FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

LayerIt: Towards a Time-Aligned Framework for Composable Music Visualization Layers

Fernando Azeredo



Mestrado em Engenharia Informática e Computação

Supervisor: Prof. António Sá Pinto

October 19, 2026

LayerIt: Towards a Time-Aligned Framework for Composable Music Visualization Layers

Fernando Azeredo

Mestrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

President: Prof. António Fernando Vasconcelos Cunha Castro Coelho
Examiner: Prof. Pedro José Mendes Martins
Member: Prof. António Humberto Sá Pinto

October 19, 2026

Abstract

Music information retrieval often requires integrating signal-derived, algorithmic, and symbolic information across different representations. Existing visualization tools typically keep notation separate from physical time, while composites that combine them are assembled manually. This thesis presents LayerIt, a Python-based object-oriented framework for composing independent visual representations into unified figures on a shared performance-time axis. Built on a modular layer-composition model, LayerIt treats audio plots, algorithmic outputs, symbolic scores, and other data sources as independent components that can be flexibly combined and regenerated. Figures are exported as SVG, preserving the score's MEI structure while keeping added components identifiable. The thesis asks how heterogeneous MIR visualizations can be combined in a common score-aligned representation, to what extent modular layers and scriptable composition support their generation, modification, and regeneration, and how such visualizations can support qualitative inspection of beat-tracking outputs and annotation data. LayerIt is validated qualitatively through a set of demonstrations built on real score and performance data, showing the composition of score, audio, and algorithmic representations on a common performance-time axis, while establishing that the usefulness of the resulting visualizations depends on the quality of the available score-performance alignment. More broadly, the project establishes a reusable, extensible foundation for composable score-aligned MIR visualizations.

Keywords: Music Information Retrieval • Visualization • Audio-to-Score Alignment • Beat Tracking • SVG • Object-Oriented • Python

Contents

1 Introduction **1**
- 1.1 Contextualization . . . 1
- 1.2 Motivation . . . 2
- 1.3 Problem, Research Questions, Objectives and Contributions . . . 3
- 1.4 Developed Project . . . 4
- 1.5 Thesis Structure . . . 4

2 Literature Review **5**
- 2.1 Visualization Types . . . 5
  - 2.1.1 Sheet Music and Score Images . . . 6
  - 2.1.2 Symbolic Representations . . . 6
  - 2.1.3 Audio Representations . . . 8
- 2.2 MIR Visualization Tools . . . 11
- 2.3 Audio-to-score alignment . . . 12
- 2.4 Summary . . . 14

3 Methodology **15**
- 3.1 Research Approach . . . 15
- 3.2 Iterative Development Process . . . 16
  - 3.2.1 BeatMapJS . . . 16
  - 3.2.2 MEI-Basic-Edit . . . 16
  - 3.2.3 BeatPrecision . . . 17
  - 3.2.4 Score-Alignment.py . . . 17
  - 3.2.5 Score-Alignment.js . . . 17
- 3.3 Scope Decision . . . 18
- 3.4 Validation Criteria . . . 19

4 LayerIt Design and Implementation **20**
- 4.1 The Problem and Our Solution . . . 20
- 4.2 LayerIt Overview . . . 20
- 4.3 Design Choices and Rationale . . . 21
- 4.4 Generating Visuals . . . 21
  - 4.4.1 Getting the Warped Score . . . 22
  - 4.4.2 Getting the MAPS file . . . 22
  - 4.4.3 Providing BT data . . . 25
- 4.5 Layer Alignment Method . . . 25
- 4.6 Layers on Layers, within Layers . . . 26
  - 4.6.1 Visual Level . . . 26

4.6.2 SVG Level . . . 27
4.6.3 Code Level . . . 28
4.7 Summary . . . 31

5 Validation **32**
5.1 Validation Approach . . . 33
5.2 Reproducing Composite Figure Types (RQ1) . . . 34
5.2.1 Score, audio, and chroma on a shared axis . . . 34
5.2.2 Score-aligned BT analysis . . . 35
5.2.3 Logits and position displays . . . 35
5.2.4 Layered time-based display . . . 36
5.3 Beat-tracking analysis of *Clair de Lune* . . . 38
5.4 Workflow Capabilities for Annotation and Evaluation (RQ2, RQ3) . . . 39
5.4.1 Evaluation workflow . . . 40
5.4.2 Annotation workflow . . . 40
5.5 Modular Composition Evidence (RQ2) . . . 41
5.6 Limitations and Difficult Cases (RQ2, RQ3) . . . 42
5.6.1 Expressive performances . . . 42
5.6.2 Sparse onset annotation . . . 43
5.6.3 Manual editing of the final SVG . . . 44
5.6.4 Rendering and representation fidelity . . . 45
5.7 Summary . . . 45

6 Conclusions and Future Work **46**
6.1 Achievements . . . 46
6.2 Future Work . . . 47
6.3 Final Remarks . . . 48

References **50**

Statement on the Use of Generative AI **53**

A Additional Code Listing **54**

List of Figures

2.1 Example of an audio waveform. 8
2.2 Example of Mel spectrogram. 9
2.3 Example of a CQT. 10
2.4 Example of a chromagram. 10

4.1 Workflow positioning of alignment technologies and LayerIt 24
4.2 Composition of the LayerIt visualization system. Each coloured box represents a distinct visual element that composes a higher-level visualization. 26
4.3 Architecture of LayerIt. External audio, beat-tracking, alignment, and score inputs feed the framework; ScoreWarp produces the warped score, while LayerIt loads audio and constructs layer instances. Matplotlib and the shape abstractions generate the individual layers, which `Visualizer.compose()` combines into a score-aligned SVG. 29

5.1 LayerIt composition for the opening six bars of Debussy's *Clair de Lune*, performed by Maria João Pires. From top to bottom: time-warped score, waveform, mel spectrogram, and chromagram. The waveform is shown in yellow, and cyan dotted lines mark score-aligned onsets across the time-based panels. 34
5.2 LayerIt score-aligned BT display for BWV 856, performed by András Schiff. From top to bottom: time-warped score, mel spectrogram, and waveform with BeatThis beat estimates shown as red vertical lines, and onsets from Piano Aligner shown as black dotted vertical lines. All panels share the performance-time axis. 35
5.3 LayerIt activation-and-position display for BWV 856. The upper panel shows BeatThis beat logits as a red curve, with beat estimates overlaid as vertical black dotted lines; the time-warped score occupies the middle panel; and the lower panel shows downbeat logits as a blue curve, with estimated downbeat positions overlaid as vertical black dotted lines. 36
5.4 LayerIt display of an excerpt from BWV 856 performed by András Schiff. The time-warped score is shown above the waveform, with BeatThis beat estimates overlaid as red vertical lines aligned with the score. 37
5.5 LayerIt composition for the first six bars of Debussy's *Clair de Lune*, performed by Maria João Pires. From top to bottom: the time-warped score; beat and downbeat logits with onsets shown as black dashed lines; the waveform in pink with beat and downbeat estimates; and a mel spectrogram. Beats are shown as dotted orange markers and downbeats as solid blue markers. 38
5.6 Demonstration of the new `OnsetNovelty` layer composed through LayerIt's existing workflow. The onset-strength curve is rendered as a time-based layer for the *Clair de Lune* example. 42

5.7 Comparison of sparse and dense score–performance alignment for an excerpt of BWV 865 performed by András Schiff. From top to bottom: score warped using sparse onsets, waveform with BeatThis beat estimates in red and retained sparse onsets as black dotted lines, score warped using all onsets, and the same waveform with the complete onset set. The sparse MAPS file retains the first and last onsets and every fiftieth onset between them. 44

List of Tables

3.1 Summary of development iterations 18

4.1 Required files for LayerIt 22
4.2 Required .npz fields 25

5.1 Composite figure types reproduced with LayerIt. 37
5.2 Workflow characteristics relevant to the score-aligned visualization task considered in this thesis. 41

List of Acronyms

BA Beat Annotation

BT Beat Tracking

GUI Graphical User Interface

HTML HyperText Markup Language

JS JavaScript

MIDI Musical Instrument Digital Interface

MIR Music Information Retrieval

OO Object-Oriented

OOP Object-Oriented Programming

PDF Portable Document Format

PNG Portable Network Graphics

SVG Scalable Vector Graphics

XML Extensible Markup Language

Chapter 1

Introduction

1.1 Contextualization

Music Information Retrieval (MIR) is a research field that develops computational methods to analyse, interpret, and retrieve musical information from performance recordings and symbolic representations. MIR bridges a wide range of disciplines, including musicology, signal processing, machine learning, and human-computer interaction. Because MIR is a data-intensive field, visualization plays a crucial role in its research. The way information is selected and represented shapes the conclusions we draw and guides further progress.

The challenge of visualization in MIR is particularly significant because music is an inherently abstract and subjective medium. Translating music into the visual domain is a problem that spans millennia; throughout history, multiple systems of musical notation have been developed in attempts to capture sound in written form, yet this remains an evolving challenge. For MIR purposes, this means that musical data comes in a variety of forms, including audio recordings, symbolic scores, and annotations of expressive qualities. Each of these requires a different visual approach. An audio visualization such as a waveform or spectrogram reveals how a performance sounds, while a musical score conveys how a piece should be performed; both serve distinct and important purposes for different research tasks.

1.2 Motivation

MIR researchers often need to inspect algorithmic outputs alongside audio representations and symbolic scores in a temporally coherent manner. This kind of multimodal, time-aligned visualization is central to both the evaluation of MIR algorithms and the development of annotation workflows. Audio-to-score alignment, the task of synchronizing a performance recording with its corresponding symbolic score, is a key enabler of this type of visualization, as it provides the temporal anchor needed to align heterogeneous data sources into a unified view.

This thesis was motivated by two closely related challenges in beat-tracking (BT) and beat-annotation (BA) research. Contemporary BT systems are predominantly based on deep learning. Consequently, the field relies heavily on tools that support annotation and evaluation tasks, both of which are essential for advancing BT research.

Annotation tasks, particularly beat annotation, remain laborious and time-consuming. Existing graphical user interface (GUI) tools such as Sonic Visualiser leave a clear gap in this area: integrating recent state-of-the-art BT algorithms is difficult, and BA-specific features have seen little meaningful innovation. This disconnect between annotation workflows and the algorithms they are intended to support is a concrete obstacle to progress in both domains.

GUI tools present a similar limitation for evaluation: they often lack methods for inspecting algorithmic behaviour in depth. Better tools for evaluating and comparing models with one another are essential for improving them.

Although solutions like Sonic Visualiser [3] are powerful and widely used, their development cannot keep pace with the needs of modern MIR research. Integrating new algorithms or producing specialized visualizations is often impractical within them. A particularly underserved case is the visualization of score-aligned data: combining audio and data representations with a musical score typically involves manual image manipulation, and no unified methodology is widely used in the community.

For these reasons, researchers frequently resort to *ad hoc* solutions that lack reproducibility and are difficult to share or extend. This fragmentation creates a significant obstacle to the collaboration on which MIR research depends. By contributing a flexible, extensible, and community-oriented visualization tool built on established libraries, this thesis aims not only to address an immediate practical gap but also to establish a foundation for a framework that can support the growing complexity and interdisciplinarity of MIR research.

1.3 Problem, Research Questions, Objectives and Contributions

Score-aligned MIR figures, which combine notation, audio-derived representations and algorithmic outputs on a common time axis, are typically produced through manual image editing, *ad hoc* scripting and tool-specific workarounds (Section 1.2). The resulting figures are difficult to reproduce, modify and regenerate when the underlying data change. This thesis addresses the following research questions:

- **RQ1**: How can heterogeneous MIR visualizations be combined into a common score-aligned representation?
- **RQ2**: To what extent do LayerIt's modular layers and scriptable composition support generating, modifying, and regenerating score-aligned visualizations, including the addition of new time-based layers?
- **RQ3**: How can score-aligned visualizations support qualitative inspection of beat-tracking outputs and annotation data on a common time axis in the demonstrated case studies, subject to the limitations of the available score-performance alignment?

To answer these questions, the thesis pursues three objectives:

- **Objective 1** (RQ1): Develop a reusable Python framework for composing score-aligned MIR visualizations based on an object-oriented (OO) layer-composition model (Chapter 4).
- **Objective 2** (RQ1, RQ2): Use the framework to generate and regenerate composite figure types combining symbolic, audio-derived and algorithmic layers, and to add a new time-based layer through the existing interface (Sections 5.2 and 5.5).
- **Objective 3** (RQ3): Apply the resulting visualizations to beat-tracking case studies and document the limits imposed by score–performance alignment and rendering (Sections 5.3–5.6).

The main contributions of this thesis are:

- **Software contribution:** LayerIt, an open-source, pip-installable Python framework for composing score-aligned MIR visualizations through an OO layer-composition model. Its official repository, published with the LBD, is `https://github.com/FernandoAze/LayerIt`; the visual examples presented in this thesis were generated using the development repository at `https://github.com/FernandoAze/LayerIt_WIP`.
- **Research contribution:** a practical approach to visualizing representations expressed in different time domains on a shared performance-time axis, by composing score, audio-derived and algorithmic layers on top of an existing score–performance alignment.

The work also resulted in a Late-Breaking Demo paper accepted for presentation at ISMIR 2026[1]. Neither contribution includes the alignment method itself, which is provided by ScoreWarp [11].

[1] Parts of Chapters 4 and 5 are included in the accepted Late-Breaking Demo paper [1].

1.4 Developed Project

To address these objectives and research questions, we developed LayerIt, a Python-based object-oriented framework for composing score-aligned MIR visualizations. LayerIt integrates established MIR libraries and tools into a Scalable Vector Graphics (SVG)-based workflow in which visual elements are treated as independent layers that can be composed, reordered, and combined into custom figures.

At its core, LayerIt supports the integration of symbolic scores with audio and algorithmic data in a single view. By relying on ScoreWarp for temporal alignment, it makes it possible to overlay spectrograms, onset information, and beat-tracking outputs on score representations within the same visualization. The result is a flexible layer-composition model that supports the visualization workflows developed in this thesis and provides a foundation for future MIR visualization workflows.

1.5 Thesis Structure

The remainder of this thesis is organized as follows:

- **Chapter 2** provides a literature review covering the main audio visualization types, existing GUI and code-based visualization tools, and audio-to-score alignment.
- **Chapter 3** presents the adopted methodology and describes the five iterative phases that culminated in the development of LayerIt.
- **Chapter 4** describes the problem addressed by LayerIt, its design and implementation, and the "layers within layers" philosophy that underpins its architecture and use.
- **Chapter 5** validates the framework against the research questions through the generated figures, a qualitative workflow comparison, and a discussion of its limitations.
- **Chapter 6** reflects on the achievements of the thesis and outlines directions for future work.

Chapter 2

Literature Review

In this chapter, we present the background required for this thesis. We begin by exploring the landscape of music visualization methods, categorizing them into three main types: sheet music representations, symbolic machine-readable formats, and audio-based visualizations. This foundation is essential for understanding how different types of musical information can be represented visually and interacted with computationally. We then examine contemporary MIR visualization tools, distinguishing between GUI-based solutions and code-based workflows to highlight the gap that motivates our work. Finally, we introduce audio-to-score alignment and explore relevant tools and approaches that have informed our solution.

2.1 Visualization Types

We can sum up all music related visualizations into three categories:

1. Sheet Music and Score Images
2. Symbolic Representations
3. Audio Representations

These categories provide a task-oriented overview of the representations most relevant to this thesis; they are not intended as a comprehensive taxonomy of music visualization. Lima et al. survey 51 music-visualization proposals, comparing their inputs, visualized features, techniques, goals, interaction, and user evaluations [20]. Their classification of the elements visualized comprises pitch, timbre, mood, structure, harmony and melody; time is not among these categories, and the authors list rhythmic features such as tempo changes among the aspects not yet treated extensively [20]. Khulusi et al. take a broader musicological perspective, classifying 129 works by data type, visualization technique, and task, and identify uncertainty and inconsistency in musical data as recurring challenges [18]. They treat musical scores and musical sound as separate data subcategories, with score data playing a secondary role in sound visualizations, and their music alignment task concerns similar patterns between or within pieces rather than the correspondence

between a score and its performance. They point instead to adapting sequence alignment algorithms and visualizations to compare interpretations of the same score as a benefit still to be drawn [18]. Neither survey therefore addresses the task considered here: placing a score on the physical time axis of a performance and composing it with audio-derived and algorithmic representations. Together, these surveys situate this thesis within the wider field while motivating its narrower focus on composing score-aligned MIR representations.

