

Atrium: Lowering the Barrier to Multimodal Context-Sensitive Applications

Bachelor's Thesis at the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

by
Anders Petruschke

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Prof. Dr.-Ing. Robert Schmitt

Registration date: 10.07.2026
Submission date: 29.09.2026

Eidesstattliche Versicherung Declaration of Academic Integrity

Petruschke, Anders
Name, Vorname/Last Name, First Name

381030
Matrikelnummer (freiwillige Angabe)
Student ID Number (optional)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende ~~Arbeit~~/Bachelorarbeit/
~~Masterarbeit~~* mit dem Titel
I hereby declare under penalty of perjury that I have completed the present ~~paper~~/bachelor's thesis/~~master's thesis~~* entitled

Atrium: Lowering the Barrier to Multimodal Context-Sensitive Applications

selbstständig und ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting) erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt; dies umfasst insbesondere auch Software und Dienste zur Sprach-, Text- und Medienproduktion. Ich erkläre, dass für den Fall, dass die Arbeit in unterschiedlichen Formen eingereicht wird (z.B. elektronisch, gedruckt, geplottet, auf einem Datenträger) alle eingereichten Versionen vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

independently and without unauthorized assistance from third parties (in particular academic ghostwriting, I have not used any other sources or aids than those indicated; this includes in particular software and services for language, text, and media production. In the event that the work is submitted in different formats (e.g. electronically, printed, plotted, on a data carrier), I declare that all the submitted versions are fully identical. I have not previously submitted this work, either in the same or a similar form to an examination body.

Aachen, 26.09.26

Ort, Datum/City, Date

Anders Petruschke

Unterschrift/Signature

*Nichtzutreffendes bitte streichen/Please delete as appropriate

Belehrung: Official Notification:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 156 StGB (German Criminal Code): False Unsworn Declarations

Whosoever before a public authority competent to administer unsworn declarations (including Declarations of Academic Integrity) falsely submits such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment for a term not exceeding three years or to a fine.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Strafflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

§ 161 StGB (German Criminal Code): False Unsworn Declarations Due to Negligence

(1) If an individual commits one of the offenses listed in §§ 154 to 156 due to negligence, they are liable to imprisonment for a term not exceeding one year or to a fine.

(2) The offender shall be exempt from liability if they correct their false testimony in time. The provisions of § 158 (2) and (3) shall apply accordingly.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:
I have read and understood the above official notification:

Aachen, 26.09.26

Ort, Datum/City, Date

Anders Petruschke

Unterschrift/Signature

Contents

Abstract	ix
Überblick	xi
Acknowledgments	xiii
Conventions	xv
1 Introduction	1
1.1 The Barrier	2
1.2 Approach	3
1.3 Results	4
1.4 Contributions	4
1.5 Thesis Structure	5
2 Related Work	7
2.1 Multimodal Context-Sensitive Applications	7
2.2 Toolkits and Platforms	8

3	Method	11
3.1	Research Approach	11
3.2	Development Process	12
4	Atrium	15
4.1	Design Commitments	15
4.2	Architecture Overview	16
4.3	Gateway to AI Models	18
4.4	Shared Asset Library	20
4.4.1	Tags and Concepts	23
4.5	Realtime Rooms	25
4.6	Persistent Event Log	26
4.7	Projects, Identity and Access	28
4.7.1	Projects	28
4.7.2	Identity and Access	29
4.8	Documentation and the Assistant	30
4.9	Extending Atrium	31
4.9.1	Enrichment Sources	32
4.9.2	Embedders and Embedding Spaces	32
4.9.3	Assistant Skills	33
4.9.4	New Capability Kinds	33
5	Building on Atrium	35

5.1	SoundMuse	35
5.2	Other Example Applications	38
5.3	Extraction into the SDK and Platform	39
5.4	Workbenches	40
6	Evaluation	43
6.1	Study Design	43
6.2	How Atrium Was Used	45
6.3	Participant Feedback	48
7	Discussion	51
7.1	Lowering the Barrier	51
7.2	Development Through Use	54
7.3	Limitations	56
8	Summary and Future Work	57
8.1	Summary and Contributions	57
8.2	Future Work	58
A	Study Materials	61
A.1	Interview Guide, First Meeting	61
A.2	Interview Guide, Follow-Up Meeting	63
A.3	Questionnaire, First Meeting	64
A.4	Questionnaire, Follow-Up	69

A.5 Consent Form	77
----------------------------	----

Bibliography	79
---------------------	-----------

Index	83
--------------	-----------

List of Figures and Tables

2.1	Existing tools compared	10
4.1	Web interface home page	17
4.2	Library search	22
4.3	Hosted room tools	27
5.1	SoundMuse on Atrium	36
5.2	Lines of code per area, prototype and rebuild	38
5.3	Workbenches	41
6.1	Study timeline	45
6.2	Build state and time spent	46
6.3	Parts of Atrium used	47
6.4	Without Atrium	49

Abstract

Researchers building multimodal context-sensitive applications reimplement a similar set of functions for this class of application in each new project. Access to AI models, material that can be searched, device sync and data recording stand out as core capabilities that have to exist before any research question can be asked. Atrium provides these capabilities bundled into one platform by acting as a gateway to AI models, implementing a shared asset library and giving developers access to realtime communication between devices and a persistent event log for data collection. It is accessible through a web interface or through SDKs for Python and TypeScript. Requirements for the platform were derived from real use and past projects, as well as interviews with students who had worked on similar projects and their studies. Further features were extracted from example applications that were built with Atrium.

A prototype application built at the start of this thesis was rebuilt on the Atrium platform, reducing its application code from 23,360 to 8,221 lines for comparable functionality.

In an exploratory and formative study, 11 participants used the platform for about two weeks while their feedback was continuously integrated. Five of the 6 respondents would probably or definitely not have attempted their project without Atrium. Three participants intend to keep actively using Atrium, two of them for their thesis.

By lowering the barrier to multimodal context-sensitive applications, Atrium supports future research into this class of application.

Überblick

Forschende, die multimodale kontextsensitive Anwendungen entwickeln, implementieren in jedem neuen Projekt erneut ähnliche Funktionen, die für diese Klasse von Anwendungen benötigt werden. Zugriff auf KI-Modelle, durchsuchbares Material, die Synchronisation von Geräten und die Aufzeichnung von Forschungsdaten stechen dabei als zentrale Fähigkeiten hervor, die vorhanden sein müssen, bevor überhaupt eine Forschungsfrage gestellt werden kann. Atrium bündelt diese Fähigkeiten in einer Plattform: Es dient als Schnittstelle zu KI-Modellen, stellt eine gemeinsame Asset-Bibliothek bereit und bietet Entwicklenden Echtzeitkommunikation zwischen Geräten sowie ein persistentes Ereignisprotokoll zum Sammeln von Daten. Atrium ist über eine Weboberfläche oder über SDKs für Python und TypeScript nutzbar. Die Anforderungen an die Plattform wurden aus realer Nutzung und früheren Projekten sowie aus Interviews mit Studierenden abgeleitet, die an ähnlichen Projekten und den zugehörigen Studien gearbeitet haben. Weitere Funktionen wurden aus Beispielanwendungen extrahiert, die mit Atrium entwickelt wurden.

Ein zu Beginn der Arbeit entwickelter Anwendungsprototyp wurde mit der Atrium-Plattform neu implementiert, wodurch sich der Anwendungscode bei vergleichbarer Funktionalität von 23.360 auf 8.221 Zeilen reduziert hat.

In einer explorativen und formativen Studie haben 11 Teilnehmende die Plattform für etwa zwei Wochen genutzt, während ihr Feedback fortlaufend eingearbeitet wurde. Fünf der 6 Befragten hätten ihr Projekt ohne Atrium wahrscheinlich oder sicher nicht in Angriff genommen. Drei Teilnehmende haben vor, Atrium weiterhin aktiv zu nutzen, zwei von ihnen für ihre Abschlussarbeit.

Indem Atrium die Hürde für multimodale kontextsensitive Anwendungen senkt, unterstützt es zukünftige Forschung an dieser Klasse von Anwendungen.

Acknowledgments

I would like to thank Prof. Dr. Jan Borchers and Prof. Dr.-Ing. Robert Schmitt for examining my thesis.

I would also like to thank my advisor Paul Preuschoff for introducing me to this exciting area of research and giving me the freedom to write about a topic that aligns with my interests, as well as providing invaluable guidance and feedback along the way.

Thank you to Dr.-Ing. Anas Abdelrazeq for supporting me with my thesis and throughout the years.

Also thank you to Jan Erik Karuc for quickly responding to every request for IT support, as well as all the other lovely people of i10 who were always open to answering my questions and participating in studies.

And of course, thank you to my family and friends for supporting me throughout this journey.

Conventions

Throughout this thesis we use the following conventions:

- The thesis is written in American English.
- The first person is written in plural form.
- Unidentified third persons are referred to in the plural form.

Short excursuses are set off in colored boxes.

EXCURSUS:
Excursuses are set off in orange boxes.

Where appropriate, paragraphs are summarized by one or two sentences that are positioned at the margin of the page.

This is a summary of a paragraph.

Source code and implementation symbols are written in typewriter-style text.

Code blocks are written in grey boxes.

```
client = Atrium(base_url=URL, api_key=KEY)
```


Chapter 1

Introduction

Creating applications that notice and react to the context a user is in is a hard development problem. Multiple systems have to work together from noticing to analyzing to responding. There are examples of such context-sensitive applications used in daily life, including an AI assistant that can see what is happening on the user's screen or a smart home that can detect when someone is coming home and turn the lights on.

Context-sensitive applications are hard to build

This thesis focuses on a class of context-sensitive applications that incorporate multimodality into their input or output, reacting to text, images or sound and also outputting any of these to the user, for example in the form of music that fits the current context. While this class of applications usually requires the same spine of functionality, researchers often implement it from scratch for each project.

Researchers rebuild the same spine for every project

Making it easier to develop these kinds of applications by providing a shared backend and software toolkit could reduce the development effort drastically, leaving researchers free to focus on other parts of their work. This is why the focus of this thesis is on developing a platform that can do exactly this: lowering the barrier to creating multimodal context-sensitive applications.

