# Accomplice: Computer Accompaniment for Keyboard Performance

Roger B. Dannenberg
rbd@cs.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

**Figure 1. Image from a performance of Felicitá, a chamber opera using Accomplice accompaniment software following a live pianist (at right), supported by live percussion and vielle (at left), with singers (left), dancers, and projected video.**

## Abstract

*Accomplice* is a computer accompaniment system created for contemporary electronic music performance with a human keyboard performer. The motivation for computer accompaniment has always been to support *expressive* music performance by following a human performer. Accomplice introduces an interface, behavior, and output modes designed for flexibility and interactivity in live performance of experimental music, as opposed to a classical model of start-to-finish score following. Accomplice takes MIDI information from the performer, follows a MIDI score, and outputs MIDI, O2 and OSC control that is synchronized to the live performance and controls custom synthesizer, interactive software, robotic, or multimedia processes. To support this functionality, Accomplice runs on a laptop with a graphical interface to specify various modes of starting and cueing musical sections, enabling and adjusting accompaniment tracks, and controlling the responsiveness of the accompaniment. Accomplice is free and open-source software.

## Keywords

Computer Accompaniment, Live Performance, Syncrhonization, MIDI, OSC, O2

## 1 Introduction

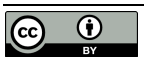
Computer accompaniment emerged with the work of Roger Dannenberg and Barry Vercoe in 1984 [1]. Computer accompaniment seemed to be an important solution to the problem of coordinating live performers and computers, especially at that time when a significant amount of computer music was restricted to fixed media. One motivation for computer accompaniment was to free performers from the fixed timing of tape music, allowing for more expressive performance and interpretation. This was also a time when real-time digital signal processing was just beginning to become a reality. (E.g., the Yamaha DX7 was released in 1983, followed by the rack-mounted multi-timbral TX816 in 1984.)

Although there was early interest by companies such as Yamaha and Sony, performances at IRCAM, and education products such as MakeMusic and Cadenza, computer accompaniment never came into common use by composers. It may be that MIDI sounds, when played conventionally, were less interesting than the results obtained in non-real-time computer music languages and later digital audio workstations. When it became possible to process audio in real time with software, composers quickly began composing "interactive" pieces that processed audio from acoustic performers, offering a variety of new sound possibilities and providing an interesting alternative way to coordinate electronic sounds with live performers.

Now, 40 years since the introduction of computer accompaniment, we see a resurgence of interest, at least in the research community, in computer accompaniment systems. Much of this work is focused on classical music and audio input. These provide research challenges, but modern composers might prioritize flexibility, open software and compatibility with familiar tools.

With this in mind, *Accomplice* is an experimental system that aims to bring together reliable score-following algorithms that extend early work with a modular design, flexible user interface, and ability to work with MIDI, OSC and O2, allowing integration with DAWs, Max, Pd, Live, and other software.

In the remainder of this paper, I will describe related work (Section 2), the problems that Accomplice intends to solve (Section 3), the software architecture (Section 4), some implementation details (Sections 5 through 8), field-testing Accomplice in an opera performance (Section 9), evaluation (Section 10) and conclusions (Section 11).




*NIME '26, June 23–26, 2026, London, UK*

## 2 Related Work

Early work by Dannenberg [2] [3], Vercoe [4], and Baird, Blevins and Zahler [5] created accompaniment systems with symbolic or discrete-event-based input. Some systems accompanied acoustic instruments by detecting pitches as events and using symbolic score following approaches. Later, attention turned to treating signals more continuously and directly using machine learning and probabilistic techniques [6][7] which seem to be a better way to handle uncertainties in acoustic input. The reader is referred to Lee for an extensive review of score following research [8]. More recent work includes learning to anticipate expressive tempo change [9], incorporation of expression into accompaniment parts [10], and a new reinforcement-learning approach to polyphonic score following [11].

## 3 Design Objectives

The basic operation of a computer accompaniment system is as follows: A live performer plays a composition, a "follower" compares the performance to a score, and the follower continuously estimates the score location. The score location and perhaps confidence information is used to update a "conductor" that dictates a current location and tempo for the accompaniment, and one or more "players" send control information from accompaniment scores to synthesizers or other processes. The final output is typically a musical accompaniment, but could also be control for signal processing, a synchronized video, a robotic dancer, lights, or some other time-based medium.

It is conventional to refer to the live performer as soloist and the computer output as accompaniment, but it is more correct to view this as collaborative performance. Indeed, there are possibilities for the computer to take a leading role where the "soloist" is following and even accompanying the computer. Nevertheless, we will keep to the soloist/accompanist terminology for simplicity.