2.1.1 Sheet Music and Score Images

Sheet music describes musical works using a formal language based on musical symbols and letters. Thus, this kind of musical representation always requires a level of musical literacy from the performer. Furthermore, sheet music usually does not convey explicit information about how a given piece should be performed; it is intended to allow the musician a certain degree of interpretative freedom. This can include variations in tempo, articulation, and dynamics, among others.

Sheet music also comes in a variety of forms that vary throughout history and across cultures. For our purposes, we focus on the Western standard of musical notation. Still, within Western music, there can be vastly different forms of notation; however, most are based on a staff consisting of five horizontal lines and four spaces, each representing a different musical pitch. Musically relevant symbols are placed on the staff to denote pitch, temporal, and expressive indications [22].

2.1.2 Symbolic Representations

Symbolic Representations are machine-readable formats where musical events and expressions are explicitly encoded. These come in various formats, such as MIDI and MusicXML.

Visually, two main applications of symbolic representations are piano-roll representations and digital score representations. A **piano roll** is a visual and interactive representation of musical events, commonly encoded as MIDI notes with attributes such as velocity and other expressive controls. Digital scores encode musical notation and structure in a form that can be processed by software.

MusicXML is a format based on the Extensible Markup Language (XML) for transcribing and exchanging digital scores. The format is widely supported by music-notation applications, but this thesis uses the Music Encoding Initiative (MEI) format because its structure and identifiers are particularly useful for linking score elements to other musical information. MEI stands both for the score-encoding format and for the community that develops it. The MEI website describes it as follows:

"The Music Encoding Initiative (MEI) is a community-driven, open-source effort to define a system for encoding musical documents in a machine-readable structure. MEI brings together specialists from various music research communities, (...) to define best practices for representing a broad range of musical documents and structures." [24]

MEI focuses on a "machine-readable structure". This structure relies on unique identifiers (@xml:id) that make every element in the score individually addressable, from individual noteheads to complex spanning elements such as slurs and ties. MEI can also store notation, performance, and interpretation data linked to individual score elements. These properties make it useful for MIR applications that need to relate score structure to other musical information.

MEI is a purely data-driven project; to be visualized or interacted with, it must therefore be decoded into a "human-readable format". Verovio provides this functionality. The MEI and Verovio projects are closely interlinked for that reason. While MEI defines how musical information is stored, Verovio is the primary tool used to transform that data into interactive visualizations and performance-ready outputs.

Verovio is a comprehensive open-source library designed to serve as a native typesetting engine for the MEI format [29]. Verovio's primary role is to engrave MEI data directly into SVG without losing rich metadata through intermediate format conversions. The usage of the SVG format is an essential one for multiple reasons, quoting from Verovio's reference book:

> "In a web environment, the addressability can be used for highlighting graphical elements such as notes or any other music symbols. One additional characteristic of SVG is that its XML tree can be structured as desired, and an innovative design feature of Verovio was to go further in the structuring of the output by leveraging this characteristic. Since Verovio implements the MEI structure internally, this key feature of SVG made it possible to preserve the MEI structure in the design of the SVG output in Verovio. Preserving the MEI structure in the SVG output is a considerable overhead in the rendering process but makes it a unique feature of Verovio.
>
> As a result, Verovio output in SVG is not the end of a unidirectional rendering process. Quite on the contrary, it should instead be seen as an intermediate layer standing between the MEI encoding and its rendering that can act as the cornerstone for a bidirectional interaction: from the encoding to the notation, but also from the notation to the encoding through the user interface." [31]

The practical relevance of MEI and Verovio is illustrated by Piano Precision, a GUI tool for audio-to-score visualization. We explore this tool further below; here, the quotation highlights how the MEI structure and SVG output support interaction with score elements.

> "Technically, what makes MEI especially useful is that it can be rendered to graphical SVG output while retaining its semantically meaningful XML hierarchy and unique identifiers for every score element (e.g., ties, notes). Piano Precision renders the scores into SVG output using Verovio, an open-source toolkit for visualising MEI data. This makes it easy to locate interesting objects in a score and, e.g., highlight them in the rendering. The result is interactive images in which users can flip pages, zoom in and out of a score, and click on elements in a score to jump to their playback positions. " [15]

2.1.3 Audio Representations

Audio representations are graphs and/or plots that are intended to represent sound. They are not strictly for representing music; their purpose is to translate sound from the audible to the visual domain. Audio is a term that refers to an acoustic sound that is turned into an electrical signal. Most audio representations are therefore digitally generated by applying mathematical operations to the audio signal, such as the Fourier transform.

We can identify at least three components that define a given audio signal: time, frequency, and amplitude or energy. An audio visualization will present explicit information about at least two of these.

Waveforms are among the most common and simplest ways to represent audio. A waveform consists of a single line in a two-dimensional plot, with amplitude on the y-axis and time on the x-axis. This representation does not provide frequency information; it only shows how the amplitude of the audio changes over time.

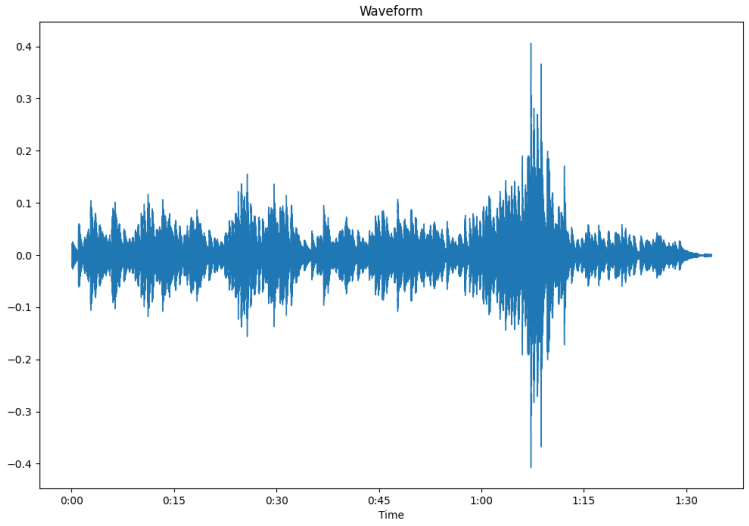


Figure 2.1: Example of an audio waveform.

To represent the three signal dimensions, the plot needs to encode time, frequency, and amplitude or energy. The spectrogram achieves this by placing frequency on the y-axis, time on the x-axis, and using colour to encode amplitude or energy. In this thesis, we use the Mel spectrogram, which organizes frequencies according to the Mel scale, a perceptual scale designed to reflect how human listeners judge pitch distances. This representation is particularly relevant to music analysis and is used in the LayerIt demonstration.

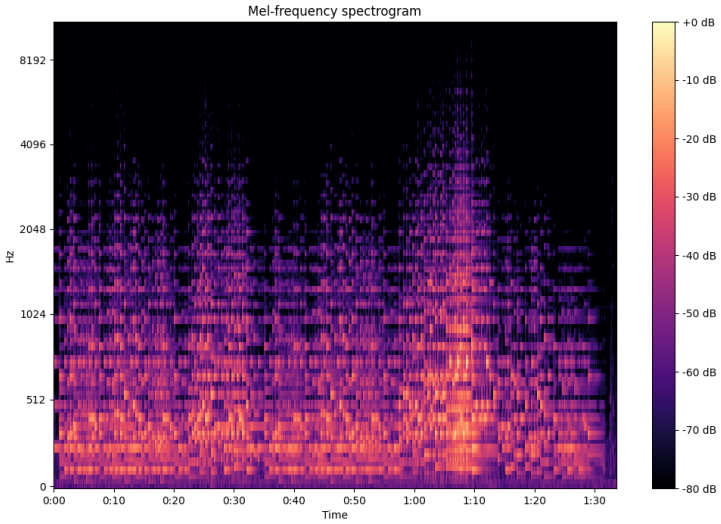


Figure 2.2: Example of Mel spectrogram.

The Constant-Q Transform (CQT) and chromagram are visual representations that derive from the spectrogram. Whereas a spectrogram shows frequency data, the CQT and chromagram can be used to visualize note-related information: tone height in the case of the CQT and chroma in the case of the chromagram. The following quote explains these terms:

> "(...) a pitch can be separated into two components, which are referred to as tone height and chroma. The tone height refers to the octave number and the chroma to the respective pitch spelling attribute contained in the set {C, C♯, D,(...), B}." [22]

The CQT organizes its frequency content using geometrically spaced frequency bins. This is done to resemble the equal-tempered system.

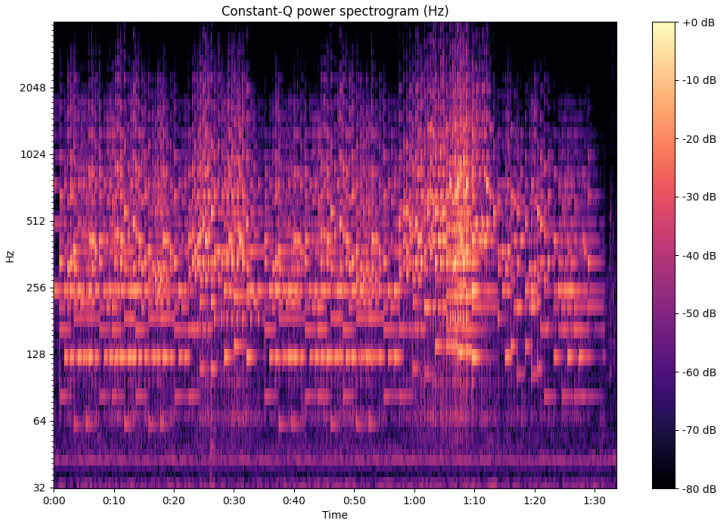


Figure 2.3: Example of a CQT.

While the CQT can distil the frequency spectrum into bins that correlate with individual notes, the chromagram takes this a step further by visually summing the energy of each note across all octaves. Thus, in the end, we obtain a plot divided into 12 lines, each representing a given note. In its essence, this type of visualization shows the use of the same notes across the recording. A chromagram can be useful for determining harmonic and scale-related information [16, 17, 22].

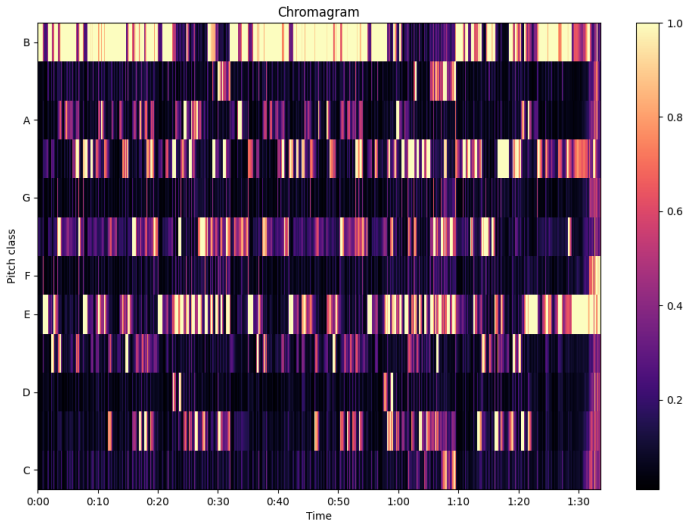


Figure 2.4: Example of a chromagram.

2.2 MIR Visualization Tools

Within MIR, visualization workflows typically fall into two broad categories: GUI-based tools and programming-based workflows. Together, these workflows address complementary needs, ranging from interactive inspection and annotation to custom analysis and visual composition. GUI-based tools emphasize direct exploration, while programming-based workflows enable greater control, customization, and reproducibility. Within this landscape, LayerIt is positioned as a framework for the reproducible composition of score-aligned visualizations.

Sonic Visualiser [3] is an open-source desktop application for the analysis, visualization, and annotation of audio recordings, particularly music recordings. It provides standard visualization facilities and supports plugins for automated analysis methods, allowing users to inspect audio and visualize computational outputs. These capabilities make it useful for general audio analysis as well as for the development and interpretation of MIR methods. Its open-source distribution also supports the adaptation and extension of the application to meet specialized research needs, making it a relevant baseline for research-oriented visual work.

A particularly relevant example is Piano Precision, a Sonic Visualiser fork that is a GUI for visualizing score-aligned spectrograms. By adapting Sonic Visualiser, Piano Precision extends this workflow with a score view and score-aware annotations. The authors describe this relationship as follows:

> "The Piano Precision UI is adapted from Sonic Visualizer, with an additional area displaying a score (...) The interface is designed to help streamline the processes of loading a digital score and a performance recording, editing and visualising annotations (...), and exporting score-aware annotations." [15]

This makes Piano Precision particularly relevant to research involving correspondence between a performance and its score. Its main strength is the integration of interactive navigation and score-aware annotation within a GUI, enabled by extending an open-source audio-analysis environment. To establish this correspondence, Piano Precision uses Piano Aligner, an integrated plugin developed by the same authors that detects onsets in a performance and links them to their corresponding positions in the MEI score. These links allow the interface to synchronize the playback view with the relevant score elements and associate annotations with both modalities.

Specialized MIR research can also require visualizations that combine several data types, such as audio, scores, annotations, and computational outputs, in a layout designed for a specific research question. These combinations are not always available as pre-built views in GUI-based tools. In such cases, programming-based workflows provide a complementary solution. Python libraries such as librosa and Matplotlib, together with tools such as Jupyter Notebooks, allow researchers to implement custom feature extraction, analysis, and visualization pipelines. Such workflows can be versioned, shared, and included directly in publications, which supports reproducibility. They also integrate naturally with machine-learning and large-scale data-analysis pipelines. Their flexibility, however, usually means that the researcher must implement the composition logic, data transformations, and alignment-dependent layout themselves.

LayerIt addresses this composition problem. It provides reusable objects for composing score, audio, annotation, and algorithmic layers into a static visualization. The score-performance alignment is supplied externally, while LayerIt uses that alignment to place the different representations on a shared visual axis and export the result as structured SVG. In this sense, LayerIt combines the customizability and reproducibility of code-based workflows with a framework dedicated to reusable score-aligned visual composition.

2.3 Audio-to-score alignment

Audio-to-score alignment, also referred to as score matching or score alignment, consists of synchronizing an audio recording of a musical performance with its corresponding symbolic score [4]. In practical terms, the goal is to establish correspondences between what is heard in the recording and what is represented in the score, enabling tasks such as score following, guided listening, interactive navigation between audio and score, and research-oriented analysis.

Time To Align! distinguishes graphical, logical, and physical temporal domains and provides a general model for recording alignments among them [13]. In this model, an alignment is a reusable artefact that records correspondences and their provenance across timelines. LayerIt addresses a complementary problem: it uses an available score-performance alignment to compose graphical, symbolic, and audio-derived representations into a shared score-aligned visualization.

For the purposes of this thesis, it is useful to distinguish between **physical time** and **abstract time**. Physical time refers to positions in the performance, such as the duration of the audio recording, samples or frames used by audio features, detected beat times, and model activation probabilities. Abstract time refers to positions in the musical structure represented by the score, such as notes, beats, measures, and other metrical or structural locations. The alignment step maps positions in abstract score time onto positions in the physical performance time. LayerIt then uses this mapping to place score material and other representations on a common visual axis; it does not compute the mapping itself.