A shared backend could reduce this effort

1.1 The Barrier

Inspired by prior work at
the i10 chair

This thesis was inspired by the prior theses on SoundMuse, a system that analyzes an author's story and plays fitting music and ambient sounds [Gujejani, 2025] and can also analyze and display related pictures [Emir, 2025]. This was achieved by giving the input as context to a multimodal LLM and performing keyword search in online picture and sound libraries for fitting material. Multiple studies were run around SoundMuse with the goal of investigating this system's impact on the creativity of participants.

Implementation became
a barrier to research

When we spoke with students who had the task to implement, add functionality to or run studies around SoundMuse, they described several obstacles they faced during and after development. Even though they were interested in the effects of these kinds of environments on creativity, the implementation of the system itself had become a big barrier to conducting that research.

Four main goals for a
shared system

From their input during interviews and the implementation of a new SoundMuse prototype for this thesis, four main goals for a shared system were extracted from problems and barriers that were encountered.

Understanding the context This often involves different kinds of AI models and related functionality and how access to these can be gained. It can be hard to host models locally due to hardware limitations or just the difficulty of setting them up in the first place. Access to cloud models needs to be managed through their APIs, which require corresponding API keys and often bespoke implementations or adapters for different providers.

Finding fitting material This involves finding sounds and images that fit the context in question. One of the main limitations of the keyword search approach is that the results are not very accurate and heavily rely on the assets in the online databases being tagged correctly. Since most of them are made up of user-generated content, the quality can vary drastically.

Syncing devices and applications in real time Often the context being recorded does not happen on the same machine on which the reaction is supposed to take place. The screen for the image to be displayed may sit across the room, for example. Alternatively, a researcher may want to monitor what is happening on the user's screen and in the background from their own device. In the prior SoundMuse implementations, everything ran on one machine. This caused issues especially during study sessions, as what was happening in the backend became invisible to the researcher while the user used the device.

Recording research data Since the goal is to do research on how these applications are used, data naturally has to be collected during sessions with study participants. During past studies, this would include manual copying of console output or heavy note-taking during a session.

This thesis introduces Atrium as a platform to fulfill these goals.

1.2 Approach

The Atrium platform itself was implemented as a shared backend, available online through a web interface or through SDKs for Python and TypeScript. The goals were fulfilled by implementing four core capabilities: a gateway to AI models, a shared asset library, realtime rooms and a persistent event log. After building these and documenting their design, an exploratory and formative study was conducted with developers and researchers to find out how the platform would be used and to identify gaps and issues they would encounter.

During the study, participants had free access to implement their own ideas based on the provided functionality, and the platform was continuously optimized based on their feedback.

Four core capabilities
fulfill these goals

A study examined how
the platform is used

After a window of around two weeks, an additional interview and questionnaire were used to evaluate how Atrium aided and influenced their projects.

1.3 Results

Varied use during the limited timeframe, resulting in multiple applications

Every participant approached the platform differently, resulting in a variety of projects and areas on which we received feedback. Even though most participants spent less than 5 hours with the platform, multiple applications were built during that limited timeframe.

Evidence points towards Atrium lowering effort

Results show that 5 out of 6 participants who answered would probably or definitely not have attempted their project if they had not had access to the platform. Similarly, 5 of these 6 participants noted that their project would have taken them much more effort to create without Atrium.

Of the 8 participants who took part in the follow-up, 5 would recommend the platform to someone working on a similar project, 3 would probably recommend it and nobody answered anything below that. Three intend to keep actively using Atrium, two of them building their thesis on it.

Building similar applications with Atrium requires less code

The SoundMuse prototype used as the motivation was also rebuilt with Atrium and compared against the original prototype built at the start of this thesis without Atrium, which showed a reduction of around 65% in lines of application code.

1.4 Contributions

Atrium as an artifact contribution

This thesis contributes Atrium as an artifact [Wobbrock and Kientz, 2016] and evaluates it based on what it enables other users to do [Ledo et al., 2018]. Atrium implements four core capabilities to lower the barrier and remove obsta-

cles to creating multimodal context-sensitive applications, as well as additional features and research tools.

The platform itself is hosted and available to users of the i10 chair at RWTH Aachen University. It includes instructions and documentation on how it can be used.

Several example applications were built with Atrium, including a new version of SoundMuse, to serve as a guide for future developers.

1.5 Thesis Structure

This thesis starts with related work on multimodal context-sensitive applications and their precursor systems, as well as existing toolkits and platforms in the field (Chapter 2). In Chapter 3, we describe the research approach and the development process, including the sources of evidence behind the design decisions.

Chapter 4 describes the architecture and features of the platform in detail, including its design commitments and how it can be extended. We then present applications built on top of Atrium, including the rebuild of SoundMuse, and how they shaped the platform itself through extracting their features (Chapter 5).

The exploratory and formative study and its results are presented in Chapter 6. The thesis concludes with a discussion of the claim that Atrium lowers the barrier, how the platform developed through its use and the limitations of this work (Chapter 7), followed by a summary and an outlook on how the platform can be used and built upon (Chapter 8).

Chapter 2

Related Work

This chapter grounds the thesis in existing literature on context-sensitive applications and introduces precursor systems. It also gives an overview of existing platforms and frameworks that implement features similar to Atrium's, and explains what sets Atrium apart from them.

2.1 Multimodal Context-Sensitive Applications

In this thesis, context follows the definition from Dey [2001, p. 5]: “Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.”

Defining context

He also defines a system as context-aware “if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task” [Dey, 2001, p. 5].

This thesis uses the term context-sensitive application, following Cheverst et al. [1999], to mean the same class of context-aware applications.

We call an application multimodal when it senses and reacts with more than one type of data, typically text, images or sound.

SoundMuse as
precursor system

Precursor systems such as SoundMuse [Gujejani, 2025] use an author’s text input as context, in Dey’s terms, to play fitting audio. Emir [2025] extends this functionality by also allowing images as an additional modality for input and output in the application. Both theses focus on the effects of context-sensitive applications on creativity.

Other multimodal
context-sensitive
applications

The project ContextCam [Fan et al., 2024] analyzes a user’s surroundings as the context. This can include the current location and weather, their personal state such as facial expressions, or music being played. Based on that information, the user gets a choice between multiple themes to generate a fitting image, which can be refined further in a conversation with the ContextCam AI system.

Won et al. [2021] use stories as text input to match them to audio with the goal of enhancing storytelling by eliciting emotions through fitting music. The paper focuses on the multimodal retrieval of content through shared embedding spaces between audio and text.

Soundify [Lin et al., 2023] uses video as context to match it to fitting sound effects. The system is able to sync and pan the sounds according to the video input. It demonstrates its usefulness in reducing the workload of video editors when matching sound to video.

All of these applications implement their own custom systems for matching context with a retrieved or generated output.

2.2 Toolkits and Platforms

Established context
toolkits

Infrastructure for context-sensitive applications is an established challenge for the field, named by Abowd et al. [1999]. Toolkits to aid in the creation of context-sensitive applica-

tions are also a well-trodden area, for example the Context Toolkit described in Dey et al. [2001].

Since those contributions, the emergence of generative AI has opened new possibilities for context-sensitive applications. Large language models and multimodal retrieval were not part of these toolkits.

Many different applications exist that cover features similar to Atrium's. Model routers like [LiteLLM](#)¹ or self-hosted model servers like [LocalAI](#)² act as gateways to AI models. Chat interfaces like [Open WebUI](#)³ and [AnythingLLM](#)⁴ often also offer this capability and sometimes include support for retrieving documents in integrated retrieval-augmented generation (RAG) systems [Lewis et al., 2020], which is similar to Atrium's library (Section 4.4). Orchestration and workflow tools like [LangChain](#)⁵ and [ComfyUI](#)⁶ solve problems that the core capabilities of Atrium do not touch and can be used alongside the platform. Platforms with APIs for media retrieval exist, but few offer the ability to perform semantic, vector-based search. [Freesound](#)⁷ [Anastasopoulou et al., 2025] stands out among them as an API with this capability for sound that is freely available for non-commercial use. [Weaviate](#)⁸ allows multimodal embedding and retrieval over custom data. [Home Assistant](#)⁹ is a widely used home automation platform that connects and coordinates devices, similar to Atrium's rooms (Section 4.5).

Existing tools cover only parts of Atrium

As shown in Table 2.1, all of these platforms cover part of Atrium's capabilities, but none combine them all into one package with a focus on research.

¹ <https://github.com/BerriAI/litellm>, last accessed 2026-09-25

² <https://github.com/mudler/LocalAI>, last accessed 2026-09-25

³ <https://github.com/open-webui/open-webui>, last accessed 2026-09-25

⁴ <https://github.com/Mintplex-Labs/anything-llm>, last accessed 2026-09-25

⁵ <https://github.com/langchain-ai/langchain>, last accessed 2026-09-25

⁶ <https://github.com/Comfy-Org/ComfyUI>, last accessed 2026-09-25

⁷ <https://freesound.org>, last accessed 2026-09-25

⁸ <https://github.com/weaviate/weaviate>, last accessed 2026-09-25

⁹ <https://www.home-assistant.io>, last accessed 2026-09-25

	Gateway	Library	Rooms	Event log
Model routers and servers LiteLLM, LocalAI	✓	–	–	–
Chat interfaces Open WebUI, AnythingLLM	partly	partly	–	–
Retrieval and media search Weaviate, Freesound	–	✓	–	–
Home automation Home Assistant	–	–	✓	partly
Atrium	✓	✓	✓	✓

Table 2.1: Comparison of existing tools on Atrium’s core capabilities. “Partly” means covered only for the tool’s own purposes. Existing tools can cover part of Atrium’s functionality, but the combination of these features, especially in a research context, is unique.

Chapter 3

Method

This chapter outlines the research approach and describes the contribution type of this thesis and how it is evaluated. It also gives a rough timeline of the development process and describes which sources of evidence factored into the design and changes throughout.

