most of the time. Harmonic correctness can be determined using musical analysis (even if musical analysis was not used to generate the harmony), and rhythmic correctness can be determined mathematically, by comparing the melody and harmony after both have been played.

3.2 TECHNICAL SPECIFICATIONS

The quantitative technical specifications and design requirements for the system are derived from the qualitative goals outlined above. We divide these specifications into three categories:

1. Those which quantify the range and kinds of inputs on which the system must function.
2. Those which limit the resources to which the system has access.
3. Those which characterize system correctness and lower bound the acceptable system correctness.

For a summary of all technical specifications, see Table 3.1.

3.2.1 *Input Specifications*

The range of inputs the system must handle is directly determined by the music which we expect to be played for the system. Per the problem statement given in Section 1.2, the user will produce a melody for which a harmony must be generated. Given that we are only considering monophonic melodies, the main quantities which we need to extract from the audio signal in order to fully identify the melody are the pitch and duration of each note. The playing range of a standard piano is from 27.50 Hz to 4186.01 Hz (A0 to C8), but we can limit this to the middle and high frequencies we expect to hear in melodies, about 165 to 1320 Hz (E3 to E6). Like the monophonic melody assumption, this limited range will simplify the signal processing techniques needed to identify a melody without dramatically decreasing the sample space of melodies which we are able to identify.

Similarly, the range of tempos which the system should handle can be limited to between 40 and 240 beats per minute (bpm), and the shortest rhythmic value can be limited to an eighth note. At 240 bpm, an eighth note has a duration of 0.125 seconds, which is the minimum temporal duration over which the system must be able to detect notes.

Most jazz melody instruments do not exceed a sound level of 90 dB during normal playing (with 0 dB being the weakest sound heard on a logarithmic scale). This sound level is also measurable on most commerically available microphones without distortion. Likewise,

most instruments will not be play softer than 50 dB. Accordingly, the system must be able to detect melodies which are played at a volume in range of 50-90 dB.

Because the analog audio signal is going to be processed digitally, we must also define the minimum sampling rate of the system in order to meet the preceding input specifications. The Nyquist-Shannon sampling theorem states that the minimum sampling frequency f_s to completely specify a signal is given by

$$f_s > 2f_m,$$

where f_m is the maximum frequency of the signal. Directly applying this relationship with $f_m = 1320$ Hz, we have that the sampling frequency must be at least 2640 Hz. Furthermore, the digital processing the the system employs must have a sufficiently fine frequency resolution to distinguish the smallest musical interval, the semitone, which is 100 cents. Cents are also measured on a logarithmic scale. Given two notes at frequencies a and b in Hz, the number of cents measuring the interval from a to b is given by

$$n = 1200 \log_2 \left(\frac{b}{a} \right).$$

The two lowest notes which the system much distinguish are E3 (164.81) and F3 (174.61 Hz) (we can verify that the interval between these two notes is 100 cents). Therefore, the system must have a frequency resolution of at least 10 Hz.

3.2.2 *Processing Specifications*

The most important quantitative limitation on processing and computation is the time which can be devoted to actually generating the harmony, given that the system must generate a harmony in real-time for a melody which is not known a priori. The particular way in which a harmony is generated will depend on the system design, but all designs will only have access to the information contained in the notes of the melody which have already been played.

If the system makes maximally immediate use of the information it receives, then it will "re-generate" a harmony after every note. More generally, we can consider subdividing the melody into submelodies consisting of a constant number of notes n_s, with $n_s > 0$ chosen such that the system must produce a harmony for the next submelody after hearing a single submelody. The shortest possible duration of a submelody is $d_s = n_s \times 0.125$ s. Existing systems for melody identification (whose parameters may be defined differently) effectively choose n_s in the single-digit range [18]. Thus, for the purposes of setting a quantitative specification, we will require that the system be able to generate a harmony in at most 1 second, keeping in mind that this

value is somewhat dependent on design choices and implementation details.

3.2.3 *Correctness Specifications*

It is immediately obvious from listening to multiple performances of the same jazz song that there are many different ways to harmonize a particular melody, all of which should be considered correct. Although it may be difficult to choose a finite set of rules which could generate even one of those possible correct harmonizations, it is much easier to choose a finite set of rules which, when broken, would imply that the harmonization is incorrect. Assuming that this list of rules is sufficiently complete, such that a qualitatively negligible fraction of real-world jazz harmonies would break them, we can use these rules to quantify the correctness of the system's harmonies.

In order to formulate a single, quantitative metric for evaluating the harmonic correctness of a given accompaniment, we must first define some terms for describing the musical qualities of the accompaniment. Consider all the notes which are played over a given chord. We refer to these notes as the chord's *relevant notes*. Furthermore, every chord can be associated with an appropriate major or minor scale based on its quality and root note. We refer to this scale as the chord's *relevant scale*. We call a note *harmonic* with respect to a given chord if it is either a chord tone or a member note of the chord's relevant scale. If a note is not harmonic, we call it *anharmonic*. Similarly, we define a *harmonic chord* to be a chord whose relevant notes are all either (a) harmonic notes or (b) separated by at most 2 notes from a relevant harmonic note. Correspondingly, an *anharmonic chord* is a chord which is not harmonic. More specifically, an anharmonic chord has at least one relevant anharmonic note which is separated by 3 or more notes from a relevant harmonic note (or there may be no relevant harmonic notes at all). Finally, we define the harmonic correctness of an accompaniment to be the ratio of the number of harmonic chords to the total number of chords in the accompaniment. Thus, the harmonic correctness of an accompaniment lies in the range $[0, 1]$.

Although the melody over which the system's harmony is to be played is not known as the system is generating the harmony, we can retroactively compute the harmonic correctness of a melody-harmony pair. In characterizing the system's overall harmonic correctness, we can choose a threshold for the fraction of chords per song which are allowed to be anharmonic. The choice of threshold is inherently somewhat arbitrary. Within a given range, relatively small changes to the threshold are unlikely to affect the choices made in the overall design of the system, although they could affect its behavior. A more demanding threshold might result in a system which generates less interesting, "safer" harmonies, perhaps ones which are almost trivially

CATEGORY	TECHNICAL SPECIFICATION	VALUE
Input	Pitch Input Range	165 - 1320 Hz
	Tempo Input Range	40 - 240 bpm
	Minimum Note Duration	0.125 s
	Volume Input Range	50 - 90 dB
	Minimum Sampling Frequency	2.64 kHz
	Minimum Frequency Resolution	10 Hz
Processing	Maximum Generation Time	~1 s
Correctness	Minimum Harmonic Correctness	90%
	Minimum Rhythmic Correctness	90%

Table 3.1: Summary of technical specification values.

correct. This is decidely less exciting, so we choose to require that 90% of chords generated per song by the system must be harmonic. We believe this threshold to be high enough for the system to be meaningfully useful without being overly ambitious.

In addition to being able to generate a correct harmony, the system must also be able to play its harmony in time with melody. As with harmonic correctness, this property of rhythmic correctness can only be evaluated after the melody and its corresponding harmony have been played. We define a *rhythmic* chord to be a chord which is played within a tolerance window (80 ms) of the true beat location. We define an *arhythmic* chord to be a chord which is not rhythmic. We then define the rhythmic correctness of an accompaniment to be the ratio of the total number of rhythmic chords to the total number of chords in the accompaniment. Like the harmonic correctness, the rhythmic correctness of an accompaniment lies in the range $[0, 1]$.

As with harmonic correctness, we can evaluate the rhythmic correctness of the system by choosing a threshold for the fraction of chords per song which are allowed to be arhythmic. Following a similar logic as before, we require that 90% of chords generated per song by the system must be rhythmic.

4

DESIGN DESCRIPTION

We present here a complete specification of the system design. To offer some motivation for a few of the design decisions, we first return to the system classifications presented in Section 2. Although systems using machine learning are increasingly prevalent and successful, our lack of access to a sufficiently large and well-labeled data set of harmonized jazz melodies makes this approach infeasible. Nonetheless, the idea behind using machine learning, namely that information contained in known samples can be used to predict and compute over unseen samples, is still useful and can be exploited without employing the full machinery of a neural network.

4.1 DESIGN MOTIVATION

Our proposed method for collecting and leveraging this knowledge is the following: instead of creating a model which synthesizes all of the training data, we will draw on the information contained by individual samples directly. The musical "alphabet" is relatively small (12 notes). Musical "words" or submelodies which express a coherent musical "thought" are very roughly about 5 to 10 notes long. If we consider sufficiently small sequences, then all possible sequences are likely to appear at least once throughout the jazz corpus. If we approximate this corpus with a library of jazz melodies and their corresponding harmonies, then we can generate harmonies for new melodies by fragmenting them into sequences, searching for those sequences in the library, and drawing on the harmonies used in any songs which contain those sequences.

Because we are not trying to generalize or synthesize information over the entire dataset, we expect to achieve good results with a library that is orders of magnitude smaller than a decent dataset for training a machine learning model. In particular, our library will have about a thousand entries, while most machine learning datasets contain tens or hundreds of thousands of entries.