The purpose of an alignment is not limited to algorithmic matching. Thomas et al. [35] describe score–audio linking structures as a basis for accessing and exploring multimodal music sources within a single framework. LayerIt uses an available linking structure for a more specific purpose: the generation of static, score-aligned visualizations. It does not compute the alignment itself. Existing synchronization toolkits, such as Sync Toolbox, provide dynamic-time-warping-based pipelines that could supply an upstream alignment in future work [23].

In the literature, it is useful to distinguish between two broad alignment strategies. The first is **time-warped alignment**, in which the score is mapped onto a continuous performance timeline [11]. This approach preserves a direct temporal relationship between score elements and audio playback, and it is especially relevant when every relevant musical event needs to be placed along the time axis. The second is **link-based alignment**, in which the relationship between score and

performance is established through discrete anchor correspondences [15]. Rather than continuously warping the score, this approach links selected score positions to performance positions and uses those reference points to organize the alignment [36].

These two strategies are not separate tasks so much as different ways of representing the same underlying correspondence problem, and the distinction becomes important for the rest of this thesis. Time-warped alignment is the more continuous formulation, while link-based alignment is the more anchor-driven formulation. This terminology will be used throughout the following chapters.

Alignment systems can also be classified by when the matching is performed. **Offline alignment** processes a complete recording after the performance has finished. Because it can access the full signal, offline alignment can use future information to determine the best possible correspondence, which makes it suitable for analysis, annotation, and audio editing. **Online alignment**, also known as score following, performs the matching in real time while the audio is being received. Since it cannot rely on future information in the same way, online alignment typically has to trade off latency against precision.

Beyond this temporal distinction, alignments may be carried out at different levels of granularity. The three most common are **note-level**, **beat-level**, and **segment-level** alignment.

Note-level alignments are score matching processes that aim to synchronize at the individual note level. At its core, note-level alignment is based on note and onset/offset detection. In MIR literature it can also be referred to as "fine-grained alignment" because of the precision it requires. Contemporary audio-to-score alignment research focuses on this type of alignment not only because of its precision but also because it can retrieve data relevant to other MIR fields and tasks. However, pure note-level alignments might falter if the performance deviates from the score, for example if the performer skips, repeats, or adds notes or score sections. Repeats and jumps are a well-established source of difficulty in score-performance synchronization [10]. For this reason, some score-matching solutions combine fine-grained alignment with other methods.

Piano Precision is an example of an application of a link-based note-level score-alignment method. While the recording is playing, Piano Precision shows a spectrogram and waveform that follow a conventional playback cursor, while a side window containing the score highlights the corresponding elements, note by note. This enables annotations to be mapped to both the audio and individual score elements simultaneously. The authors note the usefulness of this tool for performance researchers [15]; however, such a feature is also useful for BT and MIR research more broadly.

Beat-Level alignments aim to follow beats and/or measures specified in the score and link them to the performance. Early score aligners [7] relied on such methods. Although contemporary score-alignment solutions are mostly note-level, they may still combine beat-level alignment stages. For example, [27] improves on other alignment models by analysing "alignments at both the beat and the bar level". Beat-level alignment can provide "anchor points" that prevent "cascading errors, as misalignment will not propagate beyond the next anchor point"; the authors report that this improves automatic alignment results.

Segment type alignments, map broad portions of performance audio to corresponding visual blocks in the score. This type of score matching is mostly made manually, it is common to see it in follow-along videos to help viewers listen to the music while visualizing the score. This type of alignment is considered "weak but musically meaningful" [16] since segments can be deliberately chosen i.e., to better show melodic or harmonic structure.

Among the visual alignment tools relevant to this thesis, ScoreWarp is especially important. ScoreWarp presents a score on a physical time axis, shifting each musical element so that its visual position corresponds to the time at which it sounds [11]. In other words, it implements a time-warped visualization that maps abstract score time onto physical performance time while preserving the semantic structure of the original MEI data. This makes ScoreWarp a strong example of how alignment can be used not only as an algorithmic task, but also as a visual framework for interacting with music information. ScoreWarp's implementation and role in LayerIt are discussed in Chapter 4.

2.4 Summary

This chapter provided a comprehensive overview of MIR visualization techniques and tools, establishing the foundation for understanding audio-to-score alignment.

We began by categorizing music visualizations into three main types: traditional sheet music representations, symbolic machine-readable formats, and audio-based representations. This established the importance of MEI and Verovio, as well as the use of SVG in this context.

We then examined the contemporary landscape of MIR visualization tools, distinguishing between GUI-based solutions such as Sonic Visualiser and Piano Precision, which provide user interfaces for audio analysis, and code-based workflows that afford researchers greater flexibility to develop specialized visualizations tailored to specific research questions. Python-based approaches using libraries such as librosa and Matplotlib have become widely used in MIR research, particularly because of their integration with machine-learning pipelines and reproducibility requirements.

Finally, we introduced one of the core tasks of this thesis: audio-to-score alignment. This process synchronizes musical performances with their corresponding symbolic scores. ScoreWarp emerged as a particularly relevant tool, demonstrating how MEI and Verovio can be leveraged to create visually warped scores that maintain semantic integrity while aligning with performance time.

Chapter 3

Methodology

This chapter presents the methodological choices that guided the development of this thesis. It explains the overall research approach, the iterative process used to build and refine the work, the scope boundaries that define what the project does and does not cover, and the criteria used to validate the end result.

3.1 Research Approach

This thesis follows a design-oriented and implementation-driven research approach, supported by iterative prototyping. Rather than treating the problem as a purely theoretical exercise, the work focuses on building and refining a practical framework for time-aligned music visualization that responds to the needs identified in the literature review and in the reference tools analysed in the previous chapter. The research process is therefore grounded in making, testing, and adjusting concrete visual and interaction decisions as the project evolves.

The methodology combines conceptual analysis with software development. First, the problem space was studied through the comparison of existing tools, visualization strategies, and alignment workflows. That understanding was then used to define a focused system design that could support layered, time-aligned representations of musical data. From that point onward, implementation and validation proceeded together: each prototype was used to verify whether the proposed interaction model, visual structure, and alignment logic were adequate for the thesis goals.

This approach is appropriate because the main contribution of the thesis is not only to describe a concept, but to demonstrate how such a concept can be materialized into a workable framework. As a result, the methodology emphasizes practical feasibility, visual clarity, and alignment consistency, while still remaining informed by the academic context established in the literature review.

Methodologically, the thesis follows a design science approach, in which knowledge is produced by designing an artifact that addresses a problem in context and by investigating that artifact in context [14, 37]. In Wieringa's terms, RQ1 is a design problem, answered by the framework itself, while RQ2 and RQ3 are knowledge questions about the resulting artifact. The design cycle

was carried out through the five prototype iterations described in Section 3.2, and the artifact is validated in Chapter 5 against the criteria defined in Section 3.4.

In retrospect, these iterations follow the divergent–convergent pattern of the Double Diamond [6]: the first three prototypes explored the problem space broadly (web-based annotation, interactive score editing, and extension of an existing GUI), the scope decision in Section 3.3 marks the convergence on a defined problem, and the last two prototypes and LayerIt itself correspond to developing and delivering a solution. The model is used here to describe how the process unfolded, not as a procedure that was prescribed in advance.

3.2 Iterative Development Process

The development of LayerIt followed an iterative path made up of five prototype cycles before the final framework took shape. Rather than moving from a fixed specification directly to implementation, the project evolved through successive experiments in which each prototype was built to answer a specific question, evaluated for what it revealed, and then either discarded or used as the basis for the next step. Seen in this way, the value of the iterations is not only in the software they produced, but in the design knowledge they accumulated.

3.2.1 BeatMapJS

We started by creating **BeatMapJS**. Its goal was to explore a web-based beat-annotation interface and to test whether existing beat-tracking tools could be combined with a more interactive annotation workflow. The prototype was built in JavaScript and HTML using BeatThis [8] and Wavesurfer.js, and it included an interactive spectrogram, region annotation, a tempo-annotation table, and import/export support for Sonic Visualiser-compatible text files. The main lesson was that, although the system was functional, it did not contribute a meaningful new annotation capability beyond the integration of BeatThis. More importantly, it did not produce a reusable Python library, which, from the start, was an important requirement. The decision carried forward was that a reusable Python core would be mandatory.

3.2.2 MEI-Basic-Edit

The second cycle was **MEI-Basic-Edit**. Its goal was to test whether manual score alignment could support audio-to-score annotation by allowing score elements to be adjusted against an audio layer. The prototype was built in JavaScript and HTML with Verovio, MEI, and SVG, and it rendered a score below a static spectrogram while allowing users to drag score elements horizontally to align them with the audio representation. It also included an embedded XML editor for direct manipulation of the underlying score code. The key lesson was that interactive score editing was expensive both conceptually and technically: it increased UI complexity and required a level of implementation effort that was not justified by the results. At the same time, this cycle provided valuable familiarity with MEI, Verovio, and SVG structure. The decision carried forward was that

SVG would remain a viable composition substrate, but interactive editing would not be the main direction.

3.2.3 BeatPrecision

The third cycle was **BeatPrecision**, a fork of Piano Precision. Its goal was to extend an existing GUI environment with beat-annotation features while preserving the score-alignment functionality already present in the software. The prototype was built in Qt/C++, integrating BeatThis and implementing horizontal score-alignment alongside the existing spectrogram view. While functional, the project revealed significant challenges. Piano Precision's (and by extension Sonic Visualiser's) codebase proved complex and difficult to manage. Long compile times and the sheer size of the codebase made each developed modification increasingly challenging to implement, manage, and debug. More importantly, this iteration taught us a critical lesson: the need to distinguish between the interaction and visualization problems in BA. We realized we could not tackle both simultaneously and would need to choose a direction. Recognizing a clear gap in the field, we opted to focus on the visualization aspect, as we saw greater potential for the thesis to benefit the community in this area.

3.2.4 Score-Alignment.py

The fourth cycle was **Score-Alignment.py**. Its goal was to explore score-aligned spectrograms in a Python environment and to assess the Modusa framework as a possible basis for the project. The prototype was built in Python with Modusa and Matplotlib and produced an interactive spectrogram. The lesson was that a truly fluid interactive visualization, with zooming, panning, a timeline, and playback control, was much harder to implement than expected. Even though the prototype worked, the result was not satisfactory enough to justify continuing in that direction, and the return on effort was not viable. The decision carried forward was to abandon real-time interactivity and instead prioritize static, high-quality, reproducible output.

3.2.5 Score-Alignment.js

The fifth cycle was **Score-Alignment.js**. Its goal was to revisit the aligned-spectrogram idea using the lessons learned from the earlier MEI and beat-annotation prototypes. The prototype was built in JavaScript and HTML with Verovio, MEI, and Wavesurfer.js, and it displayed a MEI score alongside a spectrogram while storing beat timestamps in the MEI data and highlighting notes on interaction. The lesson from this final exploratory cycle was that the idea was promising, but it also confirmed the need for a tool that was strictly focused on visualization, written in Python, and organized using OO principles so that it could be reused and extended more easily. The decision carried forward was that this combination of requirements defined what would then be LayerIt.

Table 3.1: Summary of development iterations

Iteration	Goal	Tools Used	Lesson	Conclusion
BeatMapJS	Explore web-based beat annotation	JS/HTML, BeatThis, Wavesurfer.js, interactive spectrogram, region annotation, tempo table, Sonic Visualiser-compatible import/export	No new annotation feature beyond BeatThis; no reusable Python library	Reusable Python core is mandatory
MEI-Basic-Edit	Test manual score alignment against audio	JS/HTML, Verovio, MEI, SVG, drag-to-align score, XML editor	Interactive score editing is costly; gained MEI/Verovio/SVG familiarity	SVG as composition substrate; no interactive editing focus
BeatPrecision	Extend GUI beat annotation in an existing score-aligned tool	Qt/C++, BeatThis, horizontal score follow-along	Codebase was hard to extend; interaction and visualization must be separated	Focus on visualization, not the interaction problem
Score-Alignment.py	Explore score-aligned spectrograms in Python	Python, Modusa, Matplotlib, interactive spectrogram	Fluid interactivity proved impractical	Prefer static, high-quality, reproducible output
Score-Alignment.js	Revisit aligned spectrograms with richer score interaction	JS/HTML, Verovio, MEI, Wavesurfer.js, beat timestamps in MEI, note highlighting	Promising, but confirmed need for a Python OO visualization tool	Define the LayerIt conceptual baseline

Taken together, these five cycles show a deliberate narrowing of scope. The project moved from an interactive beat-annotation GUI towards a reusable OO visualization framework. Each prototype clarified one part of the problem and ruled out another, so the final system could be built on a more stable understanding of what the thesis needed to solve.

3.3 Scope Decision

The development process described above led to an important scope decision. At the outset, the project was framed broadly as a possible BA/BT graphical user interface, with both annotation and visualization treated as potential contributions. However, the iterative prototypes made it clear that these were, in practice, two different problems, each with its own technical demands, design goals, and implementation costs. Attempting to solve both within a single master's thesis would have diluted the contribution and made it harder to reach a result that was both robust and meaningful.

For that reason, the project deliberately narrowed its focus from an interactive BA/BT GUI to an OO visualization framework. The interaction problem proved too broad to address properly alongside the visualization problem, while the visualization side revealed a clearer and more relevant gap: the lack of a reusable, flexible, and well-structured Python tool for producing layered

time-aligned musical data visualizations. By focusing on this gap, the thesis could concentrate on a contribution that was both feasible and valuable.

As a result, this thesis does not attempt to provide a complete BA/BT annotation environment, nor does it seek to solve the broader interaction problem in full. Instead, it focuses on the visualization dimension, with particular emphasis on clarity, modularity, and alignment consistency. This narrower scope makes it possible to develop a more coherent contribution and to validate it against well-defined criteria.

3.4 Validation Criteria

The validation of this thesis is guided by five criteria chosen to reflect the actual purpose of the framework. Since the goal is not to produce a general-purpose MIR platform, but rather a reusable visualization tool, the validation focuses on whether the final system succeeds in covering the visualization cases that motivated its development, while keeping the implementation manageable and the output reliable. The criteria are:

- **Coverage**: the ability to represent the score-aligned visualizations required by the thesis, including layered combinations of symbolic score data with audio-derived and annotation-derived information.
- **Composability and regeneration**: the ability to combine, modify, and re-render existing visual layers through a repeatable procedure rather than manual image alignment.
- **Reproducibility**: the extent to which visual outputs can be generated through a process that can be repeated, inspected, and shared without ambiguity.
- **Extensibility**: the extent to which the framework can be adapted to new visual layers or new combinations of existing ones without requiring major restructuring.
- **Fidelity and limits**: the extent to which the resulting figures preserve the musical and temporal relationships they are intended to represent, while making clear where the visual abstraction becomes less precise, less general, or less appropriate.

Coverage addresses RQ1; composability and regeneration, reproducibility and extensibility address RQ2; fidelity and limits address RQ2 and RQ3.

Defining these criteria makes the validation of the thesis a deliberate design assessment rather than a mere presentation of the implemented tool.

Chapter 4

LayerIt Design and Implementation

This chapter presents LayerIt [1] as the concrete outcome of the research process described in the previous chapter. Its role is to show how the thesis moves from a problem statement and a set of methodological decisions to a working framework for composing score-aligned MIR visualizations. The chapter is organized from the problem definition to the system overview, then to the design choices that shaped the implementation, and finally to the mechanism used to combine the different visual layers into a single result.

4.1 The Problem and Our Solution

The central problem addressed by LayerIt is that score-aligned MIR figures are often created through a mixture of manual editing, *ad hoc* scripting, and tool-specific workarounds. In practice, this makes it difficult to produce figures that are reusable, reproducible, and easy to adapt when the underlying data changes. Existing GUI tools are useful for inspection and annotation, but they are not designed to support the kind of compositional, publication-oriented visual output required by this thesis.

LayerIt responds to this problem by treating visual elements as composable layers rather than as a single fixed image. The framework provides a structured way to generate, combine, and align music-related visual materials in SVG-based output. ScoreWarp provides the alignment backbone. LayerIt's contribution is an OO layer-composition model that supports the creation, modification, and regeneration of visual compositions.