3.1 Research Approach

The approach of this thesis centers on the design and development of Atrium as an artifact contribution [Wobbrock and Kientz, 2016]. As such, the research contribution is the platform itself. We use the framework laid out by Ledo et al. [2018] and utilize two evaluation strategies that best fit the claim of lowering the barrier. The first is demonstration in Chapter 5, where multiple applications were built on top of the platform to showcase its capabilities. The most prominent of these applications is a new SoundMuse prototype, built at the start of this thesis and later ported onto the Atrium platform, enabling us to make some comparisons between the two. The second is usage in Chapter 6, which was evaluated in an exploratory and formative study where participants were prompted to build their own application or use the platform in other ways. No hypothesis was fixed in advance, and there was no control condition

Evaluation through
demonstration and
usage

or set task given to participants, since the goal was to observe how the platform would be used and where it would fall short, rather than to confirm any expected outcome. At the end of the study period, participants filled out a questionnaire and were asked open-ended questions during an interview. For the analysis of this data, counts with denominators were used for closed questions, and recurring issues were identified from the free-text answers and interviews. The technical performance and heuristic evaluation strategies were deliberately not chosen, as they are not relevant to the claim of lowering the barrier.

3.2 Development Process

Prototype and
interviews informed the
design

The starting point for this thesis was building a new prototype version of SoundMuse to experiment with several ideas that would later be incorporated into the Atrium platform. There were multiple other versions of SoundMuse before this prototype [Emir, 2025; Gujejani, 2025] that were used as inspiration, although the prototype was rebuilt from scratch. Besides using the other versions ourselves and observing areas for improvement on a technical level, we also conducted informal interviews with students who had used some version of SoundMuse in their research. During these interviews, notes were taken to later recall the information, but no recordings were made.

Interview points
became the platform's
goals

Points from the interviews that were mentioned by multiple people or that we noticed ourselves while building the prototype turned into the goals of the platform outlined in Section 1.1. Based on these goals, an implementation plan was created, complete with technical architecture and motivations. The planning stage as well as the implementation was performed with the help of AI coding models that were prompted to critically examine each part of the plans and code. As the platform evolved, so did some design decisions, and new ones emerged with each part of the platform. Each new design plan was also checked against the codebase with the help of an AI agent, and a plan document was created for every decision. In the end, this yielded

more than 120 plan documents for different features and revisions to the system.

Once study participants were given access to the platform, they were in regular communication with us. Any issues with the platform itself were addressed while the study was still in progress, using a similar workflow to the original implementation. Decisions made during this time are also documented.

In parallel to participants using the platform, several applications were built on Atrium that serve as demonstrations of the platform’s capabilities. Building these applications surfaced several issues, which were subsequently fixed. Some features from these example applications also made it into the platform itself, as outlined in Section 5.3.

Example applications
surfaced issues and
features

After the study period, a follow-up interview was performed and recorded with 8 of the 11 participants. Feedback from this interview as well as code written on top of the Atrium platform and in some cases other documentation provided by the participants was also used as evidence for the evaluation in Chapter 6.

Follow-up interviews
and code as evidence

Chapter 4

Atrium

An atrium is the connecting room in the center of a building, letting in light. Similarly, the Atrium platform aims to be the connection between applications and the capabilities needed to create context-sensitive applications.

This chapter documents the capabilities of the Atrium platform and the design decisions made before and during the development process. Atrium is designed with a researcher audience in mind, which shaped which functionalities were and were not included.

4.1 Design Commitments

Before development on Atrium began, several design commitments were made to ensure a well-rounded and feature-rich platform while keeping the scope manageable. These were drawn from the literature and past experiences with software development to prevent the common pitfalls a project like this might encounter.

Commitments keep the scope manageable

Simplicity over completeness Atrium does not aim to be a solution for every problem. Each feature needs to earn its place. Nothing is added speculatively, even if it is free.

Sensible defaults Wherever possible, Atrium falls back to a sensible default set in advance, so a first prototype needs no configuration. The user is only required to provide specific model names or settings in cases where relying on a default would result in unintended consequences.

Replaceability through standards Follow existing standards wherever they exist. The chat API, for example, is completely OpenAI-compatible, enabling users to replace an Atrium model with an OpenAI one. Only Atrium-specific features get their own custom contract.

Extraction from real use New features are extracted from real use cases: applications that were actually built with Atrium or user feedback for specific features.

Self-contained deployment The platform is easily deployable on its own. There is no hard requirement on external providers or services. Any additional capabilities can be configured on the platform itself.

4.2 Architecture Overview

Four core capabilities to
address barriers

Atrium's architecture rests on four core capabilities, each designed to address a specific barrier we or others have encountered while developing context-sensitive applications. On a technical level, Atrium is the shared backend through which these capabilities are accessed.

One backend,
embedding models run
separately

The foundation consists of a single Python FastAPI process as its backend, which handles routing to the app's external capabilities. The app uses one PostgreSQL 16 database for storage with the pgvector extension to enable vector similarity search. File storage and serving are handled on disk through the FastAPI backend. For the vector search, multimodal embedding models are needed. These are hosted separately on a more powerful machine and connected through an HTTPS service. This makes the embedding capabilities fully separate from the core platform, allowing

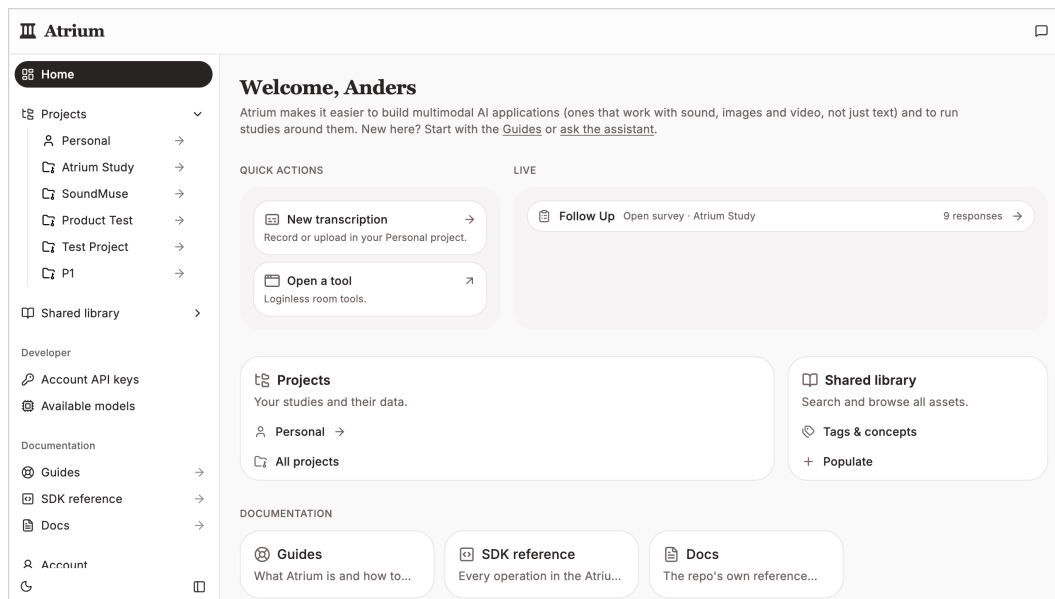


Figure 4.1: Home page of the Atrium web interface. It lets users use and learn about the platform’s capabilities without writing code.

new models to be added without touching any server code. A background job queue with several worker instances is used to process embeddings and other tasks that are not time-sensitive. The app also enables communication between different clients through the server via both Server-Sent Events (SSE) and WebSocket transport.

For easy access, a web interface was implemented with React and Vite that enables interacting with every part of the platform (Figure 4.1). The interface is optional, and using Atrium without it is possible. Any capability can also be accessed through the web API and the SDKs for Python and TypeScript.

Accessible through web interface, API and SDKs

The SDKs are intended to reduce development effort by providing an easier-to-use interface for all platform capabilities while coding and abstracting away common patterns used when developing context-sensitive applications (Listing 4.1). They are hosted on PyPI and npm respectively, so they are easy to install and use with minimal effort.

```
from hci_atrium import Atrium

client = Atrium()

reply = client.chat(
    "Name three sounds that fit a thunderstorm.",
    model="openai/gpt-5.6-luna",
)
hits = client.library.search("distant thunder",
    modality="audio", limit=5)
```

Listing 4.1: The same client reaching two different capabilities. Credentials and the instance URL are read from the environment, so a first call needs no setup beyond installing the package.

4.3 Gateway to AI Models

To analyze context, applications usually rely on AI models and adjacent capabilities. But due to constraints in hardware and software, it is often not feasible for AI chat and embedding models to be hosted in the same process or on the same machine. Because of this, the ability to connect external services was added. These can run on a different machine in the same network or be available through cloud providers like OpenAI. These external providers are largely stateless and can provide a range of different capabilities and models, which then become available on the platform and for users of the API through an OpenAI-compatible API contract.

Providers add models
behind one contract

Admins configure
providers and endpoints

In the platform's admin settings, the providers can be configured from a wide array of options. The LiteLLM package provides support for virtually any current cloud AI provider while also supporting any provider that has an OpenAI-compatible endpoint. In addition to providers, standalone "inference endpoints" can be registered that offer a single capability, like a standalone multimodal embedding model. Other capabilities are supported as well, like transcription models to convert spoken words into text or

diarization models to assign speakers inside of a transcript. Multiple instances of the same model can be registered as separate inference endpoints, which are then managed and load-balanced through Atrium.

Atrium itself exposes an OpenAI-compatible API (Listing 4.2) with the model names typically taking the form provider/model-name and inference endpoint models not having any provider prepended. This means it is possible to connect one Atrium instance to another to use its capabilities. This is especially useful for development, where a local Atrium instance can be pointed at a remote one, gaining access to all its available providers and models.

OpenAI-compatible
API, even between
instances

```
from openai import OpenAI

ATRIUM = "https://atrium.example.com/v1"
client = OpenAI(base_url=ATRIUM, api_key=KEY)

reply = client.chat.completions.create(
    model="openai/gpt-5.6-luna",
    messages=[{"role": "user", "content": prompt
    }],
)
transcript = client.audio.transcriptions.create(
    model="parakeet-tdt-0.6b-v3",
    file=open("clip.wav", "rb"),
)
```

Listing 4.2: Example calls using the Atrium API and the Python OpenAI client, accessing local and cloud models through the same endpoint.