The input to Accomplice is MIDI, normally from a keyboard. This could be an electronic keyboard or an acoustic piano with MIDI output such as a Yamaha Disklavier. It would certainly be useful to accept audio input from acoustic pianos, but audio raises many technical problems of recognizing notes in polyphonic performances, dealing with highly variable reverberation, feedback and interference from accompaniment sounds, and variability in microphone placement. MIDI offers a highly reliable and low-latency means of sensing input, leading to more dependable and consistent performance overall.

The output from Accomplice includes MIDI, allowing it to control any MIDI synthesizer or DAW. Output is not necessarily conventional notes. MIDI can be used to enable, disable, and control various signal processors and plugins, possibly processing the live keyboard performance or other live acoustic sounds. For further flexibility, Accomplice can output Open Sound Control (OSC) [12] or O2 messages [13]. A simple text representation can be used to express scores consisting of time-stamped messages with parameter values. These allow synchronized control of any number of parameters in Max, Pd or other software compatible with OSC or O2.

In many pieces, synchronization is complicated by the introduction of pauses, cadenzas, and other departures from continuous and more-or-less steady tempo. To support this, Accomplice uses "cues," which specify starting and stopping as well as changes in how the score is followed and how tempo is determined. Section 6 describes this in more detail.

MIDI output can also be modified through a user interface. In principle, one could make accompaniment changes in an editor and re-import the MIDI data to Accomplice, but it can be very helpful to "fine tune" performances with real-time adjustments. Therefore, the Player offers controls for overall adjustments in loudness, timing, and articulation.

## 4 Implementation

Accomplice is implemented in Serpent [14], a Python-like language for music programming. The implementation consists of two main modules: the Conductor implements score following and tempo and position estimation. The Player plays scores according to timing information from the Conductor (see Figure 2). This separation allows for multiple Players if there is a need for many MIDI channels or if sound synthesis is performed on multiple computers. More importantly, future Players might offer audio playback using time-stretching for synchronization, video playback using adjustable frame rates, or other media control such as lighting cues. We can also envision using multiple conductors to track ensembles or alternate conductor implementations to follow audio signals, all of which benefit from this modular design.

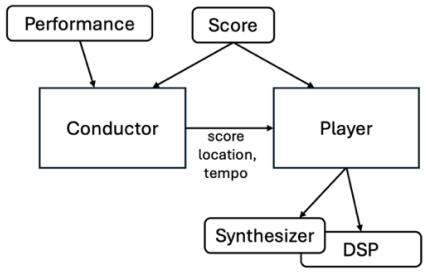



**Figure 2. Accomplice system, input, and output with separate Conductor and Player(s) processes.**

O2 provides important support for this distributed design: O2 implements clock synchronization so that processes (Conductor and Player) on separate computers can schedule according to a shared global time reference, and O2 performs "discovery," setting up network connections automatically so that users do not have to configure systems with IP addresses or port numbers.

### 4.1 Representing Time

Accomplice assumes that, to first approximation, scores are played "as written" with small variations in tempo. "As written" means that tempo changes in the score are observed. This is not the same as simply controlling beats-per-minute: If the score indicates a suddenly faster tempo, the Player should anticipate a tempo change and speed up immediately, even if the new soloist tempo is not observed until later. If the soloist changes tempo to a greater or lesser degree, at least the accompaniment will be more nearly synchronized by following the indicated tempo change.

One common way to implement this is to "flatten" the score from a beat-plus-tempo representation to a simpler one where events are simply labeled with time in seconds. Then, synchronization amounts to playing the score faster or slower by varying score-seconds per real-second. This approach has the advantage of "compiling in" tempo changes. A faster tempo will simply have more beats in the same interval of time. The disadvantage of this representation, however, is that it can be difficult for musicians to work in terms of seconds if the original score is written in terms of beats.

In Accomplice, we use a more elaborate approach that preserves the ability to anticipate tempo changes as if the score was flattened while also allowing scores to be accessed by beat

position. We do this by mapping real time to score position in several steps (see Figure 3). First, the local real-time is mapped by the Virtual Time Scheduler to a global time that is synchronized across all processes (this is necessary because processes may run on separate computers with separate clocks). Then, the conductor's score-follower provides a mapping $s = S(t)$ from real time t to score time s in the Flat Time Scheduler. Then, we use the MIDI file's tempo map to compute $b = B(s)$, mapping from flattened score time s to beat b in the Beat Time Scheduler. The score itself is a list of events with timing specified in beats, so they are scheduled by the Beat Time Scheduler.