4.2 LayerIt Overview

LayerIt is a Python framework designed to compose score-aligned MIR visualizations from independent visual components. At a high level, it takes aligned musical data, generates the corresponding visual representations, and arranges them into a layered figure that preserves the tem-

[1] The thesis figures were generated using the development repository at https://github.com/FernandoAze/LayerIt_WIP; the official repository published with the LBD is https://github.com/FernandoAze/LayerIt.

poral relationship between the score and the accompanying audio-derived or annotation-derived information.

The framework is modular: the same base structure can support different combinations of score material, spectrograms, annotations, beat information, and other derived data. These combinations can be modified and regenerated through scripts rather than reconstructed manually in an image editor.

The project uses librosa [21] and Matplotlib [34] to generate audio and data visualizations. The score layer is rendered through ScoreWarp. The final output is an SVG file that combines the score with audio and data layers. The BT examples we are going to present were produced with BeatThis [8], in order to illustrate how precomputed BT outputs can be incorporated to produce beat-data visualizations.

4.3 Design Choices and Rationale

LayerIt uses Matplotlib to generate audio and data representations when SVG is the final output format. Verovio, and subsequently ScoreWarp, render the score in SVG. Exporting Matplotlib output to SVG therefore provides a practical route for combining chart-based visualizations with the rendered score.

SVG is a vector format, so representations that can be expressed as Cartesian plots, such as continuous curves and time-based annotations, can be exported without resolution loss. Other representations, particularly heatmaps such as spectrograms, are ordinarily rendered as raster images. Although SVG can embed such images, it is a static format and does not provide the interactive zooming and panning of a dedicated visualization interface. LayerIt therefore supports PNG output for raster-based plots. This trade-off preserves the resolution independence of the score and vector plots, while raster layers may lose detail when enlarged and require greater storage.

Because SVG is a text-based markup format, it can also be inspected and modified directly. It can therefore serve as an intermediate representation between codebases written in different languages, such as Python, JavaScript, or C++. This property supports the potential integration of LayerIt into future MIR applications.

LayerIt was developed as a Python-based OOP tool using established MIR libraries, including librosa and Matplotlib. This choice keeps the framework compatible with common Python-based research workflows.

4.4 Generating Visuals

LayerIt was developed to generate visualizations for BA and BT tasks, so the current version focuses on these uses. It uses audio-to-score and data alignment to place relevant representations on a common visual timeline. In its current state, the codebase relies on the following files to generate complete visualizations:

File	Description
`.wav`	File with the audio recording of the performance
`.mei`	XML Score of the musical piece
`.maps.json`	Contains data that links onsets with score elements of the .mei
`.svg`	The score in visual format generated by ScoreWarp (requires .mei and .maps *a priori*)
`.npz`	File with algorithmic BT data

Table 4.1: Files required by LayerIt to generate complete visualizations.

4.4.1 Getting the Warped Score

The warped score can be generated using the ScoreWarp online demo. The user uploads the score in `.mei` format together with the MAPS file, then downloads the resulting score in `.svg` format.

The `.mei` file contains the score in XML format. A score in `.musicxml` format can be converted to `.mei`, for example with the Verovio online editor or MuseScore. In either case, the original score must be available in `.musicxml` format before conversion.

4.4.2 Getting the MAPS file

The MAPS file records performance onsets and links them to the corresponding score elements in the `.mei` file through their `xml:id` attributes. Two methods can be used to obtain this file.

The first method, described in [11], uses the MIDI-to-MIDI matching algorithm of [25]. It requires both the `.mei` score and a MIDI performance. Examples from the ASAP dataset [27] can facilitate this process. When a MIDI performance is unavailable, the MAPS file must be generated independently.

The MAPS file is a `.json` file. For the current version of LayerIt, it must provide score and performance onset data in the following form:

Listing 4.1: Minimal structure of the MAPS.json file to work with LayerIt and ScoreWarp

```
{
    "obs_mean_onset": 0.853333333,
    "xml_id": ["x1iw1eq1"],
    "obs_num": 1
}
```

In this example:

- `obs_mean_onset` is the time in seconds of a given onset.
- `xml_id` is the `xml:id` of the corresponding onset note in the `.mei` score. For a chord onset, this attribute can contain more than one `xml:id`.
- `obs_num` is a sequential counter for the supplied onsets.

LayerIt relies on MAPS in two complementary ways: directly for onset visualization, and indirectly through ScoreWarp, which requires MAPS to generate the warped score aligned with the timeline SVG element which constitutes the alignment reference for all overlaid data.

These components occupy different roles in the workflow. MAPS is the purpose-specific alignment representation used in this thesis: it records performance onset times and the corresponding `xml:id` anchors in the MEI score. ScoreWarp consumes these anchors to generate the time-warped score and timeline reference that LayerIt uses to position its visual layers. Time To Align!, discussed in chapter 2, provides a more general model for representing alignments and their provenance across timelines. The match file format is different in kind: it is an interchange format for encoding note-level correspondences between scores and performances [9], not an alignment or visualization framework. LayerIt consumes an available alignment to compose visual layers into a static figure; it does not compute the alignment and does not currently read or produce match files. Using match-file data with the workflow described here would therefore require a conversion step to the MAPS and ScoreWarp inputs, rather than treating the format itself as an alternative to LayerIt.

The relationship between these alignment methods, representations, and LayerIt's composition stage is summarized in Figure 4.1. Solid arrows show the workflow used for the thesis examples; dashed arrows indicate alternative routes or possible future integrations.

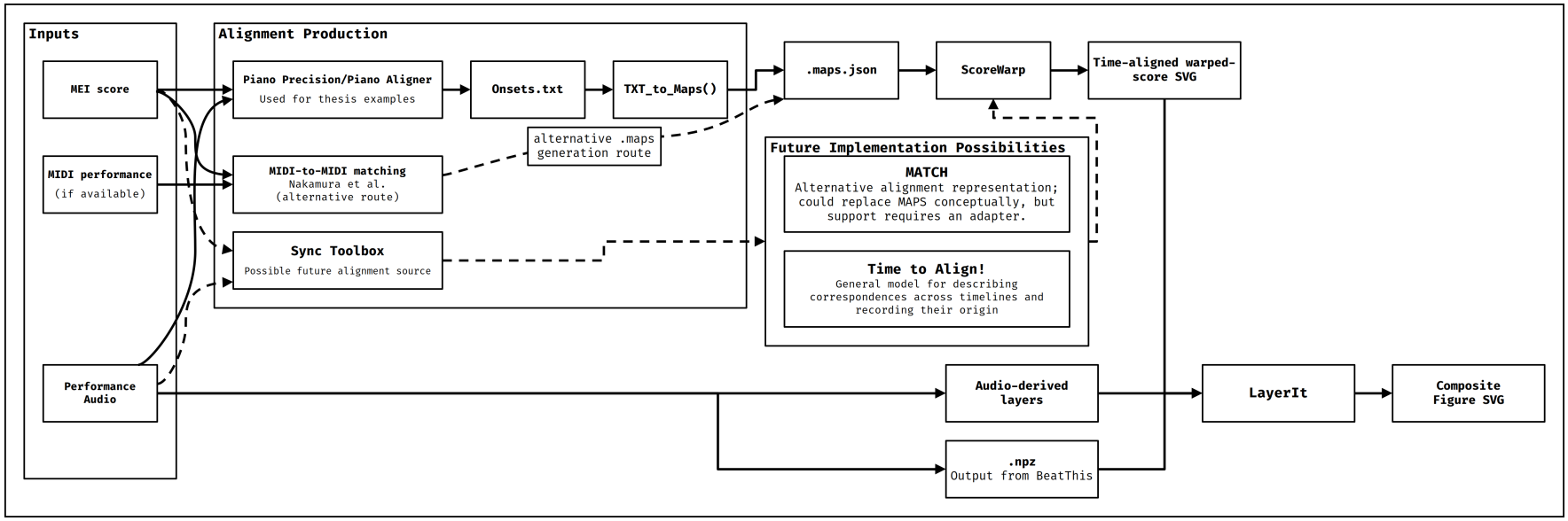


Figure 4.1: Workflow positioning of alignment methods and representations relative to LayerIt. The solid path shows the thesis example workflow: Piano Precision/Piano Aligner produces onset annotations, which are converted to MAPS and used by ScoreWarp to generate the time-aligned score SVG; LayerIt combines that score with audio-derived layers and BeatThis output. Dashed paths show alternatives or future possibilities, not integrations used in this thesis. The match file format is an alternative alignment representation that would require an adapter for this workflow [9]; Time To Align! is a general model for describing correspondences across timelines and recording their origin [13]. Sync Toolbox is shown as a possible upstream alignment source, whose compatibility with the current MAPS/ScoreWarp input has not been established [23]. The Nakamura MIDI-to-MIDI route is an alternative MAPS-generation method [25]; ScoreWarp and Piano Aligner are described in [11, 15].

ScoreWarp performs best when MAPS contains dense onset annotations across the full performance. However, it can still align scores when annotations are sparse via interpolation between available onsets and their @xml:id anchor points. In practice, this allows different precision levels (i.e., downbeats, phrase boundaries, or measure-level anchors), trading annotation effort for alignment precision. Even minimal anchors (first and last onset) may be usable for short performances or segments, but intermediate alignment becomes less reliable due to interpolation error.

For the examples provided with LayerIt, Piano Precision was modified to obtain this data. Piano Precision's Piano Aligner plugin [15] matches performance onsets to score elements defined by the `.mei` file. Piano Precision displays these correspondences in a Sonic Visualiser time-instance layer. The modified layer displays the `xml:id` of each onset rather than its score position.

The time instances can then be exported as `.txt` files containing the minimum required MAPS data using the `TXT_to_Maps()` function in `src/functions/warp_score`.

4.4.3 Providing BT data

Contemporary BT algorithms commonly produce a beat activation function: a continuous curve representing the estimated likelihood of a beat at each frame. Visualizing this curve alongside the discrete beat estimates exposes the model output from which those estimates are derived and provides qualitative context for interpreting them.

LayerIt does not run a BT algorithm itself; it consumes a pre-computed `.npz` file containing the algorithmic output. For the examples in this thesis, this file was produced by running BeatThis on the performance recording and saving its output with `numpy.savez`. Any other beat tracker can supply this data, provided the `.npz` file follows the structure expected by the beat layers in `src/functions/Beat_Layers.py`. The file should contain five key components:

Field	Purpose
`beat_times`	Time coordinates (seconds) aligned with probability frames
`beat_activation`	Continuous beat activation function values
`downbeat_activation`	Continuous downbeat activation function values
`detected_beats`	Discrete beat positions derived from peak detection
`detected_downbeats`	Discrete downbeat positions from peak detection

Table 4.2: Fields required in the `.npz` file for BT visualization.

The activation fields capture continuous model outputs across time, while the detected positions represent the final beat estimates. This separation enables the visualization of both the continuous activations and the discrete estimates.

4.5 Layer Alignment Method

LayerIt aligns each visual layer with the performance-time reference established by the warped score. ScoreWarp can include a visual timeline whose span may be greater than the recorded performance; its time span and pixel length provide the single scale factor used by LayerIt. Each

layer converts its native coordinates, such as samples or frames, to seconds where necessary and is drawn at that scale. Horizontal alignment therefore follows the shared timeline, while vertical placement is determined by the composition. In the codebase, this calculation is performed by `Visualizer().get_Layers_WidthHeight()`.

4.6 Layers on Layers, within Layers

LayerIt combines aligned visualizations of algorithmic data, audio, and symbolic music representations, such as scores, that share a common timeline. Its Python OO architecture represents these visualizations as layers that can be composed into larger outputs. This organization supports the modification and regeneration of the visual compositions described in Chapter 5.

The visualizations generated by LayerIt follow a "layers within layers" philosophy: visual elements are self-contained components that can be composed into upper-level elements, which in turn can be composed into yet higher levels. Figure 4.2 demonstrates this methodology.

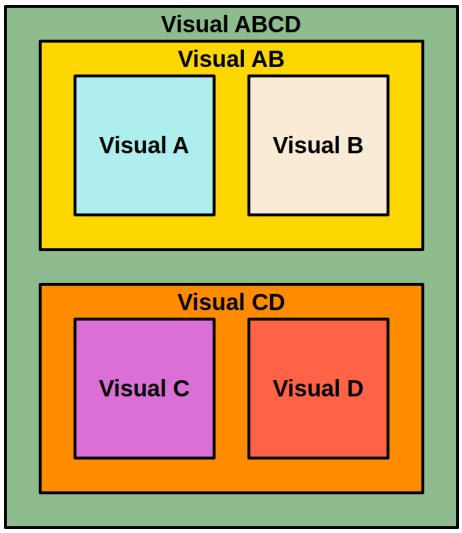


Figure 4.2: Composition of the LayerIt visualization system. Each coloured box represents a distinct visual element that composes a higher-level visualization.

This "layers within layers" design is applied at three levels of the project: the **visual level**, where visualizations combine aligned data, audio, and musical elements; the **SVG level**, where LayerIt organizes the XML structure of the output; and the **code level**, where the OO architecture organizes the corresponding program components. Together, these levels describe how individual elements are combined into composite visualizations.

4.6.1 Visual Level

At the visual level, layers are output elements aligned to a shared timeline. They can be arranged vertically in a composite view while retaining the temporal correspondence between the score, audio-derived representations, annotations, and BT output.

The concrete layer implementations are built on four shape abstractions. Each shape inherits from `Layer` and supplies the rendering behaviour for a particular kind of data; a concrete layer provides the data through the corresponding hook and configures its visual properties. The four shapes are:

- `Curve`: represents a sampled two-dimensional signal as a line. It is suitable for continuous time-series data such as waveforms, activation curves, and onset-strength envelopes. A curve can use the panel's primary axis or a shared secondary axis.
- `Events`: represents a set of instantaneous time positions as vertical markers. It is used for discrete events such as onsets, beats, and downbeats.
- `Intervals`: represents contiguous regions selected from an activation signal after applying a threshold. These regions are rendered as highlighted windows, making periods of elevated activation visible across a panel.
- `Field`: represents a dense two-dimensional matrix, such as a time-frequency or time-pitch representation. The matrix is normalized and rendered as an embedded raster image, while its axes are added to the SVG group.

These abstractions separate the representation of data from the common rendering and SVG-export mechanisms. They are therefore the basis for implementing different layer types without requiring each layer to reimplement the complete drawing pipeline. The distinction also clarifies the intended visual encoding: curves show values over time, events show isolated time positions, intervals show selected time regions, and fields show values distributed over two dimensions.

4.6.2 SVG Level

At the SVG level, LayerIt organizes the XML structure of the output so that individual and combined layers remain identifiable. The following listing shows a simplified representation of the XML structure produced for a composed SVG file:

Listing 4.2: Example structure of an SVG file

```
<svg>
|   <Top Visualization>
|   |   <Layers>
|   |   |   <image spectrogram.png>
|   |   |   <BeatThis Output>
|   |   |   <Onsets>
|   |   <Score>
|   |   <timeAxis>
|   <Bottom Visualization>
|   |   <Layers>
|   |   |   <Waveform>
|   |   |   <Beat Probability>
|   |   |   <Downbeat Probability>
</svg>
```

This XML organization allows the SVG to be inspected and modified directly through its markup. It also preserves an identifiable structure for code-based processing while keeping the SVG readable. The nesting of elements at the XML level mirrors the visual composition described above.

4.6.3 Code Level

At the code level, LayerIt's OO architecture organizes each visualization component as a self-contained module with a consistent interface. The same composition principle used at the visual and SVG levels is therefore represented in the code structure. The following snippets illustrate the base abstraction, the `Visualizer` composition API, and a short end-to-end workflow.