The models used on the deployed Atrium instance were chosen to cover a wide array of functionality, preferring ones with open-source or educational licenses and established scientific literature behind them.

4.4 Shared Asset Library

The library is the core capability Atrium was built around. To solve the problem of the quality of metadata, every asset, any kind of file or link, is embedded with multimodal embedding models automatically. By default, uploaded assets land in a shared library every single user has access to. The idea behind this is that there exists one shared pool of assets any application could hook into and retrieve from using semantic comparison between a search query and assets or between different assets. The library itself hosts a range of management features where title, descriptions, tags and more can be adjusted per asset. An asset can claim any kind of modality itself; this way, links to music and music stored as bytes can both claim to be of audio modality while having different kinds of underlying files. Embeddings happen based on what file type or representation of an asset, text descriptions or frames extracted from a video for example, is attached to an asset. These representations then essentially function as a “bridge” between different modalities. This means that as long as two assets have at least one vector space in common, they are comparable.

Shared vector spaces
make assets
comparable

Excursus:
Embeddings

EMBEDDINGS:

An embedding here means the representation of the semantic meaning of content as a high-dimensional vector. Two embeddings can be compared through methods like cosine similarity to get a similarity value. Embeddings are created with the help of embedding models. Each model creates its own vector space, and vectors from different spaces are not comparable.

Admins configure the
embedding spaces

Which models, and by extension which file types, are natively supported is defined by the platform’s admins by choosing embedding spaces that can be adjusted without touching the code. For this, one or more embedding spaces can be set up, each with its own vector space. Every asset will get embedded into all enabled embedding spaces. On the hosted platform, four different embedding spaces are active as of the writing of this thesis: one for each of the main modalities this thesis focuses on (text, image and

sound), as well as one modality-spanning model that can produce vectors in a shared embedding space for all these modalities. Each of these spaces has its own strengths and weaknesses for retrieval. Specialized spaces usually have better retrieval quality than the more general ones, though general spaces have the obvious benefit of natively connecting more modalities.

Specialized spaces
retrieve better

The models that are actually deployed and used as embedding spaces in the library are:

Text EmbeddingGemma [Vera et al., 2025]

Image SigLIP 2 [Tschannen et al., 2025]

Sound CLAP [Wu et al., 2023]

Cross-modality ImageBind [Girdhar et al., 2023]

The way search and retrieval functions is that a space to search in is chosen either automatically based on the search query, like audio searches will use the dedicated audio embedding space, or manually by the user. Then the user's search query is embedded in that space, yielding a high-dimensional vector. A search query is typically a string; however, querying by images or sounds or other supported modalities is possible if the space supports it. The high-dimensional vector, which can be interpreted as representing the semantic meaning of the embedded content, is compared against every asset in the database that also carries a vector in that same space using cosine similarity search. This process is very similar to the retrieval step of established retrieval-augmented generation systems [Lewis et al., 2020]. This similarity search finds assets close in semantic meaning to the search query (Figure 4.2). This process entirely sidesteps the need for metadata, as the content itself is being searched for, not a keyword match over some text description. However, because the library supports multiple vectors per asset and per space, a user may decide to search for the text description rather than the native content bytes themselves or even use keyword search without any vector embeddings being used.

Search ranks assets by
cosine similarity

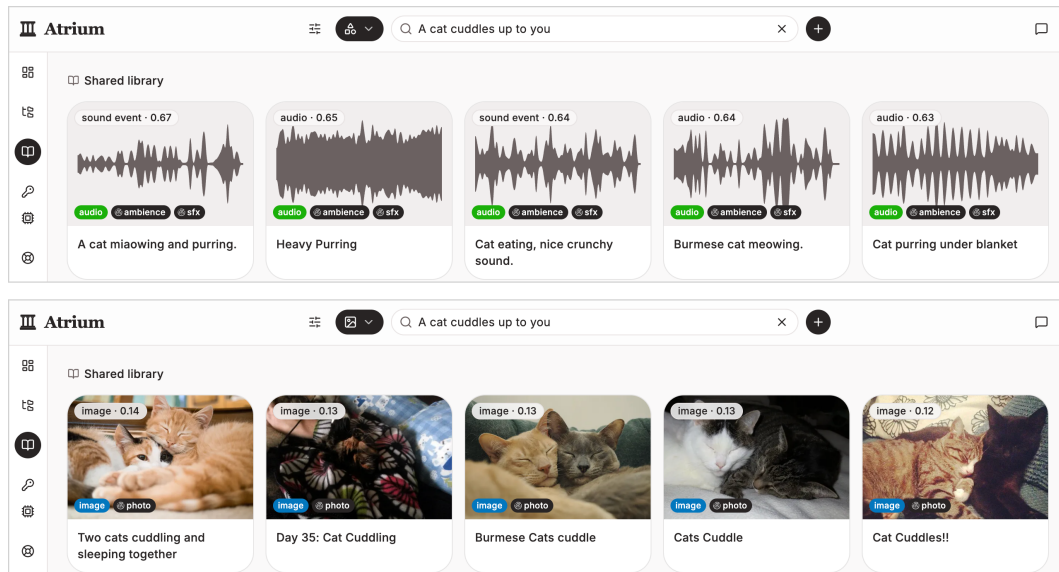


Figure 4.2: Library search with the same query “A cat cuddles up to you” in two different modalities. Audio (top) and images (bottom) are each matched in their native embedding space. Scores are not comparable across spaces.

Text descriptions
outscore native media

Also notable is a phenomenon known as the “modality gap” [Liang et al., 2022], where different modalities end up in separate regions of the same shared space, so that cross-modality matching results in different scores based on which pairs of modalities are compared. In our experience, a match between a search string and an asset’s description, which is embedded with its text content, almost always results in a better matching score than between a search string and a natively embedded sound or image, even when the description is only vaguely related and the sound or image is a theoretically perfect match.

Uploads and
enrichment fill the
library

The library can be populated with assets in several ways. Users can directly upload any file type they want and add descriptions and other metadata manually. But there are also options to automatically expand the database from external sources. The process of querying other databases and APIs to expand the shared library is called “enrichment” inside of Atrium. The main idea behind enrichment is that when no good results are retrieved from a search query, the user can decide to call an enrichment function through the SDK that automatically searches a predefined

external provider's assets through its API endpoint and adds those retrieved assets to the library. Developers can also add assets through code, essentially building their own enrichment pipeline, when the use case requires a more specialized setup or wants to use sources that are not natively supported by Atrium. The supported sources are [Freesound](#)⁷, [Openverse](#)¹, [BBC Sound Effects](#)² and [Wikimedia Commons](#)³. Openverse and Wikimedia Commons expose a keyword search API that makes a variety of modalities enrichable. BBC Sound Effects only contains sounds and has no documented public API. Instead its sound effects are queried by doing a search on a [CSV file](#)⁴ that contains a filename and a description for around 16,000 sounds. Freesound exposes the most powerful API of all the sources. It supports both a keyword search API and a similarity search API. For the similarity search, the query has to first be embedded, as for a search in the Atrium library, with a separate embedding model. The [Freesound API documentation](#)⁵ describes multiple embedding models as being available for similarity search. The one Atrium uses to query the API is CLAP-Fusion [Wu et al., 2023]. Searching by similarity can yield more accurate results depending on how the query is phrased. The SDK also enables developers to query each external enrichment source directly, including Freesound similarity search, without having to host their own version of a compatible embedding model. After an enrichment has been completed, the next identical search will include the new assets as results, matched by their local embedding vectors.

Supported enrichment
sources

4.4.1 Tags and Concepts

To further divide and categorize assets, an asset may have any number of tags. A tag is by itself just a text string and has no inherent meaning. The user can freely add any tag

Tags filter assets and
trigger side effects

¹ <https://openverse.org>, last accessed 2026-09-08

² <https://sound-effects.bbcrewind.co.uk>, last accessed 2026-09-08

³ <https://commons.wikimedia.org>, last accessed 2026-09-08

⁴ <https://github.com/FThompson/BBCSoundDownloader>, last accessed 2026-09-08

⁵ https://freesound.org/docs/api/resources_apiv2.html, last accessed 2026-09-28

they want to any kind of asset. A user could add “music” as a tag to a recording of a guitar playing while they add “ambience” to a recording of rain. Later, the assets can be filtered and retrieved by their tags. Tags can also trigger side effects when added to an asset. Adding the tag “music” can trigger the extraction of musical properties that only make sense in the context of music, like BPM or key. Properties of an asset that are extracted deterministically like this are referred to as “facts” and can also be filtered for. Which tags cause which side effect is configurable: tagging something as “speech” may trigger extraction of spoken words using the platform’s default transcription model in one instance, while in a different Atrium instance only audio tagged “podcast” is transcribed.

Manual tags do not
scale well; tags can be
added automatically in
two ways

Since the library is designed to be ever-growing, through predefined enrichment sources (Section 4.4) or manual uploads, it becomes increasingly hard to maintain consistent and high-quality tags. This is a well-documented problem of user-supplied tags [Golder and Huberman, 2006]. To solve this, Atrium supports two ways to assign tags automatically to an asset that can be managed on the dedicated “Tags” page in the web interface.

LLM tagging An LLM can be given the context of an asset: its description or an automatically generated description of its content. A description can be added to a tag, and the tagging model will decide whether a tag applies based on the provided context.

Concepts From already tagged assets, a “concept” can be extracted, which is a representation of the semantic meaning of a tag based on the tagged assets’ embeddings. The platform supports multiple ways this concept can be calculated, each resulting in one or multiple vectors in each relevant embedding space. Whether an asset carries a tag is then decided by its proximity to this concept in its vector spaces, an approach adapted from concept activation vectors [Kim et al., 2018].

LLMs are more
accurate, concepts
cheaper

Both methods have their advantages and disadvantages. In our experience, LLM tagging yields more accurate results

but is also slower and more expensive. Adding tags also requires an additional LLM call. Concepts, on the other hand, are cheap to calculate, since all the embedding vectors are already calculated ahead of time. Each change to a tag just means updating the concept and comparing each asset's vectors against the new concept vector. In practice, both methods are best combined with manual control and tagging. A user can use an LLM to tag a diverse subset of data and use those tags to create the concept. Each new asset will then be tagged by the concept. A user can correct the tags of the concept to add positive and negative examples to the dataset and improve the concept over time.