Note that each scheduler implements an offset and scale factor to provide variable speed control in terms of a reference time provided by a "parent" scheduler below it in Figure 3. At any given moment, both $S(t)$ and $B$(s) are simple linear mappings, and all event sequences are processed incrementally and in order, so computation is efficient.

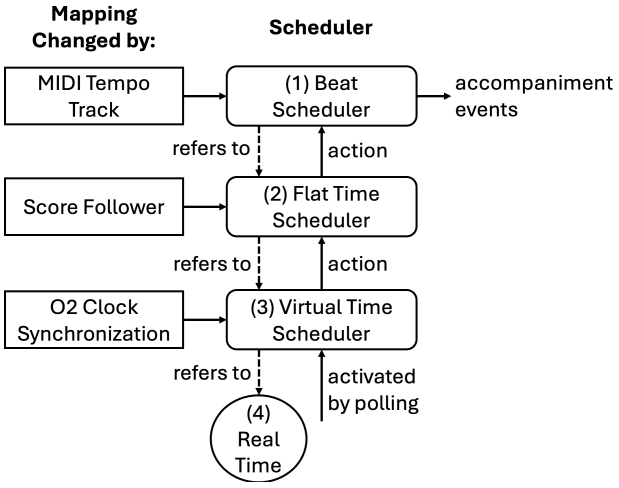



**Figure 3. Hierarchy of schedulers is used to maintain multiple representations of time. From top to bottom: (1) Beat position, which is affected by tempo changes, (2) Flat time, which is measured in "score seconds" independent of tempo, but which can be performed faster or slower than the nominal score time, (3) Global seconds, which maps local time to be consistent with a global time reference, (4) Real time is the local computer time.**

## 4.2 Dealing with Latency

Computer accompaniment uses the tempo and score to anticipate near future events, and this can be used to negate the inherent processing latency. E.g., if the total latency from key press to audio output is 10 ms, one can simply subtract 10 ms from the predicted time of the next event as if playing everything 10 ms early. This can compensate for the total latency of the Accomplice software, the keyboard, MIDI in, MIDI out, and audio synthesis. To reduce jitter, Accomplice attaches precisely computed timestamps to MIDI data. The MIDI device driver uses timestamps to schedule output more precisely than we can achieve within the Player process, which experiences some computational delay when it updates its graphical user interface. Note that the scheduler computes the ideal time to dispatch each event, so this time with an offset can be used directly as a MIDI timestamp.

Another detail is that the Conductor, which computes the ideal mapping $S(t)$ from real time to score time, runs asynchronously with respect to the Player. Thus, there will always be a very slight discontinuity in $S(t)$ when updates are received from the Conductor. The updates represent the desired mapping from real time to score time, i.e. $S(t) = s_0 + r\,(t - t_0)$. This says that at time $t_0$, the score position was $s_0$ and the performance is advancing at $r$ score seconds per real second. When an update is received (new values of ($t_0$, $s_0$, $r$)), the Player adjusts its rate $r$ to smoothly converge to the desired mapping in a short time. We call this the `adjust_map` operation. Typically, the latency from Conductor to Player is on the order of milliseconds, the convergence time is on the order of 100 ms, and the discrepancy between Conductor and Player is not noticeable.

A much more noticeable tempo adjustment occurs when the soloist deviates from the score tempo or plays expressively in ways that are not represented in the score. In these cases, the soloist can play a note earlier or later than anticipated, and the Conductor must compute a new $S(t)$ and transmit that to Player(s).

# 5 Score Following

Robust score following is aided by MIDI, which provides accurate and discrete note information as opposed to working with a complex continuous audio signal. Accomplice improves on earlier dynamic programming algorithms [3] that compute the best match between some prefix of the score and the performance seen so far. Because the algorithm must return results before observing the entire performance, there is no obvious perfect solution. For example, suppose a newly performed note matches a future note in the score after skipping a note or two. This might be a correct match, but it might also turn out that the note is spurious, and the "missing notes" are performed next, indicating that skipping ahead to the match was incorrect. As in previous work, Accomplice uses a heuristic based on a total "rating," where matched notes increase the rating and unmatched notes decrease the rating. The goal is to find the alignment of performance to score that maximizes the rating. When a new note leads to a total rating that exceeds any previous one, a "match" is reported with the score location.

An interesting problem with keyboard performance following is that chord notes can be played in any order, thus there is no simple ordered list of events to match. Following previous work [3], Accomplice has two methods for dealing with chords: the static algorithm and the dynamic algorithm. Here, we summarize our preferred dynamic algorithm and describe new improvements implemented in Accomplice.