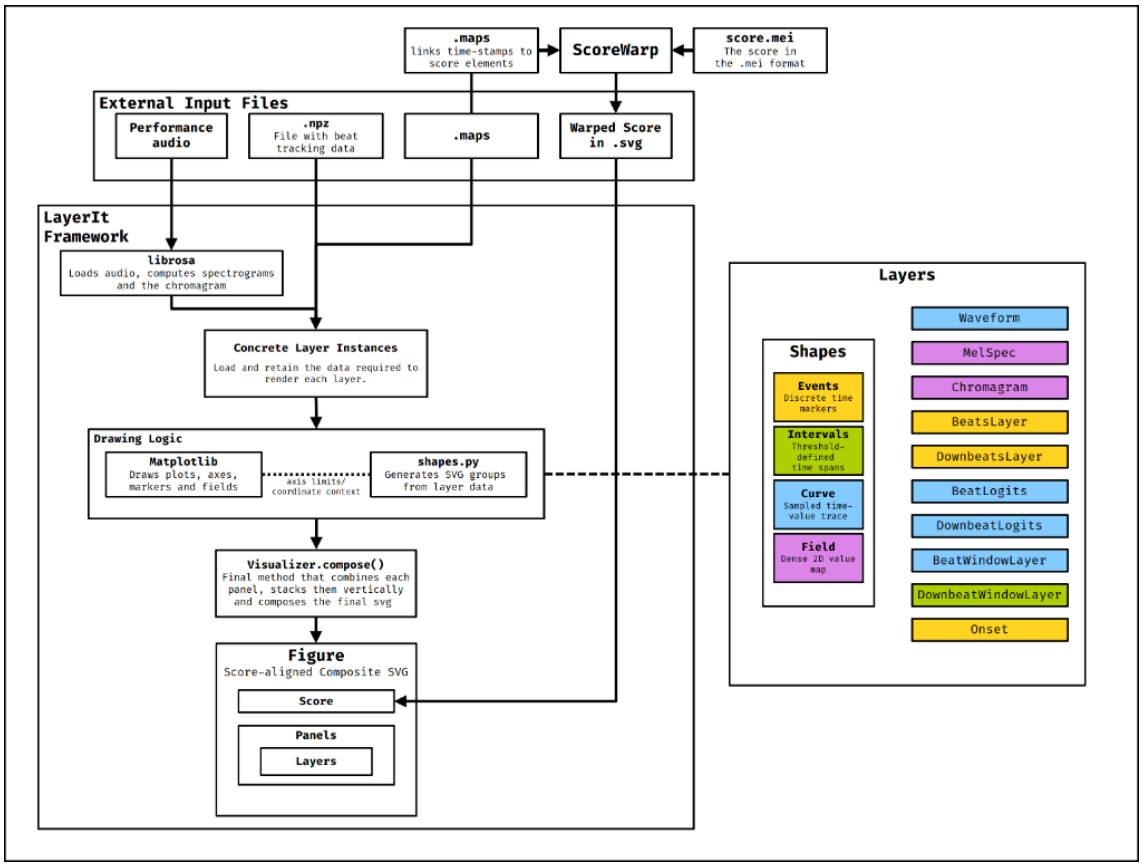


Figure 4.3: Architecture of LayerIt. External audio, beat-tracking, alignment, and score inputs feed the framework; ScoreWarp produces the warped score, while LayerIt loads audio and constructs layer instances. Matplotlib and the shape abstractions generate the individual layers, which `Visualizer.compose()` combines into a score-aligned SVG.

Listing 4.3 shows an illustrative `Layer` base-class interface. The omitted imports are not relevant to the interface.

Listing 4.3: Layer interface (concise)

```
class Layer(ABC):
  def __init__(self, name: str = "Layer"):
    self._data = None
    self.name = name

  @abstractmethod
  def load_data(self, **kwargs) -> bool:
    pass

  @abstractmethod
  def draw(self, ax: Axes, shared_data: Dict[str, Any]) \
           -> Tuple[List, List]:
    pass

  def to_svg_group(self, shared_data: Dict[str, Any]) \
                   -> Optional[str]:
    '''Optional SVG export for subclasses'''
    return None
```

In LayerIt, a `Visualizer` instance orchestrates the composition of multiple layers through a panel-based workflow. Layers are grouped into panels using `add_panel()`, which allows multiple layers to be rendered together on the same axis. The `Visualizer` maintains a shared data dictionary that all layers access during rendering via `load_all_layers()` and `draw()`. The `compose()` method then renders all panels and combines them with the warped score into a final multi-panel SVG. Listing 4.4 shows the concise composition script used for the LBD figure.

Listing 4.4: Concise LayerIt composition script

```
fig = Visualizer(audio, score, maps, beats)
fig.add_panel(Onset(), BeatLogits(), DownbeatLogits(), height_scale=.5)
fig.add_panel(Waveform(), BeatsLayer(), DownbeatsLayer(), height_scale=.5)
fig.add_panel(MelSpec(freq_window=(100, 1500)))
fig.compose("LBD_FIG.svg")
```

The script initializes the shared inputs, groups layers into three panels, and delegates rendering and SVG assembly to `compose()`. The first two panels combine curves and event markers on shared axes, while the third adds a mel-spectrogram field. The lower-level panel rendering and SVG stacking operations are shown in Appendix A.

LayerIt supports a workflow from Matplotlib-based visualization generation to SVG or PNG composition. The final composition stage can include externally produced graphics, allowing code-generated visualizations to be combined with images created outside the codebase.

This hybrid design supports multiple workflows, including Python-based prototyping and the

use of externally generated images. Custom layer implementations and external image inputs can both be incorporated into the same composite visualization.

These code-level choices provide the compositional mechanism validated in Chapter 5. The case study demonstrates that existing layers can be modified, exchanged, and regenerated through the common interface. Section 5.5 adds one new layer type through the same interface; broader extensions remain future work.

4.7 Summary

This chapter closes the methodological and implementation stages of the thesis by connecting the development process described in Chapter 3 with the framework described in this chapter. The iterative path taken through the five prototypes clarified the scope of the work, showing that the most meaningful contribution was not a new annotation system, but a reusable visualization framework grounded in time-aligned music data.

LayerIt was then shaped to answer that need: a Python OO framework for composing score-aligned MIR visualizations through layered SVG-based output. Its design emphasizes reuse, composability, and visual consistency, while relying on ScoreWarp for the alignment backbone and on established MIR libraries for the generation of the underlying audio and data layers. Taken together, the two chapters show how the project moved from exploratory prototyping to a focused implementation that can now be assessed against the criteria established in the methodology chapter.

Chapter 5

Validation

This chapter validates LayerIt against the five criteria defined in chapter 3. Each criterion is considered through the following evidence:

- **Coverage**: examined through the reproduced score-aligned figure types and their combinations of symbolic, audio-derived, and annotation-derived representations.
- **Composability and regeneration**: examined through the shared panel and scripting workflow used to combine, modify, and regenerate visual layers.
- **Reproducibility**: examined through independently re-runnable scripts that generate the demonstrated figures from prepared inputs.
- **Extensibility**: examined through the `OnsetNovelty` probe, which adds a new time-based layer without modifying the composition mechanism or core layer and shape abstractions.
- **Fidelity and limits**: examined through the discussion of alignment quality, rendering behaviour, representation fidelity, and the boundaries of the available evidence.

Following the design science framing of Chapter 3, this chapter uses the term validation in the sense established in software engineering research. Wieringa [37] distinguishes validation, which investigates whether an artifact interacting with its intended context produces effects that satisfy its requirements before it is transferred to practice, from implementation evaluation, which studies the artifact after it has been adopted by its stakeholders. Shaw [33] likewise treats validation as the evidence offered for a software engineering result, of which evaluation against stated criteria and examples of use are recognised types. This chapter combines both: LayerIt is assessed against the five criteria of Section 3.4 through examples built on real scores, performance recordings and beat-tracking outputs. The term is not used here in the verification-and-validation sense, where validation denotes confirming that a product fits its stakeholders' intended use [2]. As no study with target users was conducted, that claim is not made.

The validation reported here is therefore qualitative. It combines a capability demonstration based on figures generated with LayerIt, a contextual comparison with existing workflows, and an explicit discussion of the evidence that remains to be collected.

The validation is not based on quantitative metrics, for two reasons. First, the object of this thesis, the composition of heterogeneous representations into a score-aligned figure, is not an established MIR task, and there is no ground truth or accepted metric for the quality of such a figure. Established metrics exist for the components that LayerIt consumes: beat-tracking estimates can be compared with reference annotations using the measures implemented in mir_eval [30], and audio-to-score alignment has its own evaluation procedures [5]. Applying them here would assess BeatThis and the alignment supplied to ScoreWarp, both external to LayerIt, rather than the composition framework. The case studies in this chapter should therefore not be read as a metric-based benchmark of beat-tracking or alignment accuracy. Second, the usefulness of the resulting figures for annotation and evaluation workflows (Section 5.4.2) would be best assessed through a study with target users, which was considered premature at this stage of development. Usability evaluation of early-stage designs can add little value and may suppress promising ideas [12]; toolkit research rarely meets the assumptions of standard usability testing, such as minimal training and standardized tasks [26]; and evaluating a toolkit before its users have become familiar with it risks an unfair assessment [19]. Such a study is identified as future work (Section 6.2).

5.1 Validation Approach

The validation considers whether the framework can express the score-aligned, multi-modal figures that motivated the thesis, and whether its composition model supports their creation, modification, and regeneration once a score-performance alignment is available. The objective is not to claim that LayerIt replaces interactive annotation software, reduces annotation effort, or establishes the accuracy of an alignment algorithm. Instead, it assesses the framework as a reproducible means of composing visual material from modular layers.

Three of the four figures in Section 5.2 are original outputs generated locally from the same case study: J.S. Bach's fugue from BWV 856, performed by András Schiff. The inputs are the audio recording, its MEI score, a MAPS file, the ScoreWarp-generated SVG score, and BeatThis output. The remaining figure (Figure 5.1) uses a second case study, the opening bars of Debussy's *Clair de Lune* performed by Maria João Pires, to demonstrate the framework's audio-representation layers independently of BT data. The figures are therefore not reproductions of published images. Published work is referenced only to motivate the figure types and the workflow gap addressed by the framework.

The validation distinguishes one-time preparation from per-figure work. Producing a score-aligned figure requires preparing the MEI score, MAPS annotations, warped score, and, where relevant, algorithmic output. These are substantial prerequisites shared by all LayerIt figures. Once they are available, the scripts supplied with this thesis define and compose the desired layers, and can be re-run when the input data or visual parameters change.

5.2 Reproducing Composite Figure Types (RQ1)

RQ1 asks how heterogeneous MIR visualizations can be combined in a common score-aligned representation. The four examples in this section address this question by composing symbolic notation, audio-derived representations, onset annotations, and BT output around a warped score timeline. They cover both dense, multi-layer views and a time-based display.

5.2.1 Score, audio, and chroma on a shared axis

The first example combines a time-warped score, a waveform, a mel spectrogram, and a chromagram for the opening six bars of Debussy's *Clair de Lune*, performed by Maria João Pires. This performance was chosen as a more demanding expressive counterpart to the relatively steady BWV 856 example introduced below: its varying tempo and compound 9/8 metre make the score–performance relationship less straightforward. It demonstrates that LayerIt can compose notation and complementary audio representations on a shared time axis in this more expressive case. Cyan dotted onset markers are repeated across the spectrogram and waveform, making their common performance-time reference visible. The warped score allows for the notation and the audio representations to be read against the same time axis allowing for a more contextualized and informed visualization.

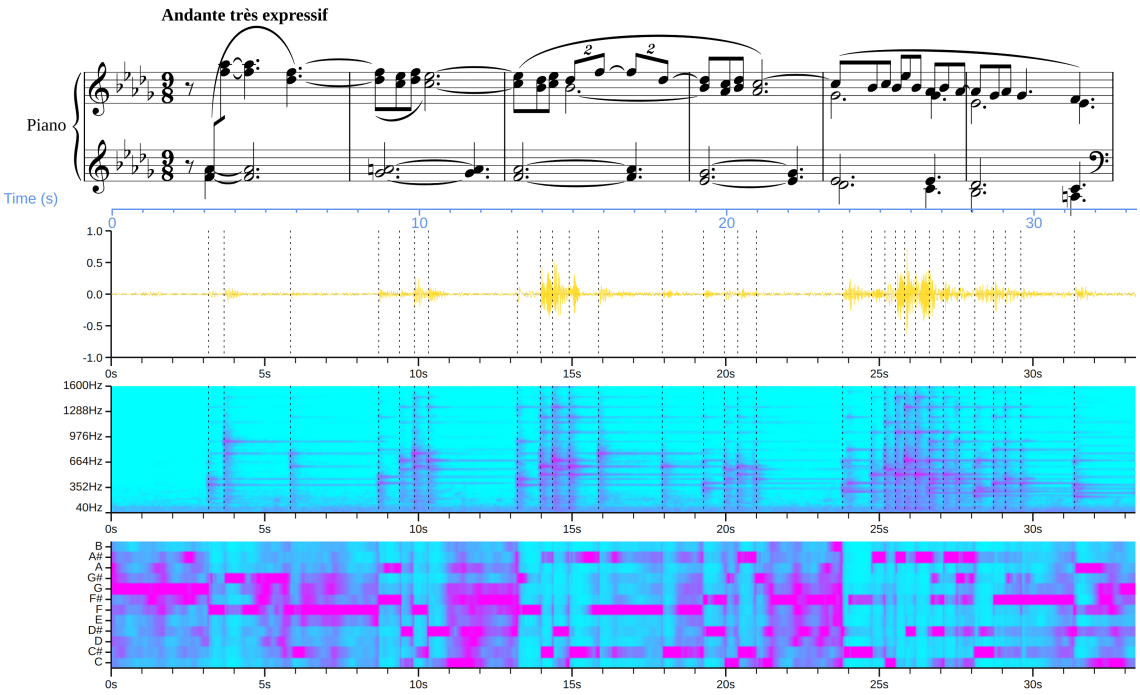


Figure 5.1: LayerIt composition for the opening six bars of Debussy's *Clair de Lune*, performed by Maria João Pires. From top to bottom: time-warped score, waveform, mel spectrogram, and chromagram. The waveform is shown in yellow, and cyan dotted lines mark score-aligned onsets across the time-based panels.

5.2.2 Score-aligned BT analysis

BWV 856 was chosen as a more straightforward baseline: it is a Baroque piece performed at a comparatively steady tempo. This example demonstrates LayerIt's ability to compose score, audio, onset, and beat-tracking layers on a shared time axis without the greater tempo variability of the *Clair de Lune* case. BeatThis was used as the tracker because it was considered state of the art at the time of thesis production, and its open-source implementation made it easier to integrate [8].

The second example places a time-warped score, a spectrogram, BeatThis beat estimates and on- set markers in a single composition. The score provides musical context directly beneath the time-based data. This is valuable when interpreting a BT result because the reader can relate an estimated beat to the corresponding notated position on the warped score.

The figure type is motivated by the visualization in Sá Pinto's BT evaluation work [32] where musical context and time-based analysis are complementary but not physically aligned. LayerIt does not change the evaluation procedure proposed in that work; it provides a way to present the relevant score, spectral, and beat information on a shared axis.