The concept activation vector (CAV) functionality is also accessible to developers outside the tag system with the `search_by_examples` function in the SDK. It can, for example, enable recommendations for a user's selection based on positive and negative assets as inputs.

Concepts also enable
search by example

4.5 Realtime Rooms

Rooms are the protocol that enables realtime communication between devices, for example to trigger an action from one device on another or to surface data generated by one device to a researcher on another device. These signals are ephemeral, meaning they are not persisted anywhere by design. Rooms do not keep any state; they are only the vehicle that data flows through. A connection is established through a randomly created code that each room is assigned. Communication itself uses SSE and WebSocket connections that is established by inputting the code. From that point on, messages can be exchanged freely through the room (Listing 4.3).

Rooms relay ephemeral
messages between
devices

Atrium provides multiple tools that can use rooms to extend an application's capabilities with little extra code and can just be opened inside of a browser tab on the same machine as the main application or a different one. These include an audio Player, an audio Capture tool, a Viewer tool for displaying images and a Dashboard for displaying debug information and monitoring events inside of a room

Atrium provides hosted
room tools

(Figure 4.3). These tools are separate inside of the code and only connect with the six-character room connect code, making them hostable anywhere and extendable to more tools by design. More on how the tools were used can be found in Chapter 5.

```
from hci_atrium import Atrium, join_room

# in the application
client = Atrium()
room = client.rooms.open(name="soundscape")
print(room.connect_code)           # K7M2QX

for event in room.receive(identity="app"):
    if event.type == "heard":
        hits = client.library.search(event.payload
                                      ["text"], modality="audio")
        room.play(hits[0], layer="ambient")

# on any other device
room = join_room("K7M2QX")
room.send("heard", {"text": "sounds like a storm
coming"})
```

Listing 4.3: A room in use. The application opens the room and reacts to what arrives in it, while a second device joins with the connect code alone, needing no account and no client of its own.

4.6 Persistent Event Log

Sessions persist data,
e.g., one participant's
run

To enable saving persistent data for research purposes, Atrium implements a persistent event log that any app can use by starting a “session”. These sessions are designed around being one participant’s run inside of the application, so they can carry additional data like a participant ID or the condition of the run. After opening a session through the SDK function (Listing 4.4), a developer may opt in to recording events broadcast inside of a realtime room to have a complete history of messages. For an even more detailed event log, the SDK can emit trace events

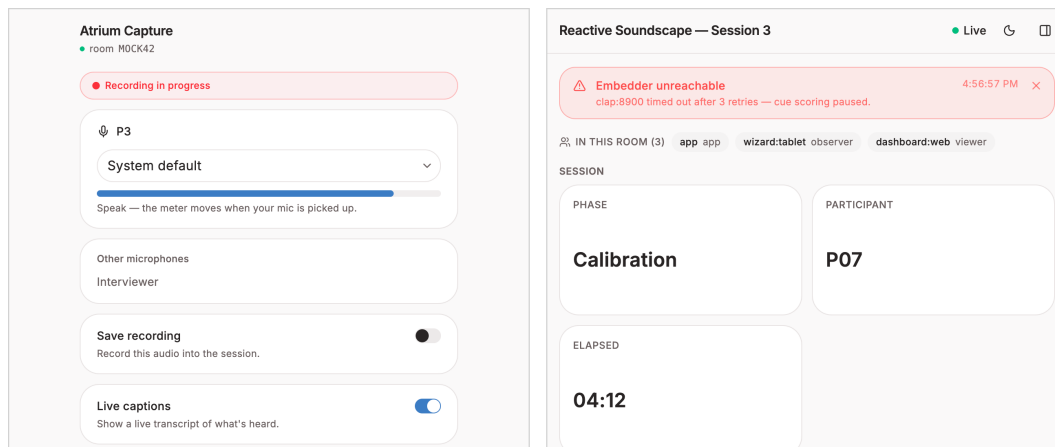


Figure 4.3: Hosted room tools with mock data. Capture tool (left) with activated live captions. Dashboard (right) showing an error. The Capture tool enables using sound as context from any device through a web browser. The Dashboard enables monitoring the events and state of an application from another device.

for its major functionalities. These are broadcast automatically and, when a session is active, recorded whenever a library search, chat call or other Atrium functionality is used through the SDK. In cases where a complete event log is not required or wanted, for example to keep the information lean and more readable, events can also be saved inside of a session manually without a room.

On top of the event log, sessions also support file uploads and offer some special cases for survey data and for live transcriptions (Section 5.4).

Data can be retrieved from a session through the web interface or through the SDK. Temporary access to write to sessions can also be granted through a one-off token, so apps that are not authenticated with Atrium directly can write to the session, for example, when redirecting from an authenticated app to a survey in the browser. The browser can gain one-off read and write access until the survey is over. Afterwards, the authenticated app can still access the data, while the browser session cannot.

One-off tokens let unauthenticated apps write to sessions

```

session = client.sessions.start(
    name="Reactive Soundscape",
    participant_ref="P07",
    condition="adaptive",
)

room.record(into=session)
session.event("phase.entered", {"phase": "
    calibration"})

client.sessions.export(session.id, "csv", path="
    run.csv")

```

Listing 4.4: Recording a run: a session opened for one participant, the room bridged into it, an event written by the application itself, and the finished log exported as a table.

4.7 Projects, Identity and Access

Access to Atrium is
managed on two layers

Access to all of these features needs to be managed. Some, like the access to external APIs, can cause charges and thus should be gated to users that are part of a specific project. Atrium provides exactly this functionality on two separate layers: who gets access to the platform in the first place and who gets access to a project and its data on the platform.

4.7.1 Projects

Projects as private
spaces

API keys scope
requests to a project

By accessing the Projects page in the web interface, a user can create a private space with its own library, data storage and developer features. Upon creation, only its creator can see this project, but they may invite any other Atrium user to become a member of this project. All members have access to read and write to the project; this is often required for applications to work in the first place. For applications it is generally advised to authenticate them with a project-scoped API key, which can be minted by any member and grants access to all the project's capabilities while scoping requests to the project automatically. So a general library

request while using a project-scoped key will query the project's library, while a request without an API key or one scoped to a user's profile will query the global library.

Each user starts off with a personal project that acts as the fallback to any request that is not scoped to another project, so data is never lost. Personal projects deliberately expose fewer features than other projects and new members cannot be added there.

Personal projects catch unscoped requests

As mentioned, by creating a project, the user gains access to a private version of the library. The libraries and surrounding features are referred to on the platform as collections. Any collection can have an arbitrary number of sub-collections, making nesting possible. A collection can inherit its parent's assets, apply a filter on them or start off fresh with an empty library of assets. Each collection can have its own vocabulary of tags and concepts, making it possible to semantically filter assets on any level. A project's library might filter the shared library to only include sounds. The project's collection then defines "music" as one of its concepts, and sub-collections can filter on that. Further sub-collections may filter further based on genre concepts or facts like BPM. In addition to filters, collections can also be filled by manually pinning assets in them from another collection or uploading new assets into them directly. The latter allows for fully private assets that are not added to the shared library.

Nested collections with their own tags and concepts

4.7.2 Identity and Access

Accessing the Atrium platform is designed to be as frictionless as possible. User access and roles can be gated by existing LDAP infrastructure [Sermersheim, 2006], removing the need to manually grant access or create accounts for every user, though Atrium does support the creation of new users separate from LDAP.

Access through existing LDAP

Inside the platform there are a few ways to gate access to data. The platform-wide settings are gated behind an "admin" role that gives access to an additional page in the web

interface where every aspect of the platform can be configured.

Sign-in for frontend-only
apps

For cases where using an API key is not feasible, like a frontend-only webpage, Atrium also offers the ability to authenticate users on behalf of an application, acting as an OAuth 2.0 authorization server [Hardt, 2012] with PKCE for public clients [Sakimura et al., 2015]. This enables giving access to Atrium features on publicly hosted URLs without having to write a backend for authentication. More on this in Section 5.1.

Room codes give
limited access

Rooms also introduce another way to authenticate non-users with room codes that give more limited access to the platform. Expensive and paid capabilities always require a user with a full sign-in through their account or an API key. If this secondary type of user wants to access additional capabilities, these need to be provided by a fully logged-in primary user.

4.8 Documentation and the Assistant

Documentation kept up
to date with AI

Since the platform is aimed at other developers, documentation becomes one of the biggest priorities to keep the platform useful, manageable and transparent. Several ways to let users retrieve data about the platform are included, and extensive documentation exists to describe every part of the platform. Generative AI models were used to create and check documentation regularly to ensure information is up to date.

Guides are practical but
incomplete

For newcomers to the platform, the guide pages give an overview with code examples for the most important interfaces of the platform. Developers can read about how to install the SDK, query the library or use AI models. Though still quite technical, these guide pages are intended to be user-facing and provide more of a guide on how the platform can be used. They are, however, not complete, meaning more specific features might not be contained in the guides. That information can be found in the documents on the “Documentation” page. These are much more de-

Detailed docs feed the
assistant

tailed, describing every part of the platform, which also makes them much less readable. They are accessible to users who want to gain a deep understanding of the platform, but they are mainly intended as the knowledge base for the on-platform AI assistant.

The Atrium AI assistant can be accessed on any page of the web interface by pressing the Chat button in the top-right corner. It opens and is designed to be used alongside the other pages. By default, the assistant has a fairly lean prompt with basic information about Atrium but can load different skills based on the user's question and which context they are in on the platform, making it in itself a sort of context-sensitive application. When the user asks a question, the assistant can do a semantic search over the documentation or access specific chapters directly. Chunks of the documentation are loaded into the assistant's context to enable it to answer. The underlying model used for the assistant is selectable by the user in the prompt input. Any model accessible through the API can also be used by the assistant, though result quality and speed vary quite drastically. Admins can set defaults for different scenarios a user might select from. An OpenAI model may be used for chatting and asking questions about the platform, since their models have a quick response time and perform well overall. But a local model default is more appropriate if the assistant handles sensitive data that should not be transferred to third parties. There are other tools the assistant has access to that can directly interact with the platform too. The assistant may link or navigate the user to a specific page or read data contained on that page. During development, the assistant also gained some experimental features, like creating surveys or marking sections inside of a transcript. More on this in Section 4.9.3 and Section 5.4.