Nonetheless, because of the finite size of the library, there will likely be instances where a perfect match cannot be found. We can address these cases by drawing on melody fragments which are similar, if not identical, to the target melody. In any case, it is likely that we will need to employ some simple tools of musical analysis to modify any matching harmonies to suit the target melody. Assuming that the library is extensive enough to minimize the instances where no perfect match is found, musical analysis will not need to be extensively relied

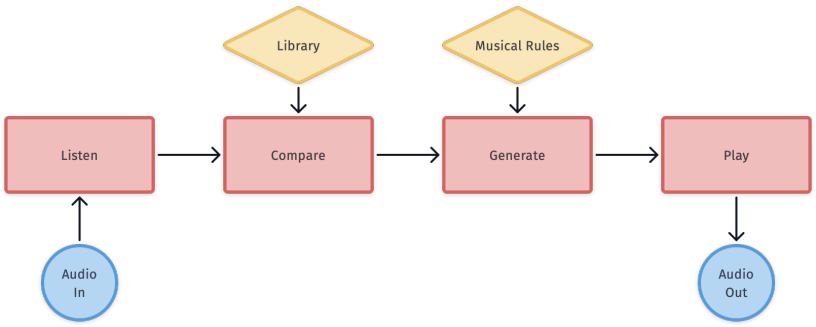


Figure 4.1: Block diagram of the system architecture. Rectangles represent system stages. Diamonds represent sources of external knowledge. Circles represent analog signals.

on, which means that we can settle for a relatively simple set of rules to help these cases.

4.2 SYSTEM OVERVIEW

In light of the preceeding discussion, we can flesh out the three main system tasks from Section 1.4 into four tasks:

1. Listen to a melody and transcribe it into a computationally useful form.
2. Compare the transcription to a library of known songs to attempt to find a similar, if not exact, match.
3. Using the best match along with simple musical analysis, generate a harmony appropriate for the melody.
4. Play the generated harmony in time with the musician.

This decomposition leaves us with four largely independent subproblems which suggest a useful structure for decomposing our system into components. We will think of the system as consisting of four stages, each of which is designed to solve the corresponding subproblem described above. We term these stages "Listen", "Compare", "Generate", and "Play". A block diagram depicting this architecture is shown in Figure 4.1. We next describe the design of each stage in detail.

4.3 LISTEN

The goal of the Listen stage is to transcribe a monophonic melody in real-time. This problem can be further divided into two subproblems:

1. Measuring and processing an audio signal to extract musically useful information.
2. Representing that information in a computationally useful format.

We refer to these two problems as "Extraction" and "Representation" respectively.

4.3.1 *Extraction*

We first define what information in an audio signal is musically useful. The easiest and most intuitive way to do this is to simply observe what information is conveyed through sheet music, which, at least for our system, is a complete representation of the song in question. For monophonic melodies, the unit of information is the note, which has a pitch and a duration. The rest of the information on a sheet generally applies to parts of or the entirety of the piece: tempo markings, time signatures, key signatures (which only influence pitch), and dynamic markings. For the purposes of generating an accompanying harmony, the only important quantities are pitch and duration.

4.3.1.1 *Pitch*

Determining the pitch of a note is equivalent to determining the frequency of a narrowband audio signal. Real-time frequency determination is a well-investigated problem and there exist a variety of solutions [17], [9]. There are a few notable assumptions that our project makes which help to select a particular method:

1. The melody is monophonic (see Section 3.1).
2. The audio source is undistorted.
3. We have unrestricted access to sufficiently fast computational resources.

Assumption 2 can be justified by using sufficiently high-quality microphones and by playing in a sufficiently quiet space. Assumption 3 is justified by the current state of consumer computing hardware. With these assumptions in mind, we choose to use the Fast Fourier Transform (FFT) method as described by Kuhn, who notes in particular its application to monophonic sources [9]. Furthermore, we can disregard the computational complexity of the method due to Assumption 3.

Although there is a decent bit of computational work required by the FFT method, it is conceptually very simple. We sample the audio signal at a sufficiently high sampling rate, compute the FFT of that sampled signal, and then use a peak detection algorithm to find the main harmonic frequencies. The harmonic with the lowest frequency

is termed the "fundamental", which is the most musically salient (although it is does not always have the largest magnitude), and thus determines the pitch of the notes. By repeatedlity computing FFTs and running a peak detector to capture the fundamental frequency, we can determine at any moment in time what pitch (if any) is currently being sounded.

The size of the FFT required to achieve a frequency resolution of 10 Hz can be calculated directly. Note that this resolution is not necessary to distinguish between two simultaneous pitches, since we are working with a monophonic melody. Although there are harmonics for each note which we could model as a series of narrowband signals, these signals separated by hundreds of Hertz, which is much greater than our resolution requirement. In particular, issues of spectral leakage and sidelobe size are unimportant; the narrow mainlobe and higher sidelobes produced by using a rectangular window are perfectly acceptable. Spectral sampling, however, is important. The frequency bins of the transform must have a maximum size of 10 Hz. The length N of an FFT required to achieve a spectral sampling resolution f_r given a sampling frequency f_s is

$$N = \frac{f_s}{f_r}.$$

Assuming the specified minimum sampling frequency of $f_s = 2.64$ kHz, an FFT of length $N > 264$ is required to achieve a frequency resolution of < 10 Hz.

4.3.1.2 *Duration*

Because we are working with monophonic melodies, the duration of a particular note n can simply be defined as the time between the start of note n and the start of the next note n'. This definition allows us to reformulate the problem of duration measurement as the problem of onset detection, which has been widely studied. Sandler et al. present a number of possible solutions to this problem [1]. The general outline of all of these solutions is to reduce the raw audio waveform to a more structured signal with prominent peaks at locations in time where note onsets occur. These peaks can then be found using common peak detection algorithms. Thus, the main design consideration is choosing how to reduce the waveform.

As stated in Section 3.1, we assume that our system will take as input melodies played by a piano. One of the main benefits of this assumption is that a piano is played percussively: its strings are sounded by a series of small hammers controlled by the keys. One physical consequence of this is that at then instant when a note is played, there is a characteristic burst of high-frequency noise which quickly decays as the fundamental and most prominent harmonics come to dominate the timbre of the note.

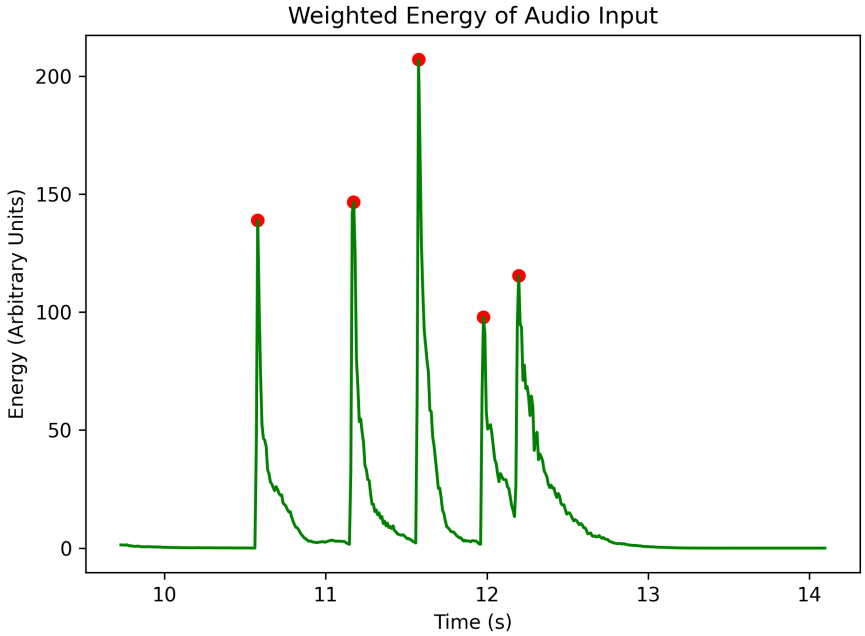


Figure 4.2: An example of a reduced waveform used for determining note onsets. Detected peaks are indicated by red dots.

Because this high-frequency noise is disproportionately likely to be found at note onsets, a very promising strategy for reducing the audio waveform to a peak detectable signal is to calculate a time-dependent weighted energy measurement. We first calculate a series of FFTs using a short window over the raw audio signal at equally spaced intervals. We then compute a frequency-weighted energy measure for each FFT using the following weighting equation:

$$E = \sum_k k^2 |X[k]|^2,$$

where $X[k]$ is the one-sided FFT of the signal and k is the discrete frequency index. The quadratic weighting in k heavily skews the measurement to favor high frequencies, resulting in very sharp peaks which are trivial to detect. See Figure 4.2 for an example.

These FFTs need not be the same as those used for pitch detection. In particular, using a shorter window length is preferable for onset detection because it provides a better time resolution. On the other hand, we prefer a longer window length for pitch detection in order to achieve a better frequency resolution.

Having computed to the onset of each note, we simply look at the differences in onset times of adjacent notes to determine duration. Another benefit of this onset detection method is that it facilitates the process of creating a transcription of notes, since we have determined not only the duration of each note but also its start time. Furthermore, because we have maintained a record of what pitch is being sounded

at every time step[1], we can simply read off the pitch of a particular note by looking at what pitch was being sounded during a given note onset.[2]

4.3.2 *Representation*

To address the question of how to best represent the melody for computation, we should first consider the computations which must be performed in the Compare and Generate stages. The design of these stages is discussed below in greater detail, but in brief, the Compare stage attempts to match the transcribed melody to known melodies using some similarity metric and the Generate stage uses these matches along with the played melody to produce a harmony. As discussed above, the relevant quantities for melodic identification are pitch and duration. A widely-used technical standard for electronic musical instruments is MIDI (Musical Instrument Digital Interface) [13], which includes a communications protocol for transmitting notes. In particular, this protocol comunicates the pitch, duration, volume, and less relevant markings for each note. Because of the universality of the MIDI standard, there is existing software support for it in many languages and environments. It has been used in previous work for similar applications [19] and it allows for easy testing of the components downstream of the Listen stage without requiring that the Listen stage be correct, since a MIDI controller (such as a MIDI piano keyboard) could be used as a direct input source to the Compare stage. In the complete system, however, the pitch and duration information produced in the Extraction step is used to construct a MIDI stream in the Representation step.