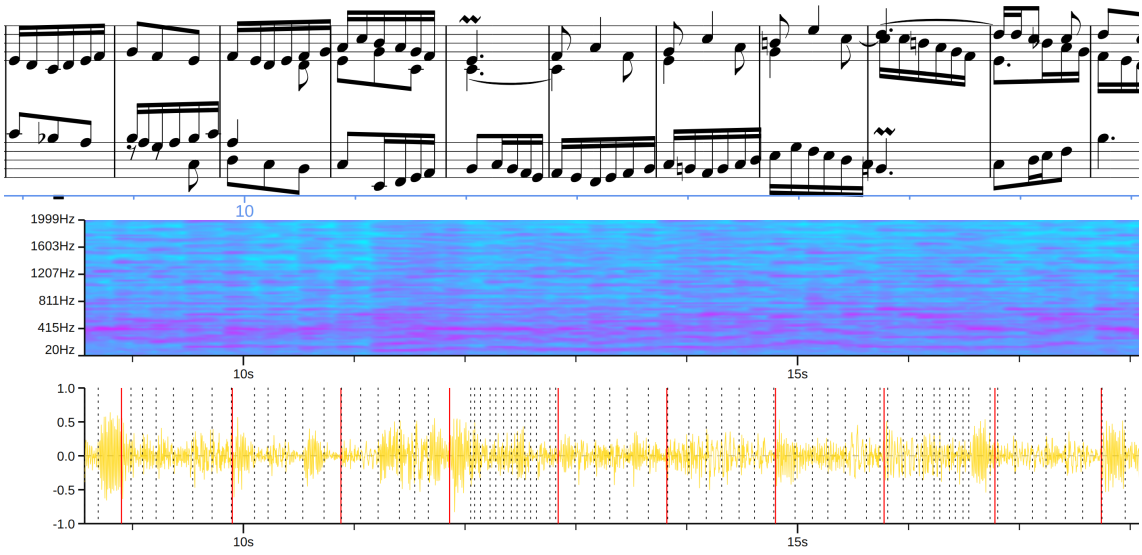


Figure 5.2: LayerIt score-aligned BT display for BWV 856, performed by András Schiff. From top to bottom: time-warped score, mel spectrogram, and waveform with BeatThis beat estimates shown as red vertical lines, and onsets from Piano Aligner shown as black dotted vertical lines. All panels share the performance-time axis.

5.2.3 Logits and position displays

BT analysis benefits from retaining both the discrete output of an algorithm and its frame-wise activation/logits functions. Figure 5.3 presents two such views. In the upper view, detected beat positions are displayed together with the corresponding activation curve. In the lower view we see

the same data this time for downbeat estimates and their respective logits. In both cases, the score, the logits curves and beat/downbeat estimates, remain tied to the same horizontal axis.

This type of visualization can help BT researchers better understand and relay how their algorithms perform. In this case we can visualize how the estimates for beat and downbeat differ. The same BWV 856 performance is retained here to demonstrate a different view of the same case: beat and downbeat activations and discrete estimates are composed around the score on the shared time axis.

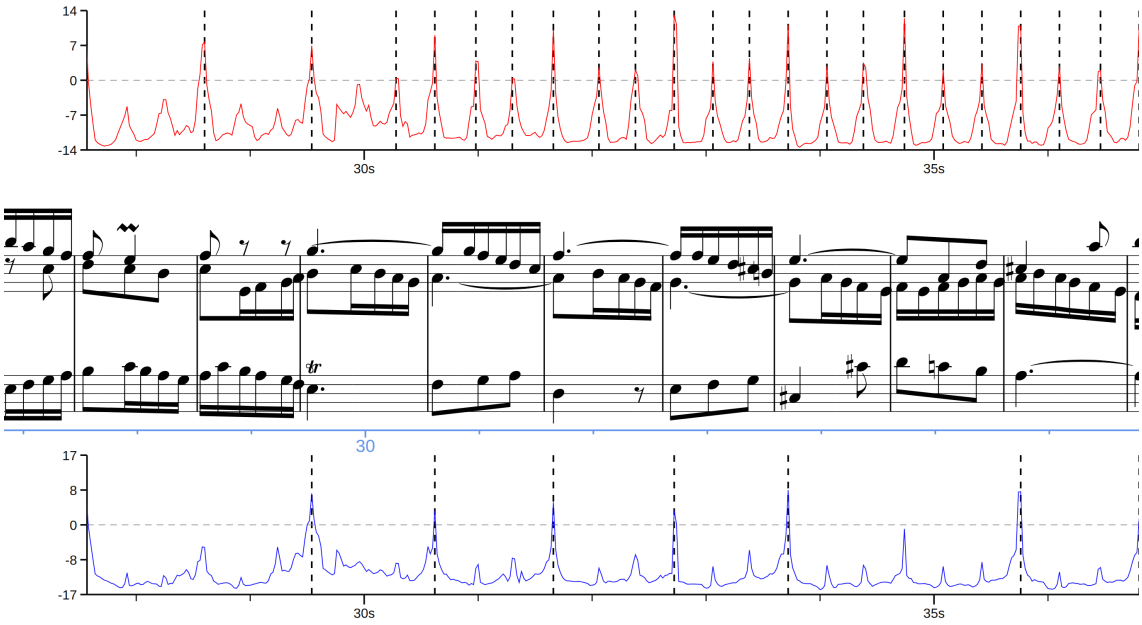


Figure 5.3: LayerIt activation-and-position display for BWV 856. The upper panel shows BeatThis beat logits as a red curve, with beat estimates overlaid as vertical black dotted lines; the time-warped score occupies the middle panel; and the lower panel shows downbeat logits as a blue curve, with estimated downbeat positions overlaid as vertical black dotted lines.

5.2.4 Layered time-based display

The final example demonstrates the lightweight end of the composition model. It overlays a normalized waveform with BeatThis beat estimated positions, then places the warped score above it. Although it contains fewer layers than the previous examples, it uses the same composition process and preserves the same time relationship. This supports rapid inspection of how detected beats relate to both acoustic events and notated musical material. The BWV 856 excerpt was chosen for this compact view to show that the same straightforward case can also be represented with only the score, waveform, and beat estimates.

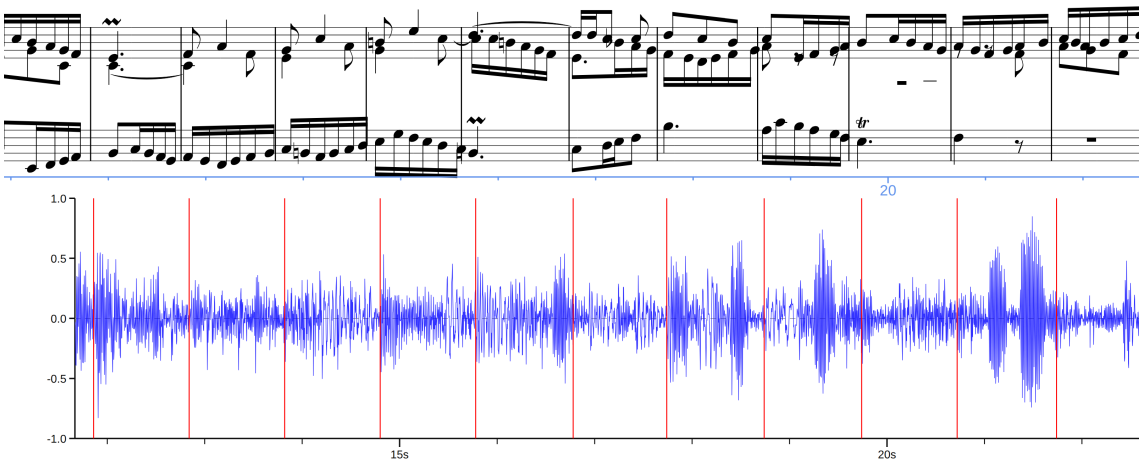


Figure 5.4: LayerIt display of an excerpt from BWV 856 performed by András Schiff. The time-warped score is shown above the waveform, with BeatThis beat estimates overlaid as red vertical lines aligned with the score.

Table 5.1: Composite figure types reproduced with LayerIt.

Figure type	**Motivating context**	**LayerIt layers**	**Composition result**
Score, audio, and chroma	Multimodal music representations [22]	Warped score, waveform, onset markers, mel spectrogram, chromagram	Multiple audio representations aligned to the score
Score-aligned BT analysis	BT evaluation [32]	Warped score, mel spectrogram, waveform, beat and downbeat positions	Musical context, acoustic evidence, and algorithmic output in one view
Activation and position display	Qualitative inspection of BT output	Beat and downbeat logits, selected positions, warped score	Continuous activations and discrete output compared on one axis
Layered time-based display	Compact BT inspection	Warped score, waveform, beat positions	A minimal score-aligned analytical display

Taken together, these examples answer RQ1 at the level of framework capability: the layer model can compose heterogeneous visualizations into a common score-aligned representation. The evidence is a set of independently re-runnable scripts rather than a manually assembled figure. The examples do not establish universal visual design rules; they show that the required figure types are expressible by the framework.

5.3 Beat-tracking analysis of *Clair de Lune*

The previous examples demonstrate the kinds of compositions that LayerIt can generate. This case study uses the same shared-axis representation to interpret a beat-tracking result in the more expressive performance introduced in Section 5.2.1. It covers the first six bars of Debussy's *Clair de Lune*, performed by Maria João Pires, using the MEI score aligned to the recording and beat and downbeat estimates produced by BeatThis.

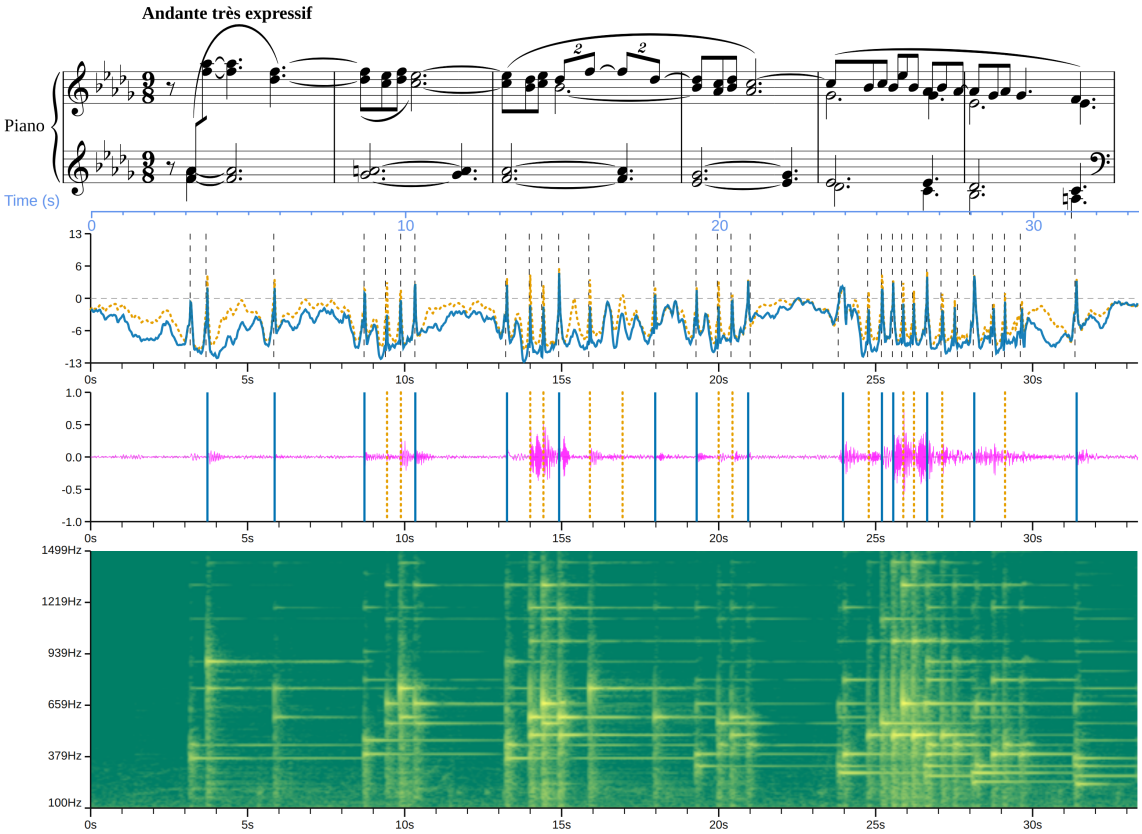


Figure 5.5: LayerIt composition for the first six bars of Debussy's *Clair de Lune*, performed by Maria João Pires. From top to bottom: the time-warped score; beat and downbeat logits with onsets shown as black dashed lines; the waveform in pink with beat and downbeat estimates; and a mel spectrogram. Beats are shown as dotted orange markers and downbeats as solid blue markers.

The six measures shown in the figure contain 18 notated beats, of which six are downbeats. Because the score has been warped onto the performance timeline, these notated positions provide a reference against which the tracker output can be inspected. The figure contains 28 beat estimates, including 15 estimates that the model labels as downbeats. Of the 28 beat estimates, 13 are positioned correctly, judged visually by whether each estimate coincides with a notated beat in the time-warped score; of the 15 downbeat estimates, five are correctly positioned downbeats, coinciding with notated downbeat positions. It is important to note that every downbeat is necessarily also a beat: a beat may or may not be classified as a downbeat, but a downbeat cannot be correct

unless its underlying beat position is also correct. This relationship is also visible in the beat and downbeat logits panel: each downbeat peak is accompanied by a corresponding peak in the beat curve. In the panel showing the notated beats, this overlap is not visually distinguishable because LayerIt deliberately draws downbeats on top of beats. The underlying SVG structure nevertheless preserves both layers, with the beat layer drawn beneath the downbeat layer.

The distribution of these estimates becomes clearer when the passage is examined measure by measure. In the first measure, BeatThis produces two beat estimates, both labelled as downbeats, but only the second estimate coincides with a notated beat. The missed first downbeat is understandable in this case: the piece begins with a quaver rest on the first downbeat, and BeatThis analyses the audio without access to the score or any other symbolic input. Nevertheless, the algorithm produces a small peak in the position of the first downbeat. In the second measure, four beats are estimated, two of which are correctly positioned; one of the two downbeat labels is correct and one is incorrect. The third measure contains seven beat estimates, three correctly positioned, with one correct and two incorrect downbeat labels. In the fourth measure, four beats are estimated, two correctly positioned, again with one correct and one incorrect downbeat label. The fifth measure contains the densest cluster of estimates: eight beats are produced, three correctly positioned, while one downbeat label is correct and three are incorrect. Finally, the sixth measure contains three beat estimates, two correctly positioned, with one correct and one incorrect downbeat label.

This pattern shows that the principal problem is not simply a small temporal displacement of otherwise correct beats. Several quaver onsets are treated as beats, and the resulting downbeat labels impose an incorrect metrical interpretation on parts of the passage. Two conditions known to trouble beat trackers hold here: compound triple metre, given its scarcity in training datasets, and highly expressive interpretations [28]. Both are apparent in the figure: compound triple metre in its 9/8 time signature, and expressive playing in its *très expressif* marking. The performer stretches and compresses the beat, as the warped score's spacing shows.

Read against the warped score, the estimates become an account of the difficulties behind them rather than a list of errors. Each false positive carries a musical address: the measure in which it occurs and the figure that produced it. The corresponding passage can therefore be located in the score and heard in the recording. A signal-derived view shows only where the estimates fall; the shared-axis composition also shows their relationship to the score, logits, waveform, and spectrogram.

5.4 Workflow Capabilities for Annotation and Evaluation (RQ2, RQ3)

This section positions LayerIt alongside Sonic Visualiser, Piano Precision, and manual SVG editing in relation to the workflows that motivated the project. It concerns the production and inspection of score-aligned figures, rather than a complete feature comparison of the other tools. Sonic Visualiser and Piano Precision remain purpose-built interactive applications, whereas LayerIt is a scriptable framework for generating static, structured SVG images.

5.4.1 Evaluation workflow

For BT evaluation, the principal advantage of LayerIt is that the algorithm's discrete positions and continuous activation functions can be read against the same time-warped score. Figure 5.3 illustrates this combination: the reader can inspect peaks in the beat and downbeat activations, the positions selected by the algorithm, and the corresponding notation without translating between separate time references.

This supports qualitative diagnosis. For example, a selected beat can be compared with a nearby activation peak and with the score event at the same horizontal position. Conversely, an activation peak that does not result in a selected position can be identified and interpreted in musical context. Sonic Visualiser supports rich audio inspection, but the workflow considered here requires an additional score-aligned composition. Piano Precision provides score-performance links for navigation and analysis [15]; however, its link-based score view does not provide the continuously time-warped score layer required by these static composite figures.

5.4.2 Annotation workflow

LayerIt is not an interactive annotation environment and does not itself let an annotator place or correct beat times. Its contribution to annotation is instead a visual reference for an upstream annotation workflow. An annotator can inspect the alignment between the spectrogram or waveform, onset markers, candidate beat positions, and the time-warped score; corrections are then made in the relevant annotation or MAPS source and the figure is regenerated.