The assistant loads
skills based on context

Local models for
sensitive data

The assistant can act
on the platform in
different ways

4.9 Extending Atrium

Atrium is designed with the fast-changing landscape of AI in mind. No models or providers are hard-coded, and they can be configured on the platform. This section describes

Models are not
hard-coded

the seams at which Atrium can be extended and configured as well as possibilities and areas for future extensions.

4.9.1 Enrichment Sources

More enrichment
sources can be added
in the future

More sources beyond the supported ones can be added as adapters in the code. These then become available in the web interface and SDK for direct querying.

In one-off cases where an enrichment source is just needed for a specific application, the enrichment can also be done manually, meaning querying and uploading assets with the SDK from the application's own code. This gives more flexibility and does not require a change to the platform. However, it may make sense to extract these sources and make them available to other users of Atrium in the future.

4.9.2 Embedders and Embedding Spaces

New models as
inference endpoints

Embedders and embedding spaces are easily configurable globally on the platform itself. As long as embedding models speak the same contract as the existing ones, they can just be added as inference endpoints in the admin settings, even beyond the three modalities that are currently supported by Atrium. Adding support for 3D models or video natively is as easy as having an embedding model that supports these modalities hosted on a server that speaks the Atrium contract and offers these as capabilities. There may be limitations due to file size and processing power that should be considered when setting up the platform before adding any more modalities.

New spaces can be
added to the library

Embedding spaces for the library are configurable in a similar manner. A new embedding space can be created in the admin settings by selecting an embedding model and setting its dimension count. Toggling it on for the library will try to embed every asset into that space, so this option needs to be handled with care. Adding a space to the library can take multiple hours, depending on how many

assets are inside of the library. The models considered and used changed during development. A remotely hosted instance of the Qwen3 text embedder [Zhang et al., 2025] was replaced with a locally hosted version of EmbeddingGemma. On assets that were added before this switch, the old vectors can still be seen and accessed. They are deliberately kept in case the embedding space is enabled again later. Some models, like Omni-Embed, were considered but ultimately excluded from the currently hosted version of the platform due to licensing and memory concerns. A server for this model is still included and could be enabled later.

4.9.3 Assistant Skills

The assistant is built around a set of skills that can be loaded in dynamically as needed. As such, more skills to aid in data analysis or other tasks could be added in the future. It does already contain basic skills for creating surveys or marking sections inside of a transcript; however, these have not been tested thoroughly and need more work in the future to really be useful in actual research.

Assistant skills and tools are extensible

4.9.4 New Capability Kinds

New capability kinds from external providers or inference endpoints can be added in the code. During development, several capability kinds were added, such as speaker diarization to separate multiple speakers inside of a transcript. In the future, more capabilities could be added here, which can then be called through the Atrium API.

Adding completely new kinds of capabilities

Chapter 5

Building on Atrium

Since the aim of Atrium is to help build context-sensitive applications, several such applications were built during the development of the platform. Some are just prototypes, while others are complete applications that are intended to be used in future research, such as a new version of SoundMuse. This chapter shows the design and development process of a few of those example applications and how they influenced the shape of the platform and SDKs.

Building applications on top of Atrium

5.1 SoundMuse

Many different types of applications with different architectures can be built with Atrium. The case of SoundMuse is especially interesting since Atrium was able to replace the entire backend of the original prototype. This section will cover the rough workflow of creating the new version of SoundMuse, but the principles are applicable to other applications as well.

Atrium replaced the entire backend in the SoundMuse rebuild

SoundMuse is built around a web interface visible in Figure 5.1. The TypeScript SDK (Section 4.2) was chosen so that Atrium can be directly used inside of a JavaScript framework, meaning the application only consists of a frontend. Since the application is intended to be hosted online,

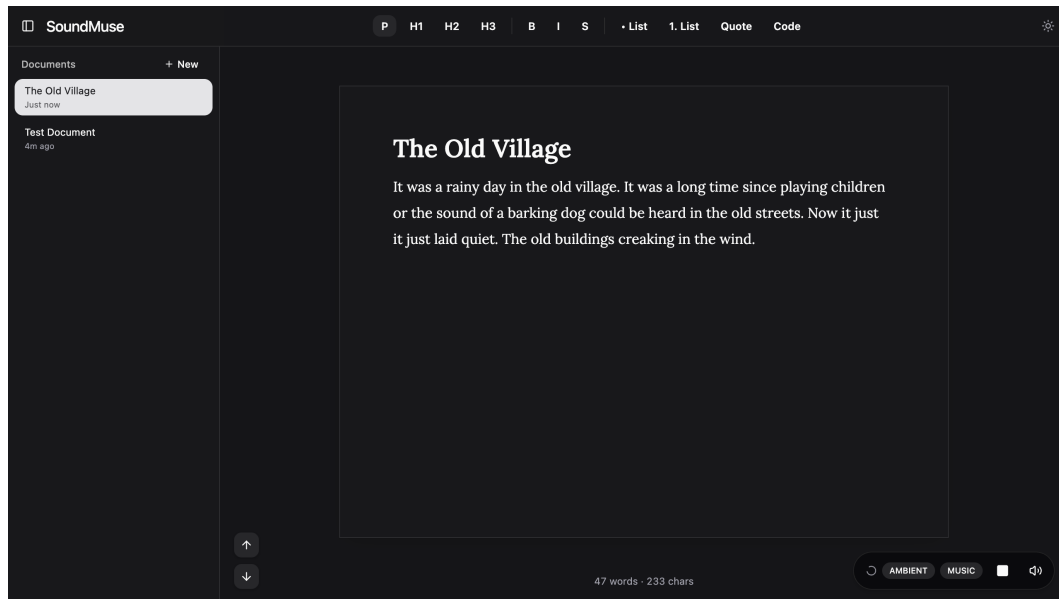


Figure 5.1: SoundMuse editor rebuilt on Atrium. Soundscape controls in the bottom right. Music, ambient sounds and sound effects are played based on the story being written in the middle.

some sort of access control had to be implemented. Atrium provides the option to sign in through the platform for this exact use case (Section 4.7.2). This gives each authenticated user access to Atrium features inside of SoundMuse.

Multiple Atrium capabilities used to pick a soundscape

Now that a user has access, the next step is building the user interface. For this, the React framework was chosen, but other frameworks would also work since the SDK is not framework-specific. Inside the UI, the functionality for authors to write text was added, with multiple text documents persisted in the local storage of the browser. When a user ends a sentence while typing, multiple different systems trigger at once. There is one system for the soundscape (the background music and ambience sounds) that gives the last typed sentences of the user to an AI agent through the gateway to AI models (Section 4.3). Atrium also provides several different tools this agent can call to access the library (Section 4.4). These can be added as tools to an agent built with the SDK's agent framework, making it possible for the agent to query the library and decide on a sound to use in a multi-step process.

Sounds can be pre-filtered by utilizing the tags and concepts to create different categories of sounds. For SoundMuse, each sound carries one of three tags (music, ambience or sfx), decided by its concepts. These categories are dynamic and learned as outlined in Section 4.4.1.

After this agent has decided on the most fitting soundscape for the current context, the sounds are played by the player component from the Atrium SDK. In cases where the application is not running in the browser and cannot play audio directly, audio can be sent through a realtime room to the Player tool (Section 4.5).

Playback through the SDK or Player tool

Adding sound effects, short noises instead of full musical pieces, works similarly to the soundscape, with the difference that their requests are optimized for speed. The library is queried first with the raw sentence of the user. This yields much less accurate results, especially if the user is typing in a language other than English, but since no LLM call is involved, the query is much faster. This pre-selection of sounds is then given to a separate agent, also created with Atrium, to pick one of the sound effects or query the library for more. This enables sound effects to be played within seconds of a sentence finishing, while the soundscape can take significantly longer.

Sound effects within seconds

If no fitting sound is found for any of these requests, the agents can decide to call the library enrichment tool to extend the library's pool of sounds for all future requests.

To monitor what is happening in the background, the Dashboard room tool is used, which shows the current sounds being played as well as a history of calls to the library and to enrichment. Everything being broadcast in this room is also persisted as a session (Section 4.6) in the SoundMuse project so it can later be revisited.

Monitoring through the Dashboard room tool and recording into a session

Comparing this version of SoundMuse based on Atrium with the prototype created at the beginning of the thesis, the lines of code needed for comparable functionality have significantly dropped from 23,360 to 8,221 for the application code excluding vendored UI components, tests and database migrations. Code needed for the features that

Lines of code for similar functionality reduced significantly

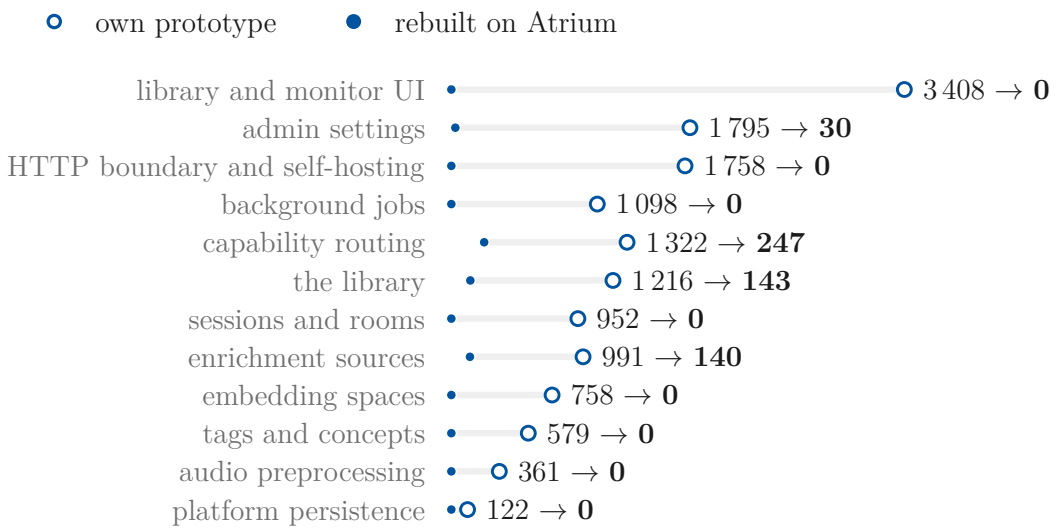


Figure 5.2: Lines of application code per area in the prototype built at the start of the thesis and in the version rebuilt on Atrium. Most areas were moved to Atrium completely while others could be reduced to only SoundMuse-specific code.