The dynamic algorithm attempts to match the sequence of incoming performance notes to the sequence of score chords. Notes can be matched to chords in any order, solving the "in any order" problem, and by keeping track of matched notes so far, we can penalize the matching of repeated notes to the same chord. This approach works well in practice and is flexible with respect to performance timing, including rolled chords, which can extend over relatively long time intervals. Accomplice introduces two improvements over the original dynamic algorithm. First, we limit the time span of performed notes that can be associated with the same chord. Second, we add a small penalty for performed notes that do not match a note in the score. Both of these changes help to avoid unusual cases where a combination of performance errors and patterns in the score lead to surprising mismatches.

Figure 4 shows an abstract view of the data used in this approach. A matrix is computed where each row represents a compound event (chord) from the score, and each column represents one performance event (note). Each cell records the rating of the best alignment of the score up to the current row with the performance up to the current column. The "trick" that makes dynamic programming efficient is that a matrix element $z$ depends only on neighbors $x$ and $y$ (the transition from $w$ to $z$ is subsumed by $w \rightarrow y \rightarrow z$). The algorithm to update $z$ given

performed note *b* is shown in Figure 5. The `rating` field is an integer, `used` is an array of integer pitch values, and `time` is a timestamp; The expression `[b]` constructs an array, and `+` is used to concatenate two arrays as well as to add integers. The constant `SC=2` is the cost for skipping a note in the score, `MC=2` is the credit for matching a note in the score, and `EC=1` is the cost of an extra note that does not match the score.

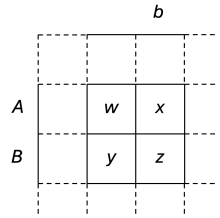



**Figure 4. Dynamic programming approach to score following using the dynamic grouping approach computes a matrix incrementally. Rows represent compound events (chords) in the score, and columns represent performed events (notes).**

The algorithm (Figure 5) is applied iteratively to compute a column from beginning to end as each performance note arrives. Notice the condition `z.time - y.time < 0.1`, which is a new constraint that restricts the time-span of notes matched to chords. This is somewhat simplified: In the actual algorithm, the time contraint is relaxed in the case of rolled chords.

```
// penalty for unused notes when moving to next chord:
z.rating = x. rating - SC * (len(A) - len(x.used))
z.used = []
z.time = get_time() // performed note time
// consider grouping new note with current chord:
if (b in B) and (b not in y.used) and
   (z.time - y.time < 0.1): // a match
    d = y. rating + MC // MC = match credit
    if d >= z. rating:
        z. rating = d
        z.used = y.used + [b]
     // time field is already set above
else: // group with chord, but no match
  d = z.rating - EC // EC = "extra" cost
    if d >= z. rating:
        z. rating = d
        z.used = y.used
        z.time = y.time // maintain y's time
```

**Figure 5. The dynamic grouping algorithm to update matrix cell z given input b (as shown in Figure 4).**

In practice, rather than computing the entire matrix column, only the neighborhood of the current score location is considered (currently, 60 score events), and only two columns are needed, so rather than store a large matrix, we only store about 120 matrix cells.

### 5.1 Limitations

A current limitation is handling trills where the number of notes is not fixed in the score. Previous work addressed this problem successfully, but it requires additional labels in the score to locate the trills [15]. Yet another dynamic programming approach has been developed that finds the optimal partitioning of input notes, where each partition matches one chord, possibly containing optional grace notes or repeated trill notes. We expect to incorporate this new algorithm into Accomplice.

One remaining problem is the fact that in some music, the left and right hand play so independently that the left hand gets ahead of the right hand or vice versa. This is especially true when playing polyrhythms such as 11 against 4, or with many notes played expressively against a slow and regular sequence of chords. We are experimenting with additional transition rules in the dynamic programming framework and also with methods to match the left and right hands separately [16].

### 5.2 Hidden Markov Model Approaches

Other researchers have used Hidden Markov Models for score following [17], guided by the success of HMMs in other areas and HMMs' strong probabilistic foundation. Although both approaches are related by their use of dynamic programming (Viterbi decoding in the HMM context), recent work indicates that our partition-based algorithm is more robust [18].

## 6 Cues

In practice, music is not normally so simple that a soloist can simply play a piece from beginning to end with score following. Different musical situations require some planning and direction for a successful performance. Accomplice includes a user interface where the performer can edit a sequence of annotations called cues to direct the human-computer interaction. The cues all have start and end times, and their behaviors are described below:

**Start** launches a performance at a given score location and tempo. The cue can be given by a pedal, mouse click, or OSC message.