In this workflow, the score is physically positioned on the same axis as the data to be inspected. This differs from a link-based score display, where correspondence is expressed through navigation or connections between separately laid out views. For tasks that require deciding whether a candidate beat coincides with a notated event, the shared-axis view provides direct visual context. It should nevertheless be understood as a diagnostic and communication aid, not as evidence that LayerIt improves annotation speed or accuracy; such a claim would require an empirical user study.

Table 5.2: Workflow characteristics relevant to the score-aligned visualization task considered in this thesis.

Dimension	**Sonic Visualiser**	**Piano Precision**	**Manual SVG editing**	**LayerIt**
Shared score time axis in the workflow considered	Not provided	Link-based score view	Possible through manual placement	Provided through ScoreWarp
Arbitrary algorithmic curves	Limited by available layers/plug-ins	Limited by available layers/plug-ins	Possible through manual construction	Provided through Layer subclasses
Scriptable figure generation	Not a supported workflow	Not possible	Depends on a separately maintained procedure	Provided through the composition script
Regeneration after data change	Manual reconfiguration	Manual view recreation	Repeat manual editing	Re-run with updated inputs
Adding a visualization type	Plugin development	Plugin development	Manual construction	Implement or configure a Layer

No controlled count of manual operations or wall-clock time was recorded for the workflows considered here. Consequently, this thesis does not report numerical claims about time saved by LayerIt. The evidence for RQ2 is functional rather than comparative: once the inputs are prepared, the supplied scripts generate the demonstrated figures without manual image alignment, and the same scripts can be re-run after changes to the data or visual parameters. A future study should measure one-time setup and per-figure effort separately.

5.5 Modular Composition Evidence (RQ2)

The architecture described in chapter 4 defines a common `Layer` interface and composes layers through `Visualizer`. The examples in this chapter exercise that interface with several independent layer types including: spectrogram, chromagram, waveform, onsets layer, BT output layers, beat/downbeat activation curves. Each is added through the same `add_panel()` operation and rendered through the common `compose()` workflow.

The scripts provide practical evidence of composability: Figure 5.1 combines four different representation types, while Figure 5.3 combines separate beat and downbeat activation and event layers around the same score. These substitutions require no change to the composition mechanism, which is consistent with the modular claim of Chapter 4.

The `OnsetNovelty` example was chosen specifically to demonstrate LayerIt's extensibility: it adds a new time-based visualization as a `Curve` subclass in a standalone module. The class contains approximately 60 lines of Python code. In it, the layer loads an audio file, computes its onset-strength envelope with `librosa`, converts the frame positions to seconds, and supplies the resulting curve to the inherited rendering machinery. The implementation provides the required data and rendering hooks through `load_data()`, `_get_xy()`, and a small `draw()` override for axis presentation and limits.

Nothing outside the new class was changed in the LayerIt framework or in any existing layer implementation: `Visualizer`, the base `Layer` and shape classes, and the existing audio and beat layers remained unchanged. Two supporting additions were made only to expose and demonstrate the class: its public export in `__init__.py`and the separate `OnsetNoveltyDemo.py` example script. The result is shown in 5.6.

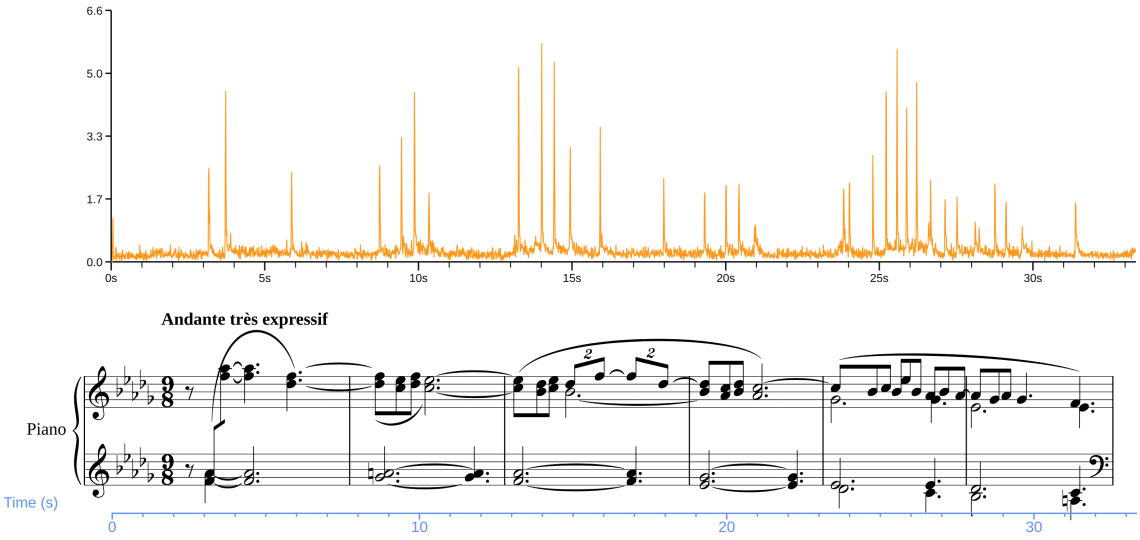


Figure 5.6: Demonstration of the new `OnsetNovelty` layer composed through LayerIt's existing workflow. The onset-strength curve is rendered as a time-based layer for the *Clair de Lune* example.

This probe provides focused evidence that at least one new time-based visualization type can be added without modifying the composition mechanism or the core layer and shape abstractions. It does not establish that all future layer types will require the same implementation effort, nor that broader extensions can be made without changes to package exports or example scripts.

5.6 Limitations and Difficult Cases (RQ2, RQ3)

The examples in this chapter depend on a reasonably faithful correspondence between the performance and the score. LayerIt composes and scales visual layers against the ScoreWarp timeline, so the fidelity of the composite visualization is bounded by the quality and density of the underlying alignment data. This is an important distinction: a misleading score position caused by an inaccurate MAPS alignment is not a failure of the layer-composition mechanism, but the final figure nevertheless inherits that limitation.

5.6.1 Expressive performances

Highly expressive performances, including substantial rubato, skipped material, repetitions, or departures from the notated score, can make the relationship between score position and performance

time difficult to represent with a simple warped timeline. Structural discontinuities such as repeats and jumps are a recognized challenge for score-performance synchronization [10, 35]. In such cases, a score element may be placed at a time that does not adequately describe the local musical event, and the misalignment then propagates visually to every layer that uses the same reference.

No dedicated expressive-performance stress test was completed for this thesis. The present evidence therefore establishes this as a boundary of the approach rather than measuring a specific degradation pattern. A suitable follow-up validation would compare dense and manually verified MAPS annotations for an expressive performance with a less expressive reference performance, then report where the warped score remains informative and where it ceases to be reliable.

5.6.2 Sparse onset annotation

Sparse MAPS annotations present a related limitation. ScoreWarp can interpolate between available anchors, reducing the human effort required to prepare a score-performance alignment. However, the temporal position between widely separated anchors is then an interpolation rather than a directly supported observation. The resulting score placement may be sufficient for an overview, but it is less appropriate for detailed beat-level inspection in passages with tempo variation.

Figure 5.7 reports this comparison for an excerpt of BWV 865 performed by András Schiff. This excerpt was chosen because the difference between its sparse and dense annotation is particularly clear in the resulting figure. The upper pair of panels uses a sparse MAPS file that retains the first and last onsets and every fiftieth onset between them. The lower pair uses the complete onset file. In both cases, the red vertical lines show the BeatThis beat estimates, while the black dotted lines show the onsets retained by the corresponding MAPS file.

The sparse alignment preserves the broad placement of the excerpt and remains most useful near the retained anchors. Between anchors, however, the score position is determined by interpolation rather than by a directly observed onset, so local spacing becomes less reliable. The dense alignment supplies onset references throughout the passage and keeps the score and waveform correspondence more stable, including around passages with several closely spaced notes. This comparison supports the expected trade-off: sparse anchors reduce preparation effort but provide weaker evidence for detailed beat-level inspection, whereas dense anchors support more reliable local placement at the cost of additional annotation data.

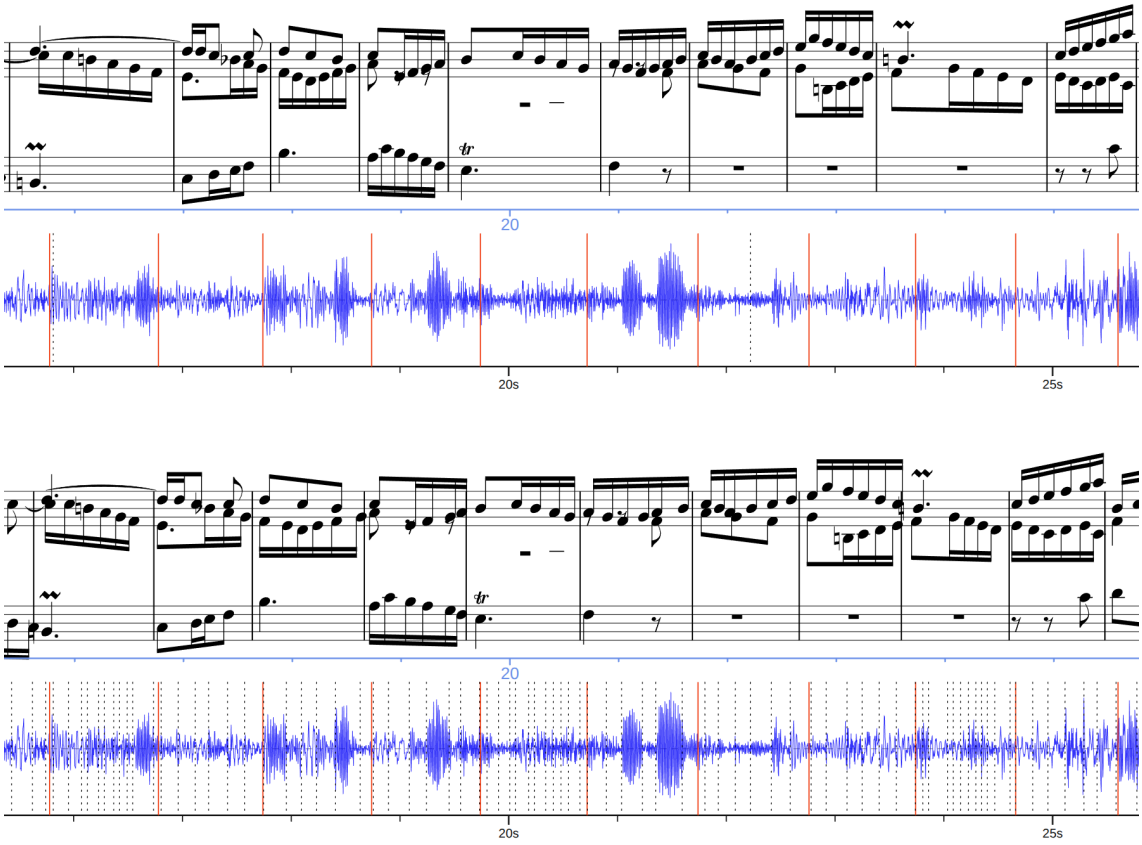


Figure 5.7: Comparison of sparse and dense score–performance alignment for an excerpt of BWV 865 performed by András Schiff. From top to bottom: score warped using sparse onsets, waveform with BeatThis beat estimates in red and retained sparse onsets as black dotted lines, score warped using all onsets, and the same waveform with the complete onset set. The sparse MAPS file retains the first and last onsets and every fiftieth onset between them.

5.6.3 Manual editing of the final SVG

Manual editing of a final SVG is not required for visual customization in LayerIt. The framework exposes customization through its layer configuration and composition code, allowing users to adjust existing layers and incorporate the visual elements required for a particular analysis while retaining a re-runnable generation script. It can also include externally prepared PNG and SVG assets, allowing a composite visualization to combine LayerIt-generated layers with other image-based material.

When an existing layer does not provide the required behaviour, the OO architecture supports modifying or extending the codebase by implementing a new `Layer` or adapting an existing one. This keeps the visual logic within the framework and preserves traceability from the source data and configuration to the resulting figure. Direct hand-editing of the exported SVG remains possible as a fallback, but it should be avoided when the adjustment can instead be represented in code, since unrecorded manual changes weaken reproducibility. If such editing is necessary, the

modified SVG and the reason for the modification should be retained alongside the generation script.

5.6.4 Rendering and representation fidelity

The fidelity of the final visualization is also bounded by the rendering pipeline. ScoreWarp depends on the quality and density of the MAPS annotations, the structure of the MEI score, and the expressiveness of the performance. Closely spaced onsets can cause score elements to overlap in the warped score, while complex MEI structures can expose limitations in the way Verovio renders the notation. In addition, the same SVG score may not be displayed identically by every application. In particular, a score that appears correctly in a browser may contain missing or incorrectly positioned elements when opened or imported into an image editor. This observation concerns compatibility between SVG producers and renderers or importers, rather than the combination of the score with LayerIt panels. The present work does not investigate the technical causes of these differences; it records them as a practical limitation of relying on SVG for the score representation. Nevertheless, they remain relevant to RQ3 because an occluded or visually ambiguous score can reduce the usefulness of the shared-axis visualization for interpreting beat-tracking and annotation data.

5.7 Summary

This chapter provides a qualitative answer to the three research questions. For RQ1, the four main figures demonstrate that LayerIt can compose symbolic score material, audio representations, onset annotations, beat and downbeat activations, and BT output into common score-aligned views. For RQ2, the generated examples and composition workflow show that existing layers can be created, modified, exchanged, and regenerated through a common scriptable interface; the `OnsetNovelty` probe additionally provides focused evidence that a new time-based layer can use that interface without changes to the composition core.

For RQ3, the examples show how a shared-axis visualization can support qualitative inspection of BT results and can serve as a diagnostic reference for annotation. LayerIt is not itself an interactive annotation tool, and no user-study claim is made. Its usefulness is conditioned by the quality of the available score-performance alignment and by the fidelity of the resulting rendered score. Expressive performances, sparse annotation anchors, and rendering limitations therefore remain important boundaries that require dedicated empirical evaluation.

Chapter 6

Conclusions and Future Work

6.1 Achievements

This thesis makes two main contributions:

1. **Software contribution:** LayerIt, an open-source, pip-installable Python framework for composing score-aligned MIR visualizations through an OO layer-composition model. Its official repository, published with the LBD, is `https://github.com/FernandoAze/LayerIt`; the visual examples presented in this thesis were generated using the development repository at `https://github.com/FernandoAze/LayerIt_WIP`. The framework provides a reusable and extensible architecture for composing independent visual layers.

2. **Research contribution:** a practical approach to integrating complementary musical representations expressed in different time domains. By building on score–performance alignment and the score warping provided by ScoreWarp [11], LayerIt enables symbolic music notation, expressed in abstract (musical) time, to be visualized alongside audio-derived representations and algorithmic outputs, expressed in physical (performance) time (Section 2.3). This allows scores, waveforms, spectrograms, annotations, and beat-tracking results to be inspected on a shared time axis, providing musical context for the analysis and interpretation of temporal information.

Revisited against the objectives and research questions set out in Chapter 1, the work can be summarized as follows:

- **Objective 1** was met through the development of LayerIt, a Python OO framework in which score, audio, and algorithmic data are treated as independent, composable layers. chapter 4 showed that this "layers within layers" philosophy holds consistently at the visual, SVG, and code levels, giving the framework a coherent architecture rather than a single-purpose script.

- **Objective 2** was met through the generation and regeneration of four distinct composite figure types and the implementation of one new time-based layer. The figure types include

a *Clair de Lune* view combining a warped score, waveform, mel spectrogram, and chromagram; BWV 856 views combining score, spectrogram, waveform, beat and downbeat events, and activation curves; and a compact score-and-waveform display. The additional `OnsetNovelty` probe was implemented as a `Curve` subclass and composed through the existing `Visualizer` workflow without changes to the composition core or existing layer modules. The same composition mechanism creates the figures, and the demonstrated layers can be modified or exchanged through the common Layer and Visualizer interfaces. This provides functional evidence for RQ2 without making comparative claims about development time or user effort.