4.4 COMPARE

The goal of the Compare stage is to find a melody from a library of known songs that is as close as possible to the observed melody, so that the harmony corresponding to the known melody can be used as a starting point for the Generate stage to produce a harmony for the observed melody. If the library is very large while still being efficiently searchable, then we can almost always expect to find a harmonically

1 The timesteps between pitch measurements are larger than those between onset measurements, due to the differing window lengths used for the FFTs. Thus, multiple "onset timesteps" map to a single "pitch timestep".

2 In practice, it is best to actually take the pitch that was measured a few pitch timesteps after the detected peak in the weighted energy curve. The presence of high-frequency noise which allowed for straightforward peak detection also has the side effect of muddying the spectrum of the note. A note enters the "sustain" phase a few timesteps after onset, and it is then that fundamental frequency is most likely to have the largest magnitude. In any case the problematic high-frequency energy will have died down by then.

correct accompaniment for the observed melody. The construction of this library is feasible because the number of relevant jazz standards only numbers in the thousands.

The task of building this library can be broken down into two subtasks:

1. Define a metric that meaningfully describes how similar two melodies are.

2. Construct a database that is efficiently searchable using that metric.

Both of these tasks have come up in computational music problems other than automatic accompaniment, including those addressed by query-by-humming systems [4]. One design that is particularly applicable to our problem is that proposed by Uitdenbogerd and Zobel [19].

4.4.1 *Similarity Metric*

A useful metric for evaluating the similarity of two melodies is N-gram indexing, as proposed by Suyoto and Uitdenbogerd [18]. In this context, an N-gram is a representation of a melody fragment consisting of N notes using intervals. Each note is represented by the interval between it and the previous note, with the first interval in an N-gram always being 0 and all intervals taken modulo 12 (i.e. an octave is the greatest possible interval). To compare two melodies, each melody is first maximally decomposed into N-grams by creating an N-gram starting at each note in the melody. The number of N-grams which are shared by the two melodies is their similarity score. When searching for the most similar melody among many, the melody with the highest similarity score is taken as the closest match.

This method performs competitively with more complex methods, and its simplicity makes it attractive for comparing a large number of melodies. Furthermore, because intervals are considered instead of absolute pitches, this method lends itself well to the task of finding an appropriate harmony for a melody. Harmonies can always be transposed into a different key to match the starting note of the melody without affecting the correctness of the harmony. Suyoto and Uitdenbogerd achieve good results using $N = 5$, but the particular choice of N can be determined experimentally and easily modified for any given application.

Note that this method does not actually take note duration into account. Two melodies receive identical similarity scores no matter how fast or slowly the notes are played relative to each other. Again, this is a benefit for our system, since changing the relative durations of notes does not affect the harmonic correctness of its accompaniment, as long as the rhythm of the harmony is adjusted accordingly.

One challenge with this method is that many N-grams have to be compared. Because our system must function in real-time, we need to ensure that these comparisons can happen very quickly. Note that even though our library only contains $\sim$1000 songs, each song can potentially contain hundreds of N-grams. Instead of explicitly computing the similarity metric, we can utilize programming shortcuts to facilitate theis process. N-grams can easily be represented as strings, which allow us to leverage existing software packages for string "fuzzy" matching. These packages allow the database to be pre-processed once before the system begins its normal execution, such that all matching can be performed over an optimized indexed set. String fuzzy matches do not perfectly correspond to the similarity metric of measuring the number of N-grams which are different, but it will always find an exact match if it is available. If no match is available, the details of the similarity metric aren't that important; all we are looking for is a close match, which string similarity metrics will get us.

4.4.2 *Database Structure*

Many music retrieval systems do not dedicate any attention to the actual database structure [19], [4], [21]. This is likely due to the fact that many of these systems do not need to perform time-sensitive searches, as our application requires. Assuming that we were to use a very simple database structure, such as that which could be implemented with a hash table or other fundamental data structure, the speed of the search is a function of how many comparisons need to be made, which is in turn a function of how many items are in the database.

As stated above in Section 4.1, the database should contain most common jazz standards. Collections of standards exist in the form of "Real Books" which musicians use as reference material in jam sessions and other performance environments to quickly review or remind themselves of a melody or chord progression. These books contain around 400 songs each, and there are multiple volumes availabe from common publishers [10]. These sheets normally have somewhere in the range of 60 notes in the melody, which implies 12 5-grams per song, meaning that we would need to perform roughly 4800 string comparisons (with 5 character strings) per song. Even on commodity hardware, this number of comparisons is by no means expensive and can easily be calculated in the approximately 1 second allocated for harmony generation, especially using the pre-processing capabilities and optimized matching speed of fuzzy string matching algorithms. As a result, we do not need to design a particularly complex database structure to facilitate this matching process.

For every N-gram, the database must contain the harmony corresponding to that melody fragment, the duration of each note in the melody fragment (in beats), and the absolute pitch of the initial note in

the fragment (written as a MIDI pitch). We can notate the harmony by simply indicating what chord is played over each note in the N-gram, even if a musician would not actually sound the chord at every note (but instead hold the chord over multiple notes), leaving the details of reproduction to the Play stage. Although we could devise a general, relative interval notation similar to that used for melody N-grams, it is simpler to just write the chords in their original key. If the Generate stage needs to transpose the chords, then it can easily do so using the absolute pitch of the initial note. Similarly, the Generate stage can use the durations of each note to help determine the duration of each chord in the harmony.

To search the database, the Compare stage must compare every entry in the database to the observed N-gram. Assuming that we are able to find a sufficiently close match, what we wish to pass to the Generate stage is not the harmony which accompanies this match, but rather the harmony which accompanies the melody a few N-grams *after* the match in the same song. Because the Generate stage is producing a harmony for a melody which will be played in the immediate future, the most useful information is the harmonies which occur after the kinds of melodies that have just been played. In order to facilitate this look-ahead functionality, we must also store with each N-gram the name of the song it is from as well as its positional index within that song. In practice, we are interested in the harmony of the N-gram immediately following the best match, since the entire melody comparison and generation process will generally occur during the time between which melody fragments corresponding to N-grams are played.

For a schematic illustration of the database structure, see Table 4.1.

4.5 GENERATE

The goal of the Generate stage is to produce a musically-acceptable harmony which can be played over the melody. Despite this being a summary of the system as whole, the Generate stage will perform relatively little work in the simplest case. If the Compare stage successfully identified a very similar match, then the Generate stage has been passed a harmony that should match the melodic structure well, although it will likely need to be at least transposed to match the absolute pitch of the observed melody, since the Compare stage only evaluates the similarity between interval patterns. In practice, because string fuzzy matching techniques are being employed and because of the structural similarity of N-gram strings, a fuzzy match will always be found, which means the Generate stage will always have a harmony to work with. This means that the only necessary musical work that the Generate stage needs to perform is transposition to match the initial note of the target melody. Because we are assuming that the

N-GRAM	HARMONY	DURATION	INITIAL	INDEX	SONG
0	C	4.5			
9	C	1.5			Take
-9	C	1.0	67	0	the "A"
5	C	1.0			Train
4	D7b5	0.5			
0	C	1.5			
-9	C	1.0			Take
5	C	1.0	76	1	the "A"
4	D7b5	0.5			Train
-8	D7b5	7.5			
0	C	1.0			
5	C	1.0			Take
4	D7b5	0.5	67	2	the "A"
-8	D7b5	7.5			Train
1	Dm7	4.0			

Table 4.1: Database structure.

library is sufficiently large, we can also assume that the harmonies which are found via fuzzy matching are, at worst, acceptable.

It is important to note that the harmony which is produced by the Generate stage will not be played over the melody which the Listen stage has transcribed, but rather over a melody which is expected to be played in the immediate future. Although the system is not explicitly referring to a probability distribution on that future melody, the patterns contained in the library used by the Compare stage provide useful information to guess at the kind of melody which will come next. This information is of course worse if the matches are not exact.

Finally, unlike the Compare stage, the Generate stage must take into account the rhythm of the melody, that is, the durations of its notes. This is because the duration of the chords for its generated harmony must also be specified. The Compare stage works in units of N-grams. It doesn't make sense, however, for the Generate stage to also work in units of N-grams, because the duration of an N-gram is unspecified. It is possible that the musician will play a series of notes that are very short, which means that very little harmony needs to be generated, or very long, which might normally call for a few chord changes. For this reason, the Generate stage must produce a harmony which is longer than will likely be needed, with the hope

that all of it will not be used as more notes are transcribed by the Listen stage. If the musician slows down to such an extent that the harmony buffer produced by the Generate stage is exhausted, then the musician will have more than likely stopped playing the melody, as such dramatic tempo changes don't normally occur in jazz. If the musician then continues to play at that slower rate, then the Generate stage will begin producing harmony buffers that satisfy that tempo. The same problem does not occur with tempo increase, as buffers can always be thrown out and updated as more notes come in more quickly then expected.

4.6 PLAY

The goal of the Play stage is to reproduce the harmony returned by the Generate stage with the musician's melody in real-time. Because the musician is not going to play at a perfectly constant tempo, but is going to vary slightly throughout the song, intentionally or otherwise, the system must pick up on these variations in order for the reproduction to sound natural. This problem is termed "beat tracking" and has been studied previously [5], [8]. We will use the solution proposed by Dixon and Cambouropoulos [5], referred to as multi-agent beat tracking, as it has been shown to have very good results with monophonic MIDI inputs.

Although this task involves "listening" to the audio input, we have separated this functionality from the Listen stage because it does not affect the process of melody matching and harmony generation, which were the tasks that the Listen stage is designed to facilitate. Beat tracking is a task which is solely relevant to the Play stage. It will, however, draw on information collected in the Listen stage, in particular the note onset times.