Atrium now provides went from 14,360 lines in the prototype to 560 lines in the new version. Figure 5.2 shows how it breaks down per area.

5.2 Other Example Applications

More applications were built to demonstrate Atrium

Several other applications were created during this thesis to test and showcase Atrium’s features.

Text adventure Using an LLM to guide the player through a set story, this application incorporates sound effects that play alongside text to showcase similar features to SoundMuse, but it builds on the Python SDK. The library (Section 4.4) and the agent framework are at the center of this application. It focuses on a small fantasy story with several story beats that all have distinct background music. As the story changes and a quiet festival day in a medieval town turns into a dragon attack, the atmosphere changes through the background music and noises. This application was

used to showcase the platform to participants of the study.

Mood board This application is based on the TypeScript SDK and implements very similar functionality to Mood Board Search by Google Research and Nord Projects [Google Research & Nord Projects, 2022]. Based on a selection of positive and negative examples of images, more similar images are retrieved from the library. This shows the image capabilities and also another use case for concepts (Section 4.4.1).

Reacting to speech This example uses the live transcription functionality of Atrium (Section 4.3) to subscribe to what is being said inside of a room, captured with the Capture tool (Section 4.5). Based on that input, some example actions can be displayed or the mood and intent of the speaker can be analyzed.

5.3 Extraction into the SDK and Platform

Some features from the example applications were extracted, generalized and incorporated into the platform and SDK. Features have to earn their place in the SDK. If only one example needs a feature, such as the state machine used for the story progression in the text adventure, it stays in the application. On the other hand, features that prove their worth by being used in multiple applications or by wrapping previously obscure parts of the platform are considered for the SDK. The system behind queuing music and crossfading between tracks, used in SoundMuse and the text adventure, is an example of the first case. The `search_by_examples` function used by the mood board is an example of the second, as it made the similarity search behind concepts on the platform accessible in a more convenient format.

Features need to earn their place in the SDK through repeated use

5.4 Workbenches

Built-in applications for
research, based on
Atrium features

We considered two applications useful for HCI research in general and added them to the Atrium web interface directly. These are built on top of the same functionality that any external app building on Atrium has access to, but were included as part of the platform for convenience under the name workbenches (Figure 5.3).

Transcription Workbench This application allows users to create or upload recordings of interviews and can transcribe them in real time. During or after the transcription step, specific areas can be marked or commented on in different colors to help with the analysis step. It also integrates with the assistant, which has the ability to access the transcript and mark sections in it.

Survey Workbench This application enables users to create their own surveys with a variety of question types. These are intended to be integrated into an application and can be opened from the SDK and connected to a study session. The workbench is also usable as a standalone tool with the ability to create public surveys that let anyone with the public link participate. As with transcripts, the assistant can also aid in creating surveys by generating questions or by helping with translation into multiple languages.

Both of these workbenches were employed heavily during the evaluation of the Atrium platform itself.

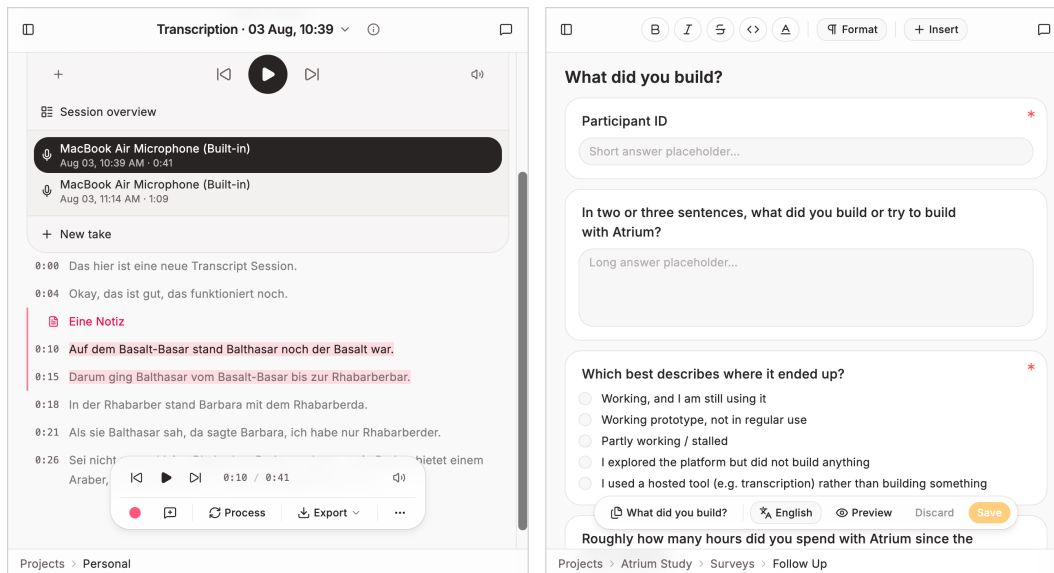


Figure 5.3: Transcription workbench (left) with mock data and a highlighted note. Survey workbench editor (right) with the actual Atrium follow-up survey. Both workbenches build on capabilities provided by Atrium.

Chapter 6

Evaluation

This chapter evaluates the usage of the Atrium platform. The study is structured as an exploratory and formative study to examine how participants would use the platform and where they get stuck. Section 3.1 explains why this approach was taken. Participants were predominantly from the i10 chair and recruited by us.

6.1 Study Design

After some initial communication, a first meeting was organized with each of the 11 participants. Each person gave consent for the meetings to be recorded and their data to be used (see Section A.5). During this meeting, general questions about experience and past projects were discussed (Section A.1). Then the participants were given a rough tour through the Atrium platform, starting from the transcription workbench, which was also used to record and transcribe the meeting's audio, to the library and projects. Finally, the text adventure was shown as a demonstration of Atrium's capabilities.

Consent, interview and
platform tour

Afterwards, the participant was given full access to the platform, where they would explore the different pages on the web interface while they were asked to express their

Free exploration while
thinking aloud

impressions and thought process. Participants who did not find projects, the documentation or the platform assistant on their own were nudged towards them as a means to gather more information on the platform. Direct questions were answered by us when the participant asked them on their own, but not unprompted.

Access to Atrium for
about two weeks to use
freely

After the participant had explored the most important sections of the platform, they were given the choice of how they would like to test the Atrium platform. The test period was set to roughly two weeks per participant, but varied based on their availability. The first option was to use the workbenches and tools already integrated into the platform, for example, to create transcriptions of their own audio. The second option was to use Atrium as a developer and build their own application. What kind of application they wanted to build was deliberately kept open and individual per participant, depending on their interest and prior experiences.

Participant
demographics

After that choice, they had the opportunity to ask more questions or be aided in setting up the development environment before the meeting was over. A questionnaire on background and demographics was answered by all participants after the meeting (Section A.3). Of the participants, 3 were bachelor's students, 4 master's students and 3 PhD students. One participant worked as a developer. Six participants were men, 4 were women and 1 preferred not to say. Every participant came from the field of computer science or media informatics. Ages of participants ranged from 19 to 36, with a median of 28. Participants had a median of 10 years of programming experience (range 5–19).

Feedback integrated
during the study

During the study, each participant had the option to ask us questions and give feedback for the platform. These could include problems that arose, general questions or feature ideas. They were also prompted to do this after the first week of testing via text message. Reported feedback was often quickly integrated and shaped the platform while the study was still running.

Follow-up with 8
participants

With 8 of the participants, a follow-up meeting was held, where they showcased the way they used the Atrium plat-

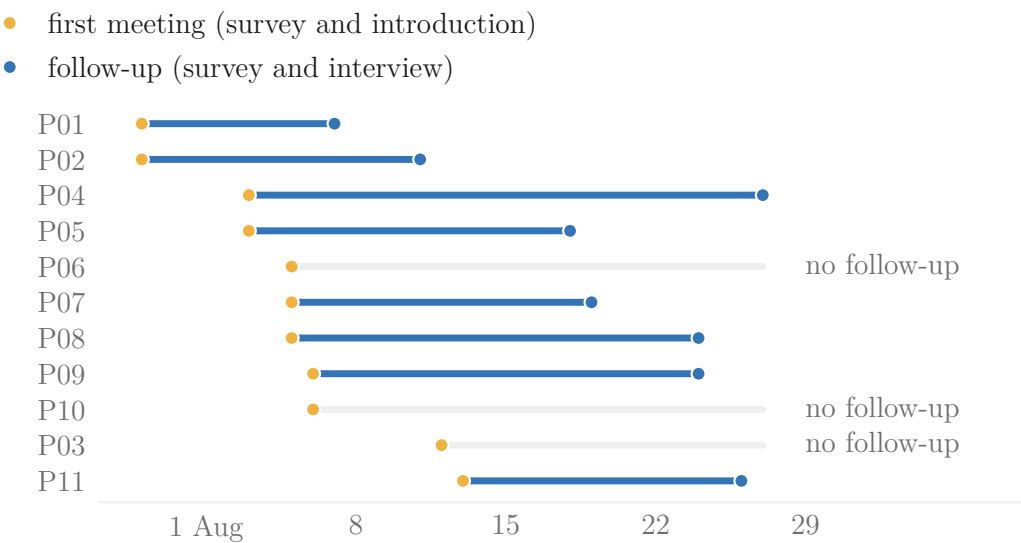


Figure 6.1: Timeline per participant, first meeting to follow-up. There was no follow-up meeting with 3 participants.

form and answered questions about their experiences (Section A.2). Before that meeting, a second questionnaire (Section A.4) was filled out by the participants about their time with the platform. Most of the follow-up meetings were also recorded. Beyond the meetings, the code that participants wrote against Atrium was also examined to further investigate how they used the platform. A breakdown of the timeline per participant can be found in Figure 6.1.