- **Objective 3** was to apply LayerIt to beat-tracking case studies and to document the limits of the resulting visualizations. The BWV 856 and *Clair de Lune* case studies show that placing the score, audio representations, tracker activations, and estimated events on one performance-time axis gives these data a shared musical context. This makes it possible to inspect disagreements between the tracker and the notated metre, and to relate an estimated event to both the recording and the corresponding score passage. The result is a visual diagnostic aid rather than an annotation tool or a beat-tracking evaluation by itself: its usefulness depends on the quality of the score-performance alignment, especially for expressive performances and sparse onset annotations.

The work also resulted in a Late-Breaking Demo paper accepted for presentation at the 2026 International Society for Music Information Retrieval Conference (ISMIR 2026), the main international conference in the field [1]. This acceptance is an initial sign of the work's relevance to the MIR research community, and it offers an opportunity to present LayerIt to researchers in the field, gather feedback, and encourage its adoption and further development.

Beyond these contributions, the iterative development process described in Chapter 3 highlighted an important distinction between the visualization and the interactive annotation of musical information. While this thesis focused on visualization, interactive annotation remains an open research direction, particularly for time-related MIR tasks such as beat tracking and onset detection. The ability to combine symbolic and audio-based representations within a common temporal reference provides a foundation for future work on annotation tools that integrate musical structure with performance-time information.

6.2 Future Work

The limitations identified in chapter 5 point directly to the most pressing follow-up work. First, the `OnsetNovelty` probe demonstrates one extension path, but it does not establish the implementation effort or integration requirements for a broader range of layer types. Future work should repeat the probe with visualizations based on different shapes, data formats, and external contributors, while recording the files and changes required in each case. Second, the fidelity

of LayerIt's visualizations is bounded by the quality of the underlying score-performance alignment; dedicated experiments comparing dense and sparse MAPS annotations, and stress-testing the framework against expressive performances with substantial rubato or structural deviations from the score, would clarify precisely where the time-warped representation remains reliable. A study with target users, such as beat-tracking researchers and annotators, would assess whether score-aligned figures support annotation and evaluation tasks in practice, a claim this thesis does not make (Section 5.4.2).

Beyond validating what already exists, several directions could extend LayerIt's scope. The framework currently integrates a single alignment method (ScoreWarp) and a single BT algorithm (BeatThis). A synchronization toolkit such as Sync Toolbox could provide additional upstream score-performance alignments [23], while support for an interchange representation such as the match file format could make these inputs more portable [9]. Supporting these and other alignment and MIR algorithms as further `Layer` implementations would test the framework under more demanding conditions and grow LayerIt into a more general-purpose visualization toolkit. Validating the framework against a broader range of instrumentations, genres, and performance styles would strengthen the generality of its conclusions. Finally, having deliberately set aside the interaction problem in chapter 3, a natural long-term direction is to revisit it: the SVG output produced by LayerIt already preserves the addressable structure inherited from Verovio and MEI, which could be leveraged to build an interactive, browser-based annotation layer on top of the visualizations this thesis already knows how to generate. Pursuing this direction would reconnect the visualization contribution made here with the beat-annotation and BT-evaluation tools that originally motivated this thesis.

The output format also suggests two technical extensions. First, accepting richer alignment inputs with provenance and domain information would make it easier to distinguish directly observed anchors from interpolated score positions. Second, dense time-frequency and pitch fields are currently embedded as raster images inside otherwise structured SVG output. This preserves the visual result but limits post-export restyling compared with the vector score and event layers. Future work should investigate metadata-preserving field layers or controlled vector representations where the resulting file size and rendering cost remain practical.

6.3 Final Remarks

This thesis began with the observation that producing time-aligned, multi-modal visualizations can require the manual combination of separately generated score, audio, and algorithmic representations. Through an iterative development process and careful architectural design, we developed LayerIt, a Python-based OO tool for composing these representations as structured, score-aligned visualizations.

LayerIt's value lies in its demonstrated ability to compose visual layers into re-runnable, score-aligned figures for BT evaluation and beat annotation inspection. The score remains identifiable through the MEI and Verovio-derived SVG structure, while added components remain separate

and restylable where they are represented as vector elements. By grounding the system in established conventions, including OO design patterns, common MIR libraries, and standardized formats, the framework provides a basis for future extension and integration.

As MIR research continues to evolve through advances in machine learning, increasing dataset complexity, and growing interdisciplinary collaboration, the need for flexible visualization tools will continue to grow. LayerIt provides a focused foundation for future work on composable score-aligned visualizations.

References

[1] Fernando Azeredo and António Sá Pinto. “LayerIt: Towards a Framework for Time-Aligned, Composable Music Visualizations”. Accepted for presentation at ISMIR 2026 in the Late-Breaking Demo track. 2026.

[2] Barry W. Boehm. “Verifying and Validating Software Requirements and Design Specifications”. In: *IEEE Software* 1.1 (1984), pp. 75–88. DOI: 10.1109/MS.1984.233702.

[3] Chris Cannam, Christian Landone, and Mark Sandler. “Sonic Visualiser: An Open Source Application for Viewing, Analysing, and Annotating Music Audio Files”. In: *Proceedings of the 18th ACM International Conference on Multimedia (MM ’10)*. Florence, Italy: ACM Press, 2010, pp. 1467–1468. DOI: 10.1145/1873951.1874248.

[4] Julio José Carabias-Orti et al. “An Audio to Score Alignment Framework Using Spectral Factorization and Dynamic Time Warping.” In: (2015), pp. 742–748. URL: https://api.semanticscholar.org/CorpusID:18255173.

[5] Arshia Cont et al. “Evaluation of Real-Time Audio-to-Score Alignment”. In: *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR)*. Vienna, Austria, 2007.

[6] Design Council. *Framework for Innovation: Design Council’s Evolved Double Diamond*. 2019. URL: https://www.designcouncil.org.uk/our-resources/framework-for-innovation/.

[7] Zhiyao Duan and Bryan Pardo. “Soundprism: An Online System for Score-Informed Source Separation of Music Audio”. In: *IEEE Journal of Selected Topics in Signal Processing* 5.6 (Oct. 2011), pp. 1205–1215. ISSN: 1932-4553, 1941-0484. DOI: 10.1109/JSTSP.2011.2159701. URL: http://ieeexplore.ieee.org/document/5887382/.

[8] Francesco Foscarin, Jan Schlüter, and Gerhard Widmer. “Beat This! Accurate Beat Tracking without DBN Postprocessing”. In: *Proceedings of the 25th International Society for Music Information Retrieval Conference (ISMIR)*. San Francisco, CA, USA, 2024. DOI: 10.48550/arXiv.2407.21658.

[9] Francesco Foscarin et al. “The Match File Format: Encoding Alignments between Scores and Performances”. In: *Proceedings of the Music Encoding Conference*. Halifax, NS, Canada, 2022, pp. 52–60. DOI: 10.48550/arXiv.2206.01104.

[10] Christian Fremerey, Meinard Müller, and Michael Clausen. “Handling Repeats and Jumps in Score-Performance Synchronization”. In: *Proceedings of the 11th International Society for Music Information Retrieval Conference*. 2010, pp. 243–248.

[11] Werner Goebl et al. “Let’s Do the ScoreWarp Again! Shifting Notes to Performance Timelines”. In: *Proceedings of the Music Encoding Conference*. London, United Kingdom, 2025. DOI: 10.17613/gqd3b-w7c28.

[12] Saul Greenberg and Bill Buxton. "Usability Evaluation Considered Harmful (Some of the Time)". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*. 2008, pp. 111–120. DOI: 10.1145/1357054.1357074.

[13] Johannes Hentschel et al. "Time to Align! Modelling Musical Timelines for Music Information Retrieval and Digital Musicology". In: *Transactions of the International Society for Music Information Retrieval* 9.1 (2026), pp. 384–404. DOI: 10.5334/tismir.296.

[14] Alan R. Hevner et al. "Design Science in Information Systems Research". In: *MIS Quarterly* 28.1 (2004), pp. 75–106. DOI: 10.2307/25148625.

[15] Yucong Jiang, David M. Weigl, and Werner Goebl. "Piano Precision: Advancing Music Performance Analysis by Integrating User-Correctable Audio-to-Score Alignment". In: *Proceedings of the 22nd Sound and Music Computing Conference (SMC)*. Graz, Austria, July 2025, pp. 160–167. DOI: 10.5281/zenodo.15838699. URL: https://doi.org/10.5281/zenodo.15838699.

[16] Jongmin Jung et al. "Unified Cross-modal Translation of Score Images, Symbolic Music, and Performance Audio". In: *IEEE Transactions on Audio, Speech and Language Processing* 34 (2026), pp. 1876–1891. arXiv: 2505.12863 [cs.SD]. URL: http://arxiv.org/abs/2505.12863.

[17] Zulfidin Khodzhaev. "A practical guide to spectrogram analysis for audio signal processing". In: *arXiv preprint arXiv:2403.09321* (2024).

[18] R. Khulusi et al. "A Survey on Visualizations for Musical Data". In: *Computer Graphics Forum* 39.6 (2020), pp. 82–110. DOI: 10.1111/cgf.13905.

[19] David Ledo et al. "Evaluation Strategies for HCI Toolkit Research". In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 2018, pp. 1–17. DOI: 10.1145/3173574.3173610.

[20] Hugo B. Lima, Carlos G. R. dos Santos, and Bianchi S. Meiguins. "A Survey of Music Visualization Techniques". In: *ACM Computing Surveys* 54.7, 143 (2021). DOI: 10.1145/3461835.

[21] Brian McFee et al. "Librosa: Audio and Music Signal Analysis in Python". In: *Proceedings of the 14th Python in Science Conference*. Austin, Texas, USA, 2015, pp. 18–25.

[22] Meinard Müller. *Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications*. Vol. 5. Springer, 2015. ISBN: 978-3-319-21944-8.

[23] Meinard Müller et al. "Sync Toolbox: A Python Package for Efficient, Robust, and Accurate Music Synchronization". In: *Journal of Open Source Software* 6.64 (2021), p. 3434. DOI: 10.21105/joss.03434.

[24] Music Encoding Initiative. *An Introduction to MEI*. 2026. URL: https://music-encoding.org/about/ (visited on 09/11/2026).

[25] Eita Nakamura, Kazuyoshi Yoshii, and Haruhiro Katayose. "Performance Error Detection and Post-Processing for Fast and Accurate Symbolic Music Alignment". In: *Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR)*. Suzhou, China, 2017, pp. 347–353.

[26] Dan R. Olsen Jr. "Evaluating User Interface Systems Research". In: *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology (UIST '07)*. 2007, pp. 251–258. DOI: 10.1145/1294211.1294256.

[27] Silvan David Peter et al. “Automatic Note-Level Score-to-Performance Alignments in the ASAP Dataset”. In: *Transactions of the International Society for Music Information Retrieval* 6.1 (June 26, 2023), pp. 27–42. ISSN: 2514-3298. DOI: 10.5334/tismir.149. URL: http://transactions.ismir.net/articles/10.5334/tismir.149/.

[28] António Sá Pinto and Matthew E. P. Davies. “Tapping Along to the Difficult Ones: Leveraging User-Input for Beat Tracking in Highly Expressive Musical Content”. In: *Perception, Representations, Image, Sound, Music – 14th International Symposium, CMMR 2019, Marseille, France, October 14–18, 2019, Revised Selected Papers*. Ed. by Richard Kronland-Martinet, Sølvi Ystad, and Mitsuko Aramaki. Vol. 12631. Lecture Notes in Computer Science. Springer, 2021, pp. 75–90. ISBN: 978-3-030-70209-0. DOI: 10.1007/978-3-030-70210-6_5.

[29] Laurent Pugin, Rodolfo Zitellini, and Perry Roland. “Verovio: A Library for Engraving MEI Music Notation into SVG”. In: *Proceedings of the 15th International Society for Music Information Retrieval Conference (ISMIR)*. Taipei, Taiwan, 2014, pp. 107–112.

[30] Colin Raffel et al. “mir_eval: A Transparent Implementation of Common MIR Metrics”. In: *Proceedings of the 15th International Society for Music Information Retrieval Conference*. 2014.

[31] Répertoire International des Sources Musicales. *Verovio Reference Book*. 2021. DOI: 10.5448/7EM6-MY23. URL: https://book.verovio.org/. Pre-published.

[32] António Sá Pinto, Inês Domingues, and Matthew E. P. Davies. “Shift If You Can: Counting and Visualising Correction Operations for Beat Tracking Evaluation”. In: *Late-Breaking Demo Session of the 21st International Society for Music Information Retrieval Conference*. 2020. DOI: 10.48550/arXiv.2011.01637.

[33] Mary Shaw. “What Makes Good Research in Software Engineering?” In: *International Journal on Software Tools for Technology Transfer* 4.1 (2002), pp. 1–7. DOI: 10.1007/s10009-002-0083-4.

[34] The Matplotlib Development Team. *Matplotlib: Visualization with Python*. Version v3.10.9. Zenodo, Apr. 24, 2026. DOI: 10.5281/ZENODO.19716234. URL: https://zenodo.org/doi/10.5281/zenodo.19716234.

[35] Verena Thomas et al. “Linking Sheet Music and Audio – Challenges and New Approaches”. In: *Multimodal Music Processing*. Vol. 3. Dagstuhl Follow-Ups. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012, pp. 1–22. DOI: 10.4230/DFU.Vol3.11041.1. URL: https://doi.org/10.4230/DFU.Vol3.11041.1.

[36] David M. Weigl. *Multimodal Music Information Alignment v2*. Tech. rep. Deliverable 3.5. TROMPA, 2020.

[37] Roel J. Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. Springer, 2014. DOI: 10.1007/978-3-662-43839-8.

Statement on the Use of Generative AI

Generative artificial intelligence was used as an assistive tool during the preparation of this thesis and the development of the LayerIt codebase. The principal tool used was GitHub Copilot, including its conversational and code-assistance features.

GitHub Copilot was used for the writing of this thesis in order to improve grammar, clarity, and concision. The tool was not treated as an academic source, and the responsibility for the interpretation of the literature, the accuracy of the claims, the selection and verification of citations, and the final wording remains with the author.

For the LayerIt codebase, GitHub Copilot was used to suggest implementation approaches, code fragments, refactorings, debugging steps, and documentation. The author defined the requirements and architecture, evaluated the suggestions, integrated or rejected them as appropriate, and remained responsible for the resulting code and its behaviour. The figures, experiments, and conclusions presented in this thesis were selected and interpreted by the author.

Appendix A

Additional Code Listing

This appendix provides the lower-level rendering and SVG assembly workflow summarized in Chapter 4. The public composition script calls these operations through `Visualizer.compose()`.

Listing A.1: Internal SVG composition workflow

```
# Internal workflow (called by compose())

# For each panel: render layers, export to SVG
panel_viz = Visualizer(audio=audio, score=score,
                       maps=maps, beats=beats)
panel_viz.add_panel(MelSpec(), BeatLogits())
panel_viz.load_all_layers(audio_path=audio,
                          beat_file=beats)
fig, ax = panel_viz.draw()
panel_svg = panel_viz.turn_to_SVG(
  filename="panel.svg",
  svg_warped_score=score)

# Stack panels with score using create_final_SVG()
composer = Visualizer(audio=audio, score=score,
                      maps=maps, beats=beats)
composer.create_final_SVG(
  svg_layers=[(panel_svg, 50),
              (score_svg, 300)],
  output_file="final.svg")
```

Each panel is rendered to SVG through `turn_to_SVG()`, preserving its layers as named SVG groups. The resulting panel SVGs are then stacked with the warped score by `create_final_SVG()`, with vertical offsets controlling their positions while the shared timeline preserves horizontal alignment.