**Metro** is like Start but it first plays 8 taps in tempo to cue the soloist.

**Match** is also like Start except it waits until the solo performance contains a note that occurs at the start time in the solo score.

**Foot3** is intended to launch a performance in a hands-free way using a foot pedal. Three foot taps are given to establish a tempo and the cue begins on the 4th tap.

**Wait** means the accompanist should stop on a rest or held note until the soloist reaches the end of the cue. This is useful for sections where the tempo is too variable, expressive or unpredictable for Accomplice to anticipate timing such as in a cadenza.

**Tempo** means that the accompaniment should play at a fixed speed, ignoring the performer.

**Hold** is similar to Tempo except the tempo upon entering the cue is held throughout the cue.

These cues support a variety of practical situations that occur in music performance and extend the ability of Accomplice to handle realistic performance requirements.

## 7 The Player Module

To facilitate fine-tuning in rehearsals or even during a performance, the Accomplice MIDI Player offers a set of adjustments for every track or channel, allowing notes to be:

- Shifted in time (earlier or later) to compensate for differences in perceptual onset times vs. MIDI note-on times,
- Stretched by a factor to vary articulation from more staccato to more legato,
- Made louder or softer by adding an offset to MIDI velocity,
- Tracks or channels can be muted or soloed,
- Tracks or channels can be rerouted to a different MIDI Program or MIDI channel.

Figure 6 shows the Player interface where these parameters can be adjusted. The Player also displays a scrolling piano-roll view of the score, which helps to select score positions in rehearsals (to start at any selected location) and to verify correct operation and current score location.

## 8 Osc and O2 Support

Accomplice can output OSC and O2 in addition to MIDI. This can be used to connect to a variety of systems such as Max, Pd, Soundcool, or other computer music systems or custom programs.

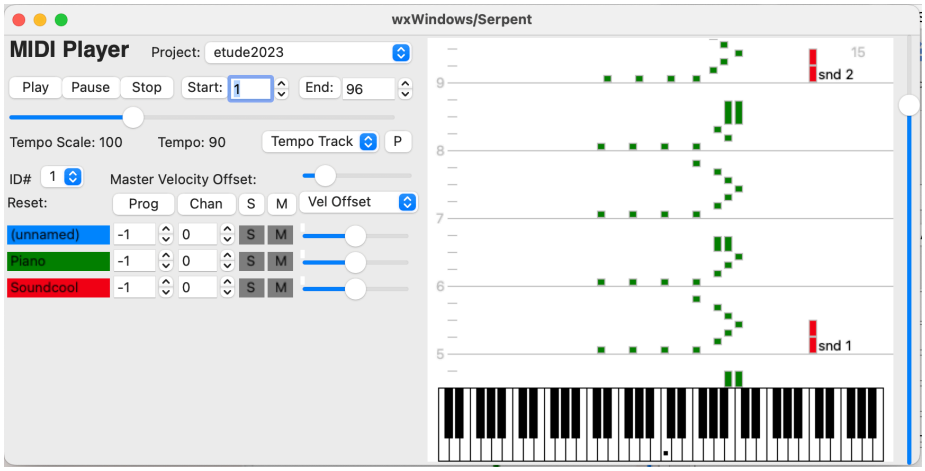



**Figure 6. The Player interface showing controls for real-time adjustments to the MIDI output (slider and number boxes at left) and annotation for OSC messages ("snd 1," "snd 2") at right (see Section 8).**

Scores for message events are written in Allegro, a text-based score language where each line represents a timed event and where different fields on the line specify attributes. For OSC and O2, the only attributes of interest are time, channel, and a string encoding the message. For example, this line sends a message on beat 8 (counting from 0):

```
TQ8 V16 -oscs:"/sampler/4/push1 1"
```

where TQ8 means "at the time of 8 quarter notes," V16 means "on voice (channel) 16" and -oscs: is the attribute used to denote an OSC message. The message consists of an address and parameters, which can be integers (digits), floats (with decimal point) or strings (in single quotes). The OSC address and port number are entered into the Player.