4.6.1 *Beat Tracking*

The multi-agent beat tracking system works in two main steps. When a melody is first played, the difference in time between notes, referred to as an *inter-onset interval* (IOI), is recorded for all pairs of notes. These IOIs are then clustered with other IOIs that are within a certain tolerance. Each cluster is used to form a hypothesis for what the true *inter-beat interval* (IBI) is, with the hypothesis being the average of the IOIs it contains. The goal is then to pick which cluster is most likely to have the correct IBI hypothesis. For an illustration of different beat intervals and their sorting into clusters, see Figure 4.3, borrowed from Dixon and Cambouropoulos [5].

After these initial hypotheses have been computed, an "agent" is created to compare the prediction of a particular hypothesis to the observed onset times of new notes. If an agent is consistently able to

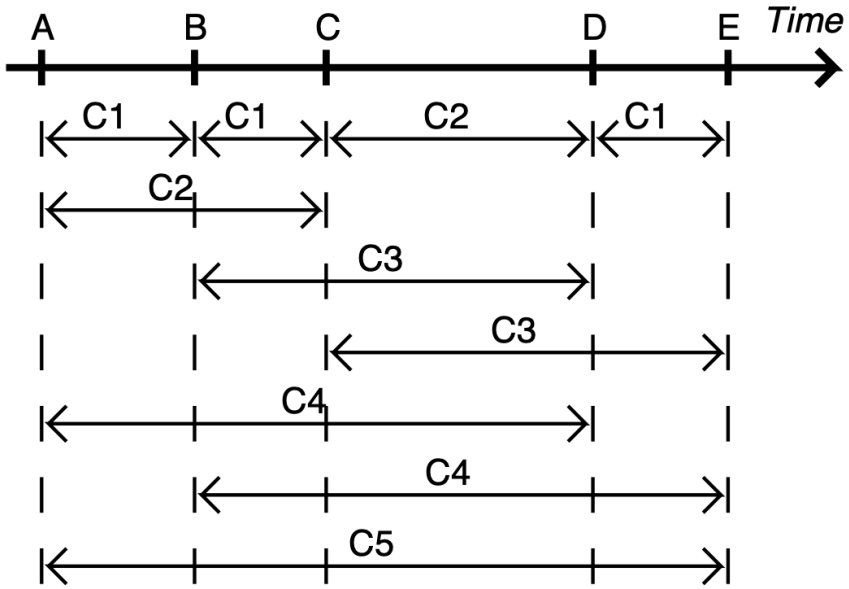


Figure 4.3: Beat diagram illustrating 5 different IOIs which are labeled by their corresponding cluster. The clusters are named C1 through C5. Borrowed from Dixon and Cambouropoulos [5].

predict the onset events well, its hypothesis is scored higher, as it is more likely to be a good estimate of the IBI. The relative rankings of different agents can evolve over time, but at any time, the hypothesis of the agent with the highest ranking is taken as the best estimate of the melody's IBI.

Updating the hypothesis scores with equal weight for every note that is observed can be problematic, particularly for melodies with a lot of notes which subdivide the IBI (e.g. melodies with many eighth notes). To help ensure that the best hypothesis is not a fractional multiple of the IBI, musical knowledge can be applied to appropriately weight the score updates [5]. It is known that lower-pitched, longer notes are more musically "salient". In particular, these notes are more likely to fall on true IBI locations than higher-pitched, faster notes. Thus, when these notes are successfully predicted by a particular hypothesis, its score should increase more than it would for having successfully predicted other notes. The function we use to calculate the salience of a note with duration d seconds and a pitch p, where p is a MIDI number ranging from 0 to 127, is

$$S(d, p) = d(88 - p).$$

The salience of a given note is exactly how much the score of a hypothesis is modified by when it successfully predicts a note's onset time. Note that MIDI number 88 corresponds to the musical note E6, which is the upper bound of our pitch input range. Thus, the salience of a note is always non-negative.

To briefly summarize the relationship between the rhythmic work performed by the Generate and Play stages, the Generate stage determines how many beats each chord should receive, while the Play stage uses its estimate of the IBI to determine how much time each beat should receive, as well as when beats should occur.

5 SOFTWARE IMPLEMENTATION

The system will be implemented in Python. Python has an extensive number of signal processing and scientific computing packages, including packages for MIDI interfaces [20], [14]. A series of prototypes will be constructed, each designed to demonstrate and validate a critical portion of the pipeline. The complete code repository can be found here:

https://github.com/agmariona/JazzMaster

The one software engineering detail worth mentioning here is how the system handles the necessary parallel processing. The sysem must be simultaneously listening to melodies, playing harmonies, and generating new harmonies. To achieve this, we use a simple multithreading paradigm in which the Compare, Generate, and beat tracking work from the Play stage run in the parent thread while Listen and harmony playback from the Play stage each run in their own separate child threads.

5.1 PROTOTYPES

Five prototypes were implemented over the course of this project.

Prototype I consists solely of the Listen stage, with the goal of validating the system's ability to take in real-world audio signals and transcribe them in accordance with the stated specifications. This prototype will be evaluated using an acoustic musical instrument (such as a piano) and consumer-grade microphones.

Prototype II consists solely of the Compare stage, with the goal of setting up the library search system and validating the system's ability to work with MIDI data. Since this prototype is isolated from the Listen stage, it will be evaluated using an electronic musical instrument (i.e. a MIDI keyboard controller) which is capable of directly delivering a stream of MIDI notes. Both melodies which are and are not in the library will be tested, in order to test the similarity scoring functionality.

Prototype III consists of the Compare and Play stages, with the goal of validating the system's ability to harmonize a melody which is in the library by reproducing the known harmony exactly. As with the second prototype, this will be evaluated using a MIDI controller. Unlike the second prototype, we will only be concerned with melodies that are exactly present in the library. With the integration of the Play

stage, we will now be outputting audio, which will be done using consumer-grade speakers.

Prototype IV will consist of the Compare, Generate, and Play stages, with the goal of verifying the system's ability to generate a harmony when given a melody which has a good, but not exact, match in the library. Thus, the generate stage will have to perform simple transformations with a minimal amount of musical knowledge. Since the Listen stage has still not been integrated, this prototype will also be evaluated using a MIDI controller.

Prototype V will consist of all four stages, with the goal of complete system integration. If earlier prototypes have been successful, then the testing and verification performed on the fifth prototype will be limited to ensuring that its behavior is as successful as the fourth prototype, but using an acoustic instrument instead of a MIDI controller. At this point, the system is functionally complete, meeting most of the technical specifications. The only task which will not have been verified in earlier prototypes is the generation of complex harmonies for melodies with distant matches. Since these are the minority of melodies based on our problem formulation, and since the musical rules are the least technical part of the system, this task was left for last. Nonetheless, the fifth prototype should be able to meet the correctness requirements outlined in Section 3.2.

5.2 BUILDING THE LIBRARY

In Sections 4.1 and 4.4, we discussed the motivation behind and the design of the database containing the library of jazz standards which serves as the main source of prior knowledge that the system draws upon when generating harmonies. Although the design of the database and the method of similarity scoring is conceptually interesting and technically significant, the gathering and formatting of the data which goes into that database is less interesting, and for that reason we only briefly summarize here how we went about that process.

A variety of commerically-available score arrangement and formatting programs, such as Sibelius and Finale, can export scores in the MusicXML format [7]. Like traditional XML files, these are hierarchical, tag-based structures which are transparently readable as plain text files (i.e. they are not compressed or complexly encoded). Furthermore, a large number of transcriptions of popular music in the MusicXML format is easily found online from sources such as Wikifonia[1].

With this in mind, we wrote an automated MusicXML parser which takes as input an unmodified MusicXML file and outputs a text file in a directly parseble format. We then wrote a script to take these text files and generate a database with the structure outlined in Table 4.1.

1 Wikifonia is defunct as of 2013, but archives of its MusicXML library can still be found online (http://synthzone.com/files/Wikifonia/).

The MusicXML file library we downloaded contained songs in a wide variety of genres, so we wrote another script to select any songs in the corpus of jazz standards as compiled by the six volumes of *The Real Book* published by Hal-Leonard [10].

6 EVALUATION AND VERIFICATION

6.1 VERIFICATION APPROACH

Reviewing the technical specifications discussed in Section 3.2 and summarized in Table 3.1, we find that it is significantly easier to design strategies for verifiying the specifications on the input and on processing constraints than it is to verify the correctness of the system.

Our strategy for verifying that our system meets our specifications is two-fold. We first independently evaluate each stage of the system described in Section 4. Because of the modularity of the stage designs, this allows for simpler, more direct, and more useful tests. After each stage (or groupings of a few stages) have been independently verified, we then evaluated the correctness of system as a whole.

For each series of tests, we highlight on which prototype the tests were performed. Not all prototypes, however, are allocated tests. This does not mean that these prototypes were not debugged or evaluated, but rather that their success is not evaluated against the system technical specifications. In particular, the value of prototypes II, III, and IV for implementing and verifying the system is in dividing up the build process into manageable chunks and in asserting the feasibility of the design at each step.

6.2 STAGE EVALUATION

6.2.1 *Listen*

Verifying the correctness of the Listen stage amounts to verifying that all of the input specifications have been satisfied. To verify the three specifications on input range (pitch range, tempo range, and volume range) and input resolution (frequency resolution and temporal resolution), we characterize the "frequency response"[1] of the system by chromatically playing every note in the pitch input range. This can be thought of as a kind of frequency sweep. We performed sweeps at the extremes of both tempo and volume, as well as in the middle of those same ranges. Because these tests involve the Listen stage alone, they can be performed on Prototype I.

1 Strictly speaking, our measurements do not properly produce a frequency response, since we are not attempting to produce a continuous sweep over all frequencies. Nonetheless, the concept is similar and this type of measurement is better suited to the actual problem, since the input space is fundamentally limited to musical notes.

Responses were produced for tempos of 40 BPM, 120 BPM, and 240 BPM, and for volumes of roughly 50 dB, 70 dB, and 90 dB. Every parameter combination of tempo and volume was tested, for a total of 9 combinations. See Figures 6.1, 6.2, and 6.3 for visualizations of the Listen stage's response to these inputs.

In these diagrams, a black dot represents an input note, a blue dot represents a note transcribed by the Listen stage which correctly corresponds to a particular input note, and a red "x" represents a note transcribed by the Listen stage which is incorrect (i.e. which does not correctly correspond to any input note). For the input notes, the horizontal axis corresponds to the time at which the note was played and the vertical axis corresponds to the fundamental frequency of that note. For both correctly and incorrectly transcribed notes, the horizontal axis corresponds to the detected onset time and the vertical axis corresponds to the measured fundamental frequency of the note.

A note can be transcribed incorrectly for one of three reasons: the measured frequency of the note can be incorrect (see Figure 6.1b at around 6 s), the note can be detected as occurring too late compared to the onset of the input note (see Figure 6.1c at around 7 s), or a note can be detected as occuring multiple times for a single input note (see Figure 6.3c at around 0.5 s). It is also possible that a note will not be transcribed at all; this is an error which is visually represented in these diagrams by the absence of any marker.

Qualitatively analyzing these results, a few patterns are clear. First and foremost, we find that the performance of the Listen stage is significantly better at working with moderate volumes than either very soft or very loud volumes. That being said, it is able to handle loud volumes decently if the tempo is moderate. When both the tempo and volume get pushed to extremes, however, the performance of the listener decreases significantly. In particular, quiet volumes present a lot of challenges for correctly identifying pitch.

To help quantify these conclusions, we ran multiple trials with the same testing parameters and, for each trial, calculated the ratios of correctly measured notes to total input notes and of incorrectly measured notes to total input notes. The distribution of these ratios can be seen in Figures 6.4 and 6.5. In these diagrams, the bars are color-coded such that trials with the same tempo parameter have the same color, while trials with the same volume parameter have the same intensity shading.

Note that, ideally, the ratio of correctly measured notes would always be equal to 1 and the ratio of incorrectly measured notes would always be equal to 0. Furthermore, while the ratio of correctly measured notes is bounded to the range $[0, 1]$, the ratio of incorrectly measured notes has no such upper bound. There can be many incorrectly measured notes if, for example, a single input notes trigger multiple notes to be transcribed. As such, the ratio of incorrectly mea-

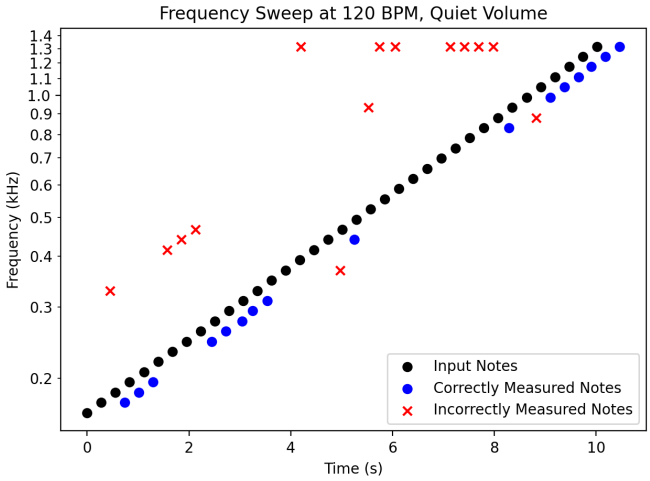
sured notes can grow quite large in some parameter combinations. In particular, we see outliers of this ratio in Figure 6.5 at loud volumes. These outliers must be due to exactly that type of error, since we see that those same trials did not have a ratio of correctly measured notes near 0. Thus, the total number of transcribed notes for these trials was significantly greater than the total number of input notes.

As formulated in Section 3.2, the behavior of the Listen stage must be perfect: across all input ranges, every input note must be correctly transcribed and there must be no incorrectly transcribed notes. It is obvious that this prototype does not achieve that result. Nonetheless, it does achieve good results for moderate input parameters, namely, moderate tempos and moderate volumes. Importantly, these are the input ranges which are most commonly found in real music; very few songs are played quite as fast as 240 BPM or as soft as 50 dB. Furthermore, the behavior of the Listen stage is completely opaque to the user; the musician hears the harmony produced by the system, but they will never see what the ratio of correctly measured notes was for a particular melodic phrase.

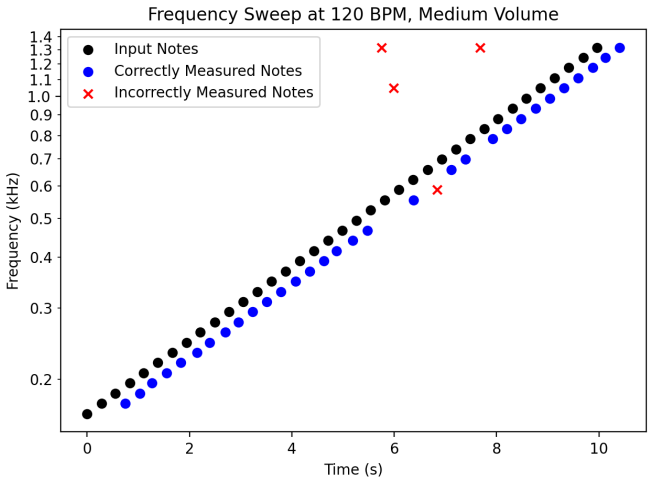
Thus, while these input specifications provide a practical framework for evaluating the performance of the Listen stage, offer insight into the behavior of the system, and overall are very useful from an engineering perspective, they are not the most critical specifications from the user's perspective. For further discussion of this point, see Section 6.3.

6.2.2 *Compare*

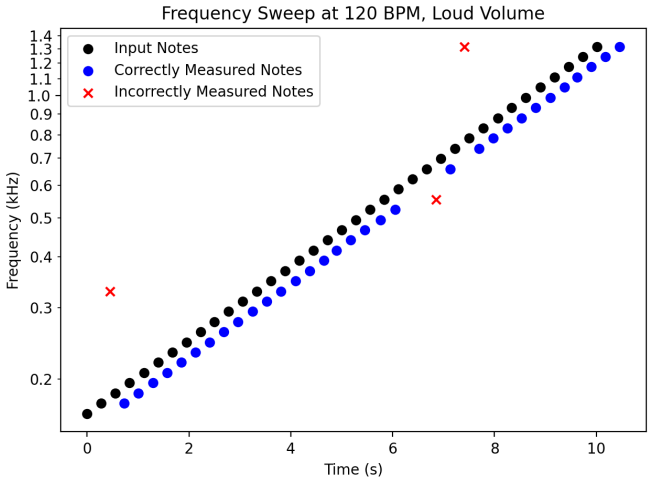
There is little to investigate particularly in the Compare stage with respect to correctness or meeting specifications. It should be noted that the Compare stage and the Generate stage must both be able to execute within the approximately 1 second allocated by the Generation time. The time taken by the Compare and Generate stages is roughly constant across all inputs, so it is sufficient to consider this specification satisified if the Generate and Play stages are capable of achieveing good correctness results, because they could not do so if the Compare stage was exceeding its allotted execution time.



(a) 120 BPM, ∼50 dB

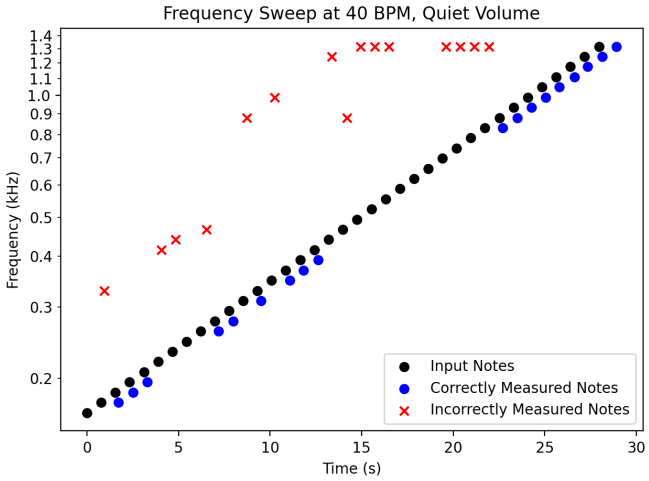


(b) 120 BPM, ∼70 dB

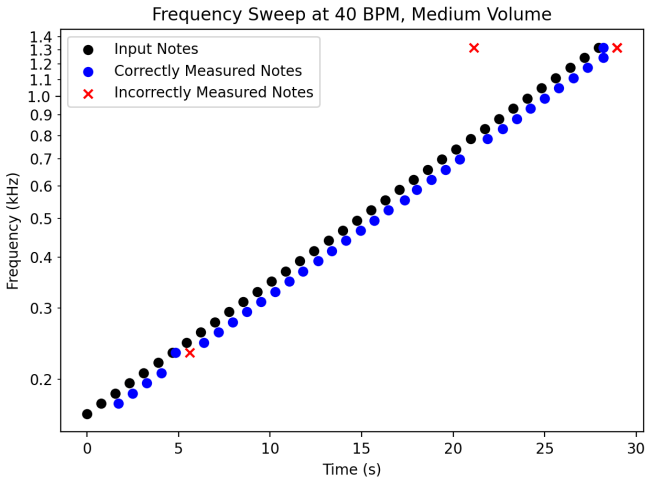


(c) 120 BPM, ∼90 dB

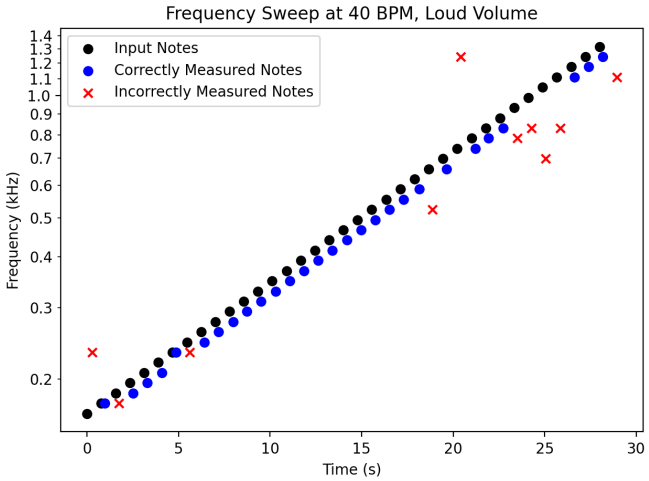
Figure 6.1: Frequency sweeps at 120 BPM.