Alternatively, O2 provides discovery to connect automatically to other processes, and you can use O2 to message multiple processes. The O2 syntax is similar but uses `-o2s:` as the attribute. An O2 address begins with a "service" name to address the destination, so a message specification for service `sc1` might look like:

```
TQ8 V16 -o2s:"/sc1/sampler/4/push1 1"
```

Although OSC and O2 messages do not have channels, we use channel 16 to tag sound-generating messages and channel 17 to tag state-changing messages. If Accomplice (re)starts in the middle of the score, it sends all odd-channel OSC and O2 messages from the beginning and up to the start point in a "pre-roll" phase. (It also pre-rolls all MIDI program and control-change messages to establish the expected controller values in synthesizers.) Finally, the Accomplice Player will display a text string with the addition of some parameters. For example, by adding: `C6 Q -anns:"snd2"` to the examples above, the scrolling score display will show the text "snd2" at the position of a quarter-note C6 as shown above in Figure 6 (`-oscs:` or `-o2s:` suppress the sending of a MIDI C4). A demonstration of Accomplice with OSC output can be viewed online: `youtu.be/0IDza7jsM3U`.

## 9 Opera Performance

Accomplice has been tested and used for small pieces as well as a professional production of a full chamber opera, *Felicità*, which was previously performed with carefully designed cues and fixed media. In the past, we found it difficult to coordinate singers and electronics given limitations on full-scale rehearsals in the performance space. Even then, the fixed media segments had to be designed to be clearly heard or to not require much synchronization. This approach could not realize the music as intended.

For a performance with Accomplice, we used Logic Pro as a MIDI sequencer and synthesizer. This allowed us to compose, design and "orchestrate" the electronic parts, using MIDI sequencing to perform a mock-up version of the opera. The live performance used a human pianist as "soloist." (Following singers directly is conceivable, but much less robust.) The pianist follows a conductor and singers, and Accomplice synchronizes to the pianist.

### 9.1 Workflow

To design the "accompaniment," the piano part is exported from a score editor as MIDI and imported into Logic Pro. There are singers and live electronics as well, so we import audio recordings to fill out the performance and approximate the overall sound. The MIDI parts are sketched in a notation editor along with the piano part and exported as MIDI files for use in Logic Pro. There, we continued to edit, orchestrate and apply audio effects for the final sound.

The main restrictions for Accomplice are that we cannot use mixer automation in Logic Pro and must instead rely on MIDI velocity, volume, and other MIDI controls. This is because in live performance, we do not run the Logic Pro sequencer at all, so mixer automation will not be operational. Nevertheless, we were free to use the mixer, including panning, equalization, reverb and plug-ins to sculpt the sound.

When the design was complete, we exported MIDI from Logic Pro for use in Accomplice. For performance, we enable all Logic Pro tracks to play incoming MIDI. This gives us exactly the same notes, synthesizer settings and mix as our mock-up so that we get the sound we intended, and further adjustments are possible in rehearsals.

## 10 Testing and Evaluation

Evaluation difficulties have plagued the study of accompaniment from the beginning. As real-time systems, input and output are time-dependent, so one cannot easily run algorithms over a dataset in batch mode and collect statistical results. There is a wide range of performance accuracy, contexts, and objectives. Datasets are few, and while Lee offers an audio performance dataset and evaluates some algorithms [8], only the more recent ASAP dataset contains aligned symbolic scores and performances [19]. Accomplice does not currently support grace notes and trills, which appear often in ASAP scores, because ornaments in general allow too much variation from one performance to the next. However, an extension to Accomplice's score follower handles ornaments well, and in our study of 133 piano performances (under review), reduces total prediction error (especially large errors and outright failures) by more than half, compared to the best (HMM-based) comparison system, with even larger improvement over systems based on DTW and deep-learning.

Ultimately, the quality of score following is largely dependent upon human deviations from the score, including both notes and timing, so evaluation results are highly dependent upon the input, and there is no standard for error rates or expressive timing deviations, particularly for the experimental music for which Accomplice is created. We note that our opera performance provides a successful real-world test of the score following as well as the interface design, use of cues, integration with sound synthesis software, and more broadly, the practicality of computer accompaniment in a complex chamber music performance.

### 10.1 Objective Measurements

With that in mind, we offer some data from our opera performance. There is no "ground truth" for a correct performance, but we can compute the predicted time of each performed note and compare that to the actual time. Figure 7 is a histogram of differences between predicted and actual match times for all notes that were matched. Prediction is based on linear regression, ideally using about 4 seconds of recent note onset times. The center bar represents a range from -50 to +50 ms, which is quite good. Larger errors have multiple causes, including (1) deviations from the score, such as when a singer stretches a note to take a breath, (2) wrong notes in the piano performance increase the time between matches and decrease the information available for synchronization, and (3) the timing of arpeggios can stretch out over several tenths of seconds, deviating from expected times in the score.

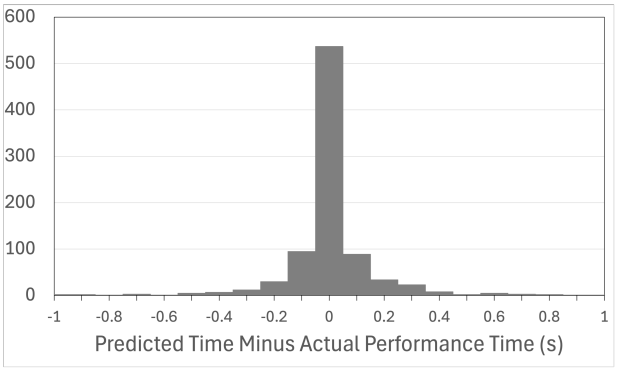



**Figure 7. Histogram of prediction errors for performed note times. Positive means the predicted time was later than the actual time.**

There were some outliers even beyond the range of the histogram. The largest "error" was 3.1 s, but upon examination, this was after a 9 s pause in the score, specifically intended as a fermata. Since the actual pause is shorter, the piano always enters "early" according to the score. Several piano notes are played to establish the time and tempo before accompaniment resumes, allowing Accomplice to catch up, so this is all designed to *avoid an actual performance error.* In general, when the actual performance is early relative to predictions, Accomplice can quickly catch up to avoid the impression of timing errors. The worst *negative* error (-1.6s) was due to a singer who held a note longer than written, essentially inserting a fermata where none was intended. There is very little one can do about this problem unless it is noticed in rehearsal where one can modify the score to reflect the expected performance. We solved several problems uncovered in rehearsal in this way. Overall, there were 7 values beyond the range of the histogram in Figure 7, but, as with new music performed by humans, these did not lead to any obviously serious mistakes.

### 10.2 Qualitative Findings

While statistics on timing errors and alignment are part of the picture, overall musicality is much more difficult to measure. For example, smooth and rhythmic adjustments to the accompaniment may not prioritize alignment but may sound more musical and appropriate overall. Thus, even synchronization accuracy is not the ultimate goal.

In rehearsals and performance, Accomplice followed the pianist without any manual intervention, despite many wrong notes and deviations in timing. Despite our efforts to make Accomplice a professional collaborative performer, our pianist found it amusing and referred to it as her "baby" since it followed her so quickly and obediently, never asserting a strong musical independence. This is of course by design, but it raises interesting questions about ideal human-computer collaboration and how we view AI collaborators. The performance can be viewed online: `youtu.be/pygC11IcCb0`.

In retrospect, two features we found missing were: Navigation and display in Accomplice using measure numbers as they appear in the score (even though this would require additional manual data entry), and automatic chaining of cues, e.g., fermatas (pauses) could be handled by immediately entering a Match cue to wait for the next note after the held fermata. Doing this manually was successful, but a bit nerve-wracking when pauses were short.

## 11 Summary and Conclusions

Accomplice is an accompaniment system designed for composers of electronic works. It uses MIDI input from a keyboard, as this provides a reliable input for score following that is not subject to audio interference in real concert situations. Symbolic score following seems to be a simpler problem than following acoustic instruments, resulting in a robust and reliable system. Particular attention has been given to accompaniment as a kind of interface between performer and computer, with appropriate controls to achieve effective rehearsals and reliable performances. Accomplice is modular, separating the score following "conductor" from one or more "player" modules. Currently, the Accomplice MIDI Player can output MIDI from Standard MIDI Files and OSC or O2 messages specified with a simple text-based score language.

We have tested Accomplice on a variety of pieces including a full chamber opera performance. We have introduced an improvement over an earlier score-following algorithm: The dynamic grouping algorithm now imposes an upper limit on the separation of note onsets that can be grouped to match a chord. Also, it was discovered that a zero cost for extra (wrong) notes can lead to eager and incorrect matching, so a small penalty was added. Further improvements are planned to increase robustness in following expressive passages where the left and right hands play "out of order," and scores with trills, grace notes, and other indeterminate events.

Accomplice is open source and available from the author, who is especially interested in working with composers on new works for keyboard instruments and live synchronized electronics including both MIDI and creative signal processing.

## 12 Ethical Standards

This research focused on the development and technical evaluation of Accomplice, which has been improved through its use in projects involving human performers. Since the primary object of study is the software system itself, this work did not require a formal human subjects protocol or explicit approval from my Institutional Review Board. Nevertheless, all participants were informed about the nature of the rehearsals and performances, no personal data was collected

other than what was published in programs and posters advertising the public performance, and opera participants were compensated as professionals.