(a) 40 BPM, ∼50 dB

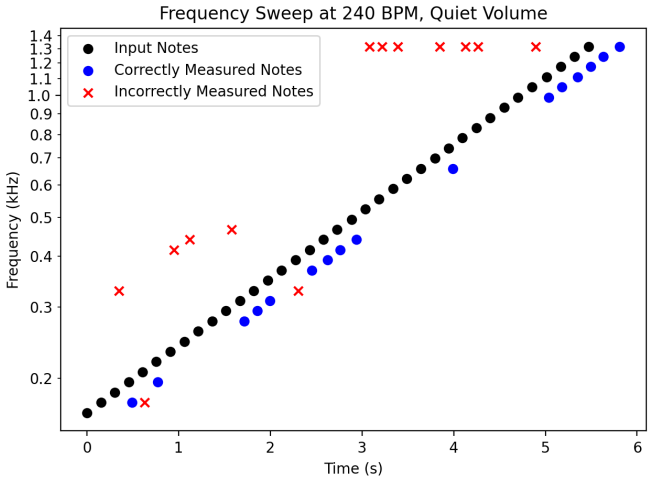


(b) 40 BPM, ∼70 dB

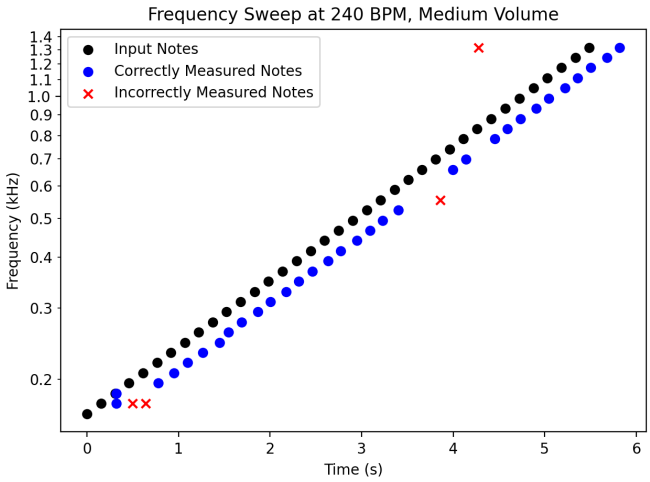


(c) 40 BPM, ∼90 dB

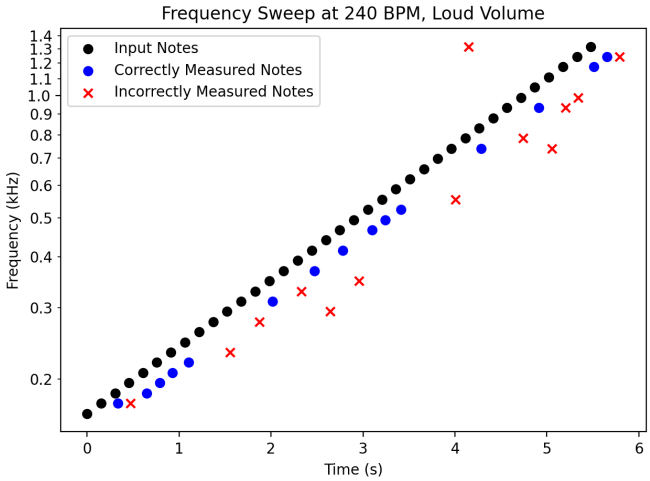
Figure 6.2: Frequency sweeps at 40 BPM.



(a) 240 BPM, ∼50 dB



(b) 240 BPM, ∼70 dB



(c) 240 BPM, ∼90 dB

Figure 6.3: Frequency sweeps at 240 BPM.

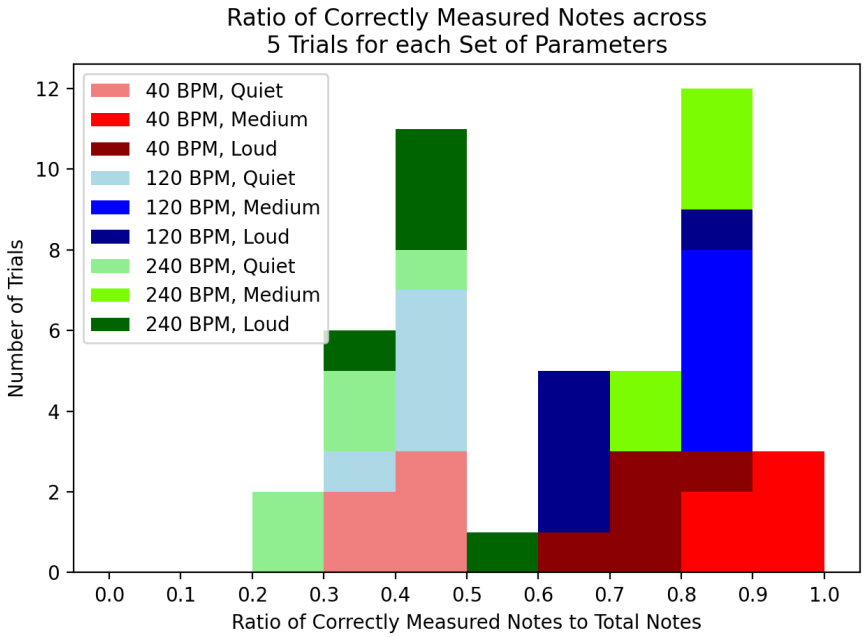


Figure 6.4: Distribution of the ratio of correctly measured notes to total input notes.

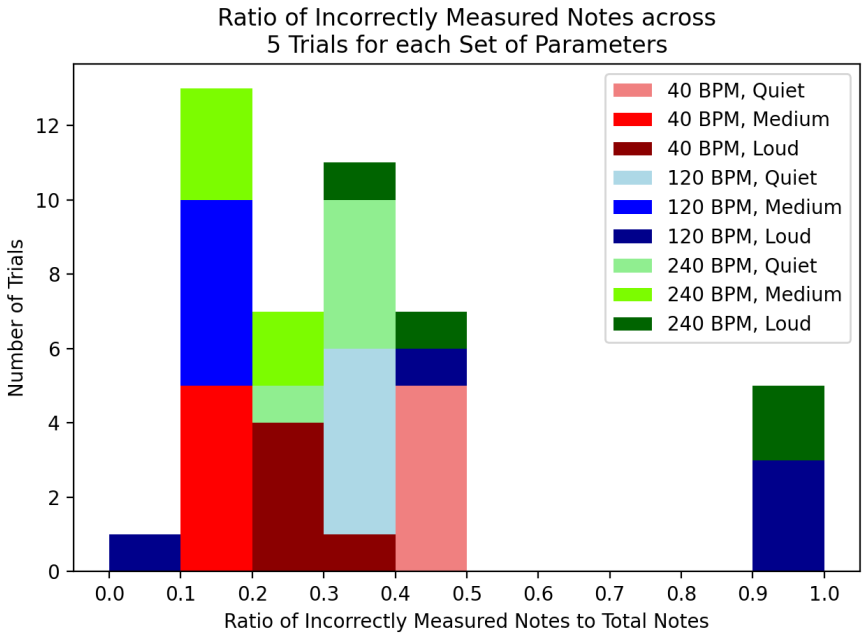


Figure 6.5: Distribution of the ratio of incorrectly measured notes to total input notes.

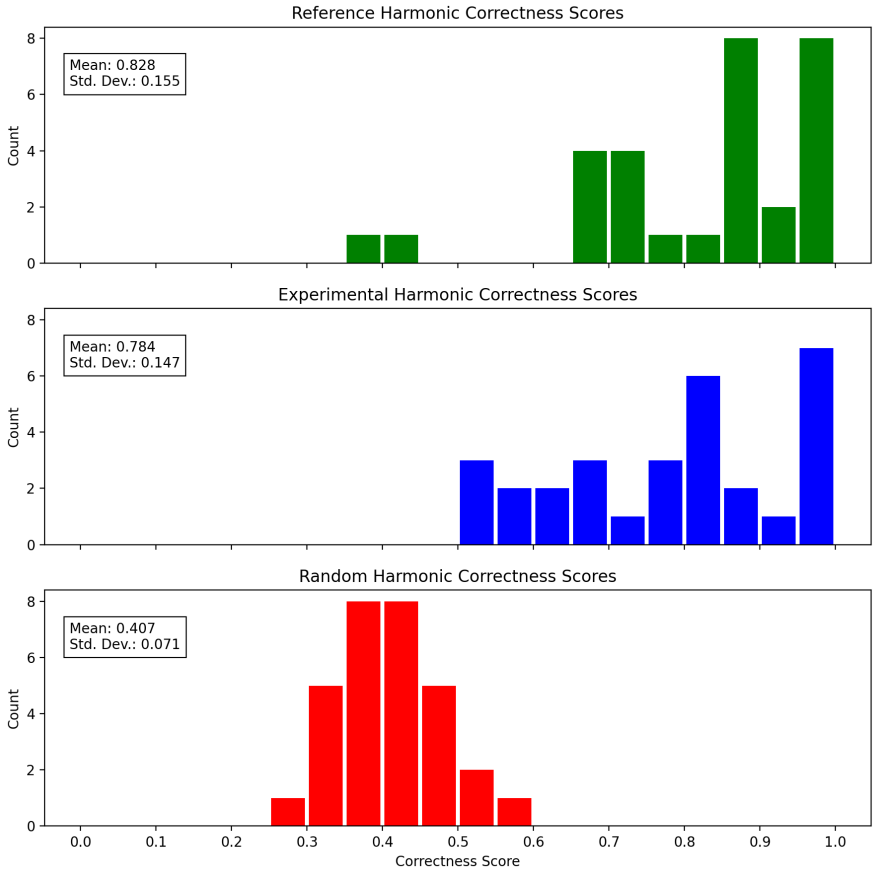


Figure 6.6: Distributions of harmonic correctness scores for reference, experimental, and random accompaniments.

6.2.3 *Generate*

The main question of verification for the Generate stage is related to harmonic correctness. As formulated in Section 3.2.3, every accompaniment produced by the system should achieve a harmonic correctness of at least 90%. To evaluate whether this specification is met, we randomly selected 30 songs from the library and played these songs as inputs for Prototype V (the complete system). We then calculated the harmonic correctness score of the reference accompaniments in the library and of the experimental accompaniments produced by the system. Additionally, we also generated and scored 30 random accompaniments, for which the chord timings, roots, and qualities were all chosen randomly. The distribution of these scores can be seen in Figure 6.6.

It is immediately apparent that this metric for harmonic correctness is imperfect, since many reference accompaniments have correctness scores less than 1. There are nuances to real music which are not captured by the relatively simple rules we used for scoring. This is the inherent challenge in trying to quantify what makes music "good":

any collection of simply-stated rules which positively evaluates a large sample of real music is likely to be so liberal as to positively evaluate essentially any sample, while any rules which are conservative enough to reject obviously poor accompaniments will likely be too rigid to positively evaluate many samples from real music. In the process of performing these tests, it became clear that defining a metric which manages to both achieve good results on real music while rejecting clearly bad music would be a project unto itself.

Nonetheless, even though our metric is imperfect, it is not useless. In particular, the reference correctness scores are still significantly skewed towards 1 and the shape of the distribution is very similar to that of the experimental scores. In addition, the distribution of scores on random accompaniments is significantly worse than the experimental scores. Comparing the reference with the experimental distribution, a two-sided t-test yielded a p-value of 0.2639. This should be interpreted as meaning that there is about a 26% chance that we would observe a difference this large in means even if the two population means were identical. Similarly, the 95% confidence interval on mean difference was $[-0.03407, 0.12207]$, which means that we are 95% confident that the mean difference lies in this range, which notably includes 0.

In summary, although we cannot claim that our system always achieves 90% harmonic correctness as we have defined it, we have shown that such a claim is fraught at best. Instead, we can more meaningfully say that, if we compare the score distributions of real jazz accompaniments to the score distributions of accompaniments produced by our system, we are 95% confident that the difference in mean score is relatively small.

The results discussed above were produced by playing songs which were in the system's library. We conducted the same experimental procedure with the same sampling of songs after having removed all 30 of them from the library. For the distribution of harmonic correctness scores for the accompaniments generated by the system in this setup, see Figure 6.7. We can see by comparison with Figure 6.6 that there is little, if any, dependence on whether or not a particular song is in the library. This speaks to the robustness of the system, especially for improvised melodies, since the library is far too small to contain accompaniments for every possible interval pattern.

6.2.4 *Play*

Our evaluation of the rhythmic correctness of the Play stage follows a similar pattern as that of the harmonic correctness of the Generate stage, although it is fraught with fewer challenges.

As formulated in Section 3.2, the specification on rhythmic correctness requires that every accompaniment produced by the system achieve a rhythmic correctness of at least 90%. We use an identical

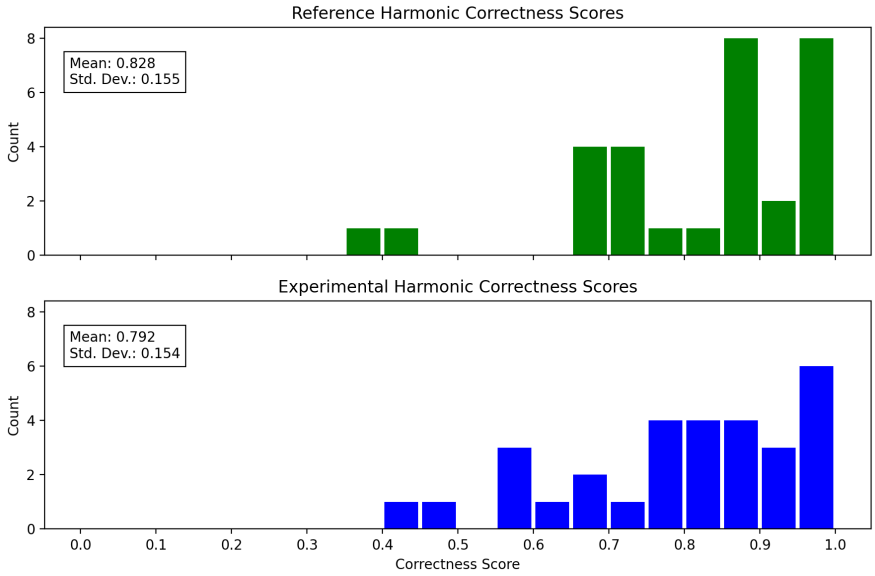


Figure 6.7: Distributions of harmonic correctness scores for reference and experimental accompaniments to songs not in the library.

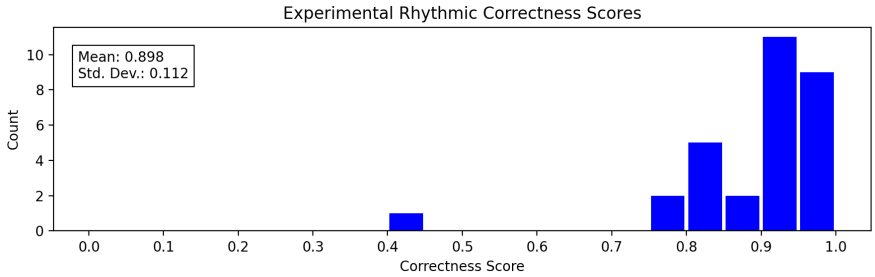


Figure 6.8: Distributions of rhythmic correctness scores for experimental accompaniments.

evaluation process to that used for harmonic correctness, down to using the same sampling of 30 songs. The distribution of the rhythmic correctness scores of the experimental accompaniments produced by the system can be seen in Figure 6.8. Note that scoring reference accompaniments is unnecessary; we can assume that the reference would play every chord perfectly in time. In reality, all of real music is indeed played in time (up to a tolerance window similar to the 80 ms window we used for our evaluations). Similarly, we do not need to compare against a random distribution to show that our system outperforms playing chords at random; it is not hard to imagine how jarring that would sound, and our metric will score such an accompaniment quite poorly.

Examining these results, we see that we do not meet our original specification of 90% rhythmic correctness on every song, but we do achieve an average correctness very close to 90% with a low spread. Excluding the one outlier, all scores are above 75%. Furthermore, that

one outlier was a relatively short piece, around 10 s long, while the typical length was around 60 s. The short length resulted in very few chords being played overall. Similarly, because the beat tracking methods used in the Play stage need time to converge on the true beat, adn because relatively few melody notes were played, the song likely ended before a good estimate of the beat could be reached. This caused the few chords that were generated to be played based on early, incorrect estimates of the beat, and these chords dominate the score. If the song were to be continued, we expect that these anharmonic chords would be outweighed by many more harmonic chords.[2] Thus, even if we can only declare an average rhythmic correctness of 90%, we are still qualitatively quite close to achieveing the original specification.

6.3 SYSTEM EVALUATION

The functionality and performance of the complete system follows directly from the evaluations performed on each stage individually. In particular, the behavior of the Listen stage is completely independent of all other stages, which means that the tests described in Section 6.2.1 are representative of the complete system even though they were performed on Protoype I. Similarly, although the functionality of the tests described in Sections 6.2.3 and 6.2.4 was particular to the Generate and Play stages, they were still performed using the full system. Considering the results of all the tests together, it is clear that the system does not fully meet the technical specifications as originally formulated in Section 3.2. The significance of these failures is described in the relevant sections above. Holistically, however, the system is indeed capable of handling a wide variety of jazz songs rather well, as indicated by the overall correctness results.

Moving beyond these quantitative measurements, we can still ask a few interseting, if qualitative, questions. The purpose of the system is to accompany a musician in the place of a band, or even more simply, to replace a single accompanist. A natural consideration is whether or not the musician would be able to tell that the harmony produced by the system was in fact produced by a computer. This "musical Turing test" would be far more meaningful than the harmonic and rhythmic correctness metrics we have defined, even if it would be far less quantitative. Nonetheless, succeeding at such a test would be a far grander achievement than meeting our specifications. Jazz musicians are distinguished by their creativity and spontaneity which this system seeks to emulat, but is ultimately very difficult to achieve.