## Acknowledgments

Many people have worked on Accomplice. I would like to thank especially Shuqi Dai, Huiran Yu, and Gus Xia who contributed to the Accomplice implementation. Thanks also to Jorge Sastre, Nuria Lloret, Universitat Politècnica de València, and many other contributors to Felicitá and its performance.

## References

[1] William Buxton, Roger B. Dannenberg, and Barry Vercoe. 1986. The computer as accompanist. In *Human Factors in Computing Systems: CHI '86 Conference Proceedings*, Boston, MA. ACM Press, New York, NY, 41-43.

[2] Roger B. Dannenberg. 1985. An on-line algorithm for real-time accompaniment. In *Proceedings of the 1984 International Computer Music Conference*, Paris. International Computer Music Association, 193-198.

[3] Josh Bloch and Roger B. Danennberg. 1985. Real-time accompaniment of polyphonic keyboard performance. In *Proceedings of the 1985 International Computer Music Conference*, Vancouver, BC Canada. International Computer Music Association, 279-290.

[4] Barry Vercoe. 1985. The synthetic performer in the context of live performance. In *Proceedings of the 1984 International Computer Music Conference*, Paris. International Computer Music Association, 199-200.

[5] Bridget Baird, Donald Blevins, and Noel Zahler. 1993. Artificial intelligence and music: Implementing an interactive computer performer. *Computer Music Journal,* 17, 2, 73-79.

[6] Lorin Grubb and Roger B. Danennberg. 1997. A stochastic method of tracking avocal performer. In *Proceedings of the 1997 International Computer Music Conference*, Thessaloniki, Greece, September 25-30. International Computer Music Association, 301-308.

[7] Christopher Raphael. 1999. Automatic segmentation of acoustic musical signals using hidden markov models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21, 4, 360-370.

[8] Lin Hao Lee. 2022. Musical score following and audio alignment. MEng Final Year Project Report, Imperial College London. https://arxiv.org/abs/2205.03247.

[9] Gus Xia, Yun Wang, Roger B. Dannenberg, and Geoffrey Gordon. 2015. Spectral learning for expressive interactive ensemble performance. In *Proceedings of the 16th International Society for Music Information Retrieval Conference*, Malaga, Spain. 816-822.

[10] Carlos Cancino-Chacón, Silvan Peter, Patricia Hu, Emmanouil Karystinaios, Florian Henkel, Francesco Foscarin, and Gerhard Widmer. 2023. The accompanion: Combining reactivity, robustness, and musical expressivity in an automatic piano accompanist. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence AI and Arts.* International Joint Conferences on Artificial Intelligence Organization, 5779-5787.

[11] Silvan David Peter. 2023. Online symbolic music alignment with offline reinforcement learning. In *Proceedings of the 24th International Society for Music Information Retrieval Conference,* Milan. 634-641.

[12] Matt Wright, Adrian Freed, and Ali Momeni. 2003. Open sound control: State of the art 2003. In *Proceedings of the 3rd Conference on New Instruments for Musical Expression*, Montreal. 153-159.

[13] Roger B. Danennberg. 2022. Communication for real-time music systems: An overview of O2. In *Computer Music Journal*, 45, 4, 7-19.

[14] Roger B. Dannenberg. 2002. A language for interactive audio applications. In *Proceedings of the 2002 International Computer Music Conference.* International Computer Music Association, 509-15.

[15] Roger B. Danennberg and H. Mukaino. 1988. New techniques for enhanced quality of computer accompaniment. In *Proceedings of the International Computer Music Conference*, Cologne. Computer Music Association, 243-249.

[16] Ziyu Cai. 2025. Enhanced real-time accompaniment algorithms for out-of-order notes. Master's thesis. School of Music, Carnegie Mellon University.

[17] Eita Nakamura, Yasoyuki Saito, Nobutaka Ono, Shigeki Sagayama. 2014. Merged-output hidden markov model for score following of midi performance with ornaments, desynchronized voices, repeats and skips. In *Proceedings of the 11th Sound & Music Computing joint with the 40th International Computer Music Conference.* International Computer Music Association, 1185-1192.

[18] Roger B. Dannenberg, Ziyu Cai, Ellie Xue, Michael Chen. Yuyang Shan, Marco Mileti. 2026. Optimal partitioning for symbolic polyphonic score following. In review.

[19] Francesco Foscarin, Andrew McLeod, Philippe Rigaux, Florent Jacquemard, Masahiko Sakai. 2020. ASAP: A dataset of aligned scores and performances for piano transcription. In *Proceedings of the 21st International Society for Music Information Retrieval Conf.* 534-541.