2 It is interesting to note that this particular piece achieved a very high harmonic correctness score of 1. The Generate stage does not necessarily improve with time, so it likely produced only a few chords which happened to work well, with the song ended before more chords needed to be generated.

If such a test were to be conducted, we could poll a number of musicians and ask them to play a melody, improvise a short solo, and generally try and challenge the system with interesting inputs. We could ask the musician to rate the perfomance of the system on a sufficiently granular scale for various important metrics. These could overlap with our testing metrics, like harmonic and rhythmic correctness, but they could also be more qualitative. How surprising was the harmony? How reactive was the system to the musician? If the system was reactive, did it follow the musician's lead well?

From the perspective of designing and building a system, these are metrics that are hard to quantify and difficult to use as a basis for iteration, but for the purposes of using and enjoying a system, this kind of test provides an additional and important level of insight which is hard to gain otherwise. Even moderate or qualified success on such a test would be a massive endorsement of the system's viability and the legitimacy of the design methodology.

7 BUDGET

Because the system is being implemented entirely in software, production costs were minimal and mainly limited to hardware for testing and verification. For a summary of these costs, see Table 7.1. In particular, an external MIDI keyboard controller was used for debugging and verifying Prototypes II-IV, which receive a MIDI input directly because they do not include the Listen stage. While trying to build the library, we purchased a fairly large database of roughly 13,000 jazz standards transcribed in a proprietary XML format. It turned out that these transcriptions did not include any melodies, making them largely useless for our purposes (see Section 5.2), but we nonetheless acknowledge that cost here.

These products were all purchased with funds provided by the Active Learning Labs at Harvard. Technically speaking, these costs were not necessary for creating any of the prototypes built. In particular, I found that using a virtual MIDI keyboard was more useful than the physical MIDI keyboard. In any case, there is no reason that these figures should reflect on the cost of the system as a whole to any consumer or end user, although the cost of the system is not an important specification or consideration for the purposes of this project.

ITEM	USE	COST
MIDI Keyboard Controller	Testing Prototypes II-IV	$200
Jazz XML Files	Building the library	$20
TOTAL		$220

Table 7.1: Budget summary.

8

CONCLUSION

There are three main conclusions we can draw about the success of the project and the viability of its methods.

1. The general approach of generating harmonies from short melody fragments is viable.

Our library-based approach attempts to accomplish similar goals to those of machine learning models. We seek to produce new harmonies based on harmonies we've seen in the past, except unlike machine learning systems, we do not pay any of the upfront costs of training a neural network nor any of the long-term costs of having to store and work with an excessively large model. Additionally, we completely side-stepped the problem of needing to create a massive data set of jazz melodies which is suitable of learning tasks. The fact that our system was able to produce harmonies of similar correctness to real music is a strong endorsement of this methodology, even if there is still much room for improvement.

2. Reasonable results can be achieved using small data sets.

Machine learning details aside, the amount of knowledge our system works with is orders of magnitude smaller than many data sets. This implies a kind of density in musical information: knowing comparatively few songs can allow you to make inferences on many. This could be a meaningful insight even for systems which do not follow our approach directly, as it suggests a new way of looking at musical data. If our system can achieve good results with this small amount of data, then maybe there are techniques from statistics and machine learning theory which can be used to augment of small data set like ours into a data set which is sufficiently large enough for training a neural network.

3. Reasonable results can be achieved with an imperfect listener.

Even in the moderate regimes where we performed our system correctness analysis, the Listen stage did not perform perfectly all the time. Again, this says something about the robustness of music to corruptions for the purposes of generating harmonies. It is true that this is somewhat dependent on our metric for correctness, since it is obvious that permuting notes randomly and then generating a harmony based on those permutations would not necessarily produce a good harmony for the original melody, but it is possible that more deterministic perturbations combined with the already fuzzy matching process of our

system provides a significant level of redundancy to allow these errors to occur without destroying the system's correctness.

With all of these considerations in mind, it is clear that we have largely filled in the gap in the landscaping of automatic accompaniment systems which we highlighted in Section 2. We have designed and implemented a system which is capable of robustly handling jazz melodies, including improvisations.

Beyond these technical conclusions, it is important to remember the benefits that our system provides for the target user. Our techniques, although perhaps limited in scope, are far simpler than the more universal machine learning methods and are more directly modifiable by the user. For musicians with no computational training or technical background, it is much easier to enter a melody and harmony like they would on any other musical notation software (or even write it on that software directly and export it), add it to the library and thus customize the knowledge that the system has, tailoring its harmonic style to the preferences of the musician.

8.1 FUTURE WORK

In an immediate sense, the most significant improvement to the system as we have designed it would be to perfect the Listen stage. Beyond the context of the system as we have designed it, there are a number of modifications which we did not implement but would nonetheless be beneficial:

- Closing a feedback loop between the Listen and Play stages, so that if the Generate stage makes assumptions about the melody which turn out to be incorrect, the system can immediately change its harmony in response.
- Adding a input filter to the Listen stage which accounts for the harmony played by the Generate stage, so that the chords played by the system can be ignored when listening to the monophic melody of the musician and so that the system can play out loud rather than through a set of headphones.
- Adding a melodic memory to the Generate stage, so that the entire history of the song as it is played can be used to determine the harmony as opposed to just the most recently heard fragment.

Completely beyond the scope of our project, it is easy to imagine other extensions that would serve the same goals and help build a better accompaniment and practice tool. Given a harmony which the system has generated, it is not hard to produce a suitable walking bass line. Given the information produced by the beat tracking functionality of the Play stage, it is not hard to produce a drum track. Even if both

of these were very deterministic, they could still be useful options for fleshing out the soundscape of the accompaniment and come closer to emulating the feeling of playing with a band. The system could even interact with itself, with the different simulated instruments taking solos, giving the player the opportunity to play a harmony. The task of generating solos is equally, if not more, complex than that of generating harmonies, but it is nonetheless an interesting question for future work.

BIBLIOGRAPHY

[1] Juan Pablo Bello, Laurent Daudet, Samer Abdallah, Chris Duxbury, Mike Davies, and Mark B Sandler. "A tutorial on onset detection in music signals." In: *IEEE Transactions on speech and audio processing* 13.5 (2005), pp. 1035–1047.

[2] Paul Brossier, Juan Pablo Bello, and Mark D Plumbley. "Fast labelling of notes in music signals." In: *ISMIR*. Citeseer. 2004.

[3] Joanna Bryson. "The reactive accompanist: Adaptation and behavior decomposition in a music system." In: *The biology and technology of intelligent autonomous agents*. Springer, 1995, pp. 365–376.

[4] Wei Chai and Barry Vercoe. "Melody retrieval on the web." In: *Multimedia Computing and Networking 2002*. Vol. 4673. International Society for Optics and Photonics. 2001, pp. 226–241.

[5] Simon Dixon and Emilios Cambouropoulos. "Beat tracking with musical knowledge." In: *ECAI*. 2000, pp. 626–630.

[6] Fadi Al-Ghawanmeh. "Automatic Accompaniment to Arab Vocal Improvisation "Mawwāl"." PhD thesis. New York University, 2012.

[7] Michael Good et al. "MusicXML: An internet-friendly format for sheet music." In: *Xml conference and expo*. Citeseer. 2001, pp. 03–04.

[8] Masataka Goto and Yoichi Muraoka. "Music understanding at the beat level: Real-time beat tracking for audio signals." In: *Computational auditory scene analysis* (1998), pp. 157–176.

[9] William B Kuhn. "A real-time pitch recognition algorithm for music applications." In: *Computer Music Journal* 14.3 (1990), pp. 60–71.

[10] H Leonard. *The Real Book, Volume I, II, III, IV, V and VI*. 2012.

[11] Daniel Martín. "Automatic accompaniment for improvised music." PhD thesis. Master Thesis, 2009.

[12] Roger B Dannenberg Bernard Mont-Reynaud and Roger Dannenberg. "Following an Improvisation in Real Time'." In: *Proceedings of the ICMC*. Vol. 57. 1987, pp. 241–248.

[13] Robert A Moog. "Midi: Musical instrument digital interface." In: *Journal of the Audio Engineering Society* 34.5 (1986), pp. 394–404.

[14] Travis E Oliphant. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA, 2006.

[15] Christopher Raphael. "Music plus one and machine learning." In: *ICML*. 2010.

[16] Ian Simon, Dan Morris, and Sumit Basu. "MySong: automatic accompaniment generation for vocal melodies." In: *Proceedings of the SIGCHI conference on human factors in computing systems*. 2008, pp. 725–734.

[17] Michael Staudacher, Viktor Steixner, Andreas Griessner, and Clemens Zierhofer. "Fast fundamental frequency determination via adaptive autocorrelation." In: *EURASIP Journal on Audio, Speech, and Music Processing* 2016.1 (2016), p. 17.

[18] Iman SH Suyoto and Alexandra L Uitdenbogerd. "Simple efficient n-gram indexing for effective melody retrieval." In: *Proceedings of the Annual Music Information Retrieval Evaluation exchange* (2005).

[19] Alexandra Uitdenbogerd and Justin Zobel. "Melodic matching techniques for large music databases." In: *Proceedings of the seventh ACM international conference on Multimedia (Part 1)*. 1999, pp. 57–66.

[20] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python." In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[21] Cheng Yang. *Music database retrieval based on spectral similarity*. Tech. rep. Stanford, 2001.