The questionnaires had both free-text answers and closed-ended questions. Closed-ended questions are reported as counts per item, because of the small sample size and because participants used the platform in different ways, so not all questions applied to all of them.

6.2 How Atrium Was Used

Atrium was used in different ways by each participant. As mentioned in Section 6.1, participants were given a choice to build an application with Atrium or use the tools provided by Atrium. Figure 6.2 shows the usage split and

Most spent under 5 hours

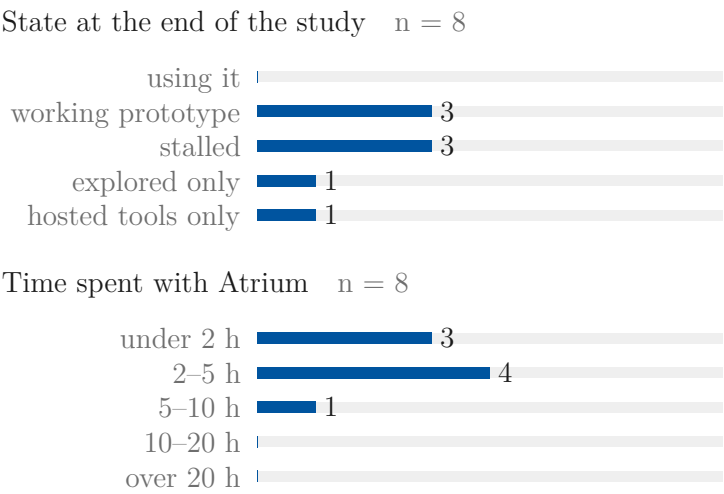


Figure 6.2: State of participants’ projects at the end of the study and time spent with the platform. Three projects ended up as working prototypes and most participants spent under 5 hours with the platform.

how much time participants spent overall with the platform. Most participants, 7 of 8, spent under 5 hours with the platform. Three participants created a working prototype, while 3 participants’ projects stalled at some point due to limited time. One participant only explored the platform and its documentation, while another only used the hosted tools and workbenches on the platform.

Varied approaches to use and code with Atrium

Multiple approaches to coding were used. Some applications were created without the help of AI, while others were created mostly using coding agents. Atrium provides documentation specifically optimized for coding agents (Section 4.8). Both the Python SDK and TypeScript SDK were used for different applications. Figure 6.3 shows which parts of Atrium participants used and which ones they knew existed.

Working prototypes created by participants

As mentioned before, three applications reached a working prototype state:

Blender plugin Built with the Python SDK, the plugin allows 3D models to be sourced from third-party APIs

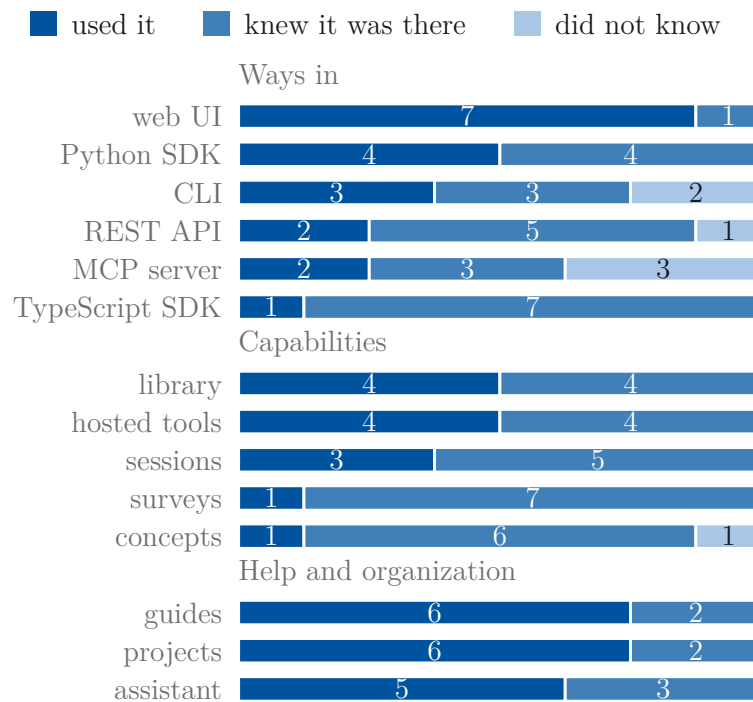


Figure 6.3: Overview of the usage and knowledge of the parts of Atrium by study participants. “Ways in” are the surfaces a user interacts with, “Capabilities” are Atrium’s features, and “Help and organization” are ways to learn more about the platform. The web interface was used by almost all participants. Library and hosted tools were the most used capabilities.

and embedded into the Atrium library, which makes them retrievable by their description. The plugin provides a search box that queries a private project library for the most fitting 3D model. It also allows uploading 3D models from [Blender](https://www.blender.org)¹ directly into the library.

One Minute Math Built in [Angular](https://angular.dev)² with the TypeScript SDK, the game lets the user select their favorite music genres. Then it queries the library for music that fits those genres and plays it while the user tries to com-

¹ <https://www.blender.org>, last accessed 2026-09-25

² <https://angular.dev>, last accessed 2026-09-25

plete as many simple math problems as they can in one minute. Afterwards, the library is queried again for music intended to distract them, while they perform the same math-based task again. A score is then used to compare the two scenarios.

Atrium Pong Built with [pygame](https://www.pygame.org)³ and the Python SDK, the game sources its sound effects and music tracks from Atrium on startup. The music gets increasingly intense as the levels and speed of the game progress.

The participant who only used the hosted tools mainly worked with the transcription workbench for transcribing and analyzing their own research data. Another participant, whose project stalled, used the AI assistant and survey workbench to create a mock survey.

6.3 Participant Feedback

Feedback depended on
the type of use

Participants gave feedback on the parts of the platform that they used. Participants who only explored or whose projects stalled mainly gave feedback on the documentation and the project setup process, as well as the layout and design of the web interface. Participants who built a working prototype also often gave feedback on the specific functions of the SDK and platform on a more technical level.

Built-in help was used
most

When participants got stuck or encountered problems, they mostly opted to use the platform's built-in help. When asked which sources of help they used, with more than one answer possible, 6 of 8 participants selected the guide pages on the web interface and 5 of 8 Atrium's built-in assistant (Section 4.8). Only 2 participants used an LLM outside of Atrium for help.

Most respondents
would likely not have
started without Atrium

Participants were asked to assess the amount of effort their project would have taken them if they did not have access to the Atrium platform. These questions were optional, and participants were able to skip them if they did not use

³ <https://www.pygame.org>, last accessed 2026-09-25

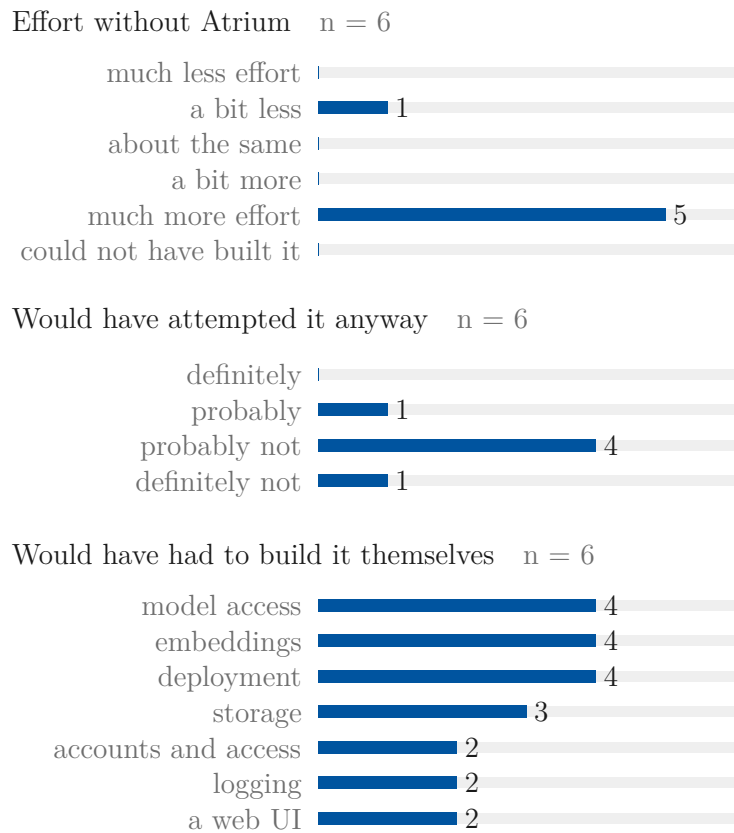


Figure 6.4: Effort and attempt without Atrium, and the parts participants would have had to build themselves. Most participants who answered would not have attempted their project without Atrium and said it would have taken them much more effort without it.

Atrium to build something. In the end, 6 of the 8 participants answered them. Figure 6.4 shows that 5 of 6 participants said that without Atrium, building their application would have required much more effort. Additionally, 5 of 6 participants answered that they probably or definitely would not have attempted their project at all without Atrium. The figure also gives an overview of the features that participants would have had to build had the platform not provided them. Model access, embeddings and deployment are the most prominent here with 4 of 6 each. This is a similar list to the parts SoundMuse no longer had to implement itself after the port (Section 5.1).

Mostly model access,
embeddings and
deployment

All 8 would recommend
or probably recommend
it

All 8 participants answered that they would recommend (5) or probably recommend (3) Atrium to someone working on a similar project. Three participants answered that they intend to keep actively using Atrium after the study, two of them building their thesis on it.

Core functions just
worked

The core functions of the platform were mentioned as working well. When setting up audio playback in an Angular app, P07 found that “it just worked out of the box.” P02 described search, chat completions and rooms as working “very well and ... fast” and noted: “I never felt like I was stuck with these.”

The workbenches, especially the transcription workbench, were also mentioned as a positive. P09 preferred it over existing transcription tools because “you can directly mark text,” and P04 named speaker recognition as working better than expected and liked “having all of my audios in one place, the transcriptions and the notes.”

Hierarchy and naming
confused some

Some systems were less intuitive to participants. P02 found the hierarchy of collections and libraries “overkill and confusing.” P08 described the start page of projects as “a bit crowded, but the offered tools seem helpful,” while P09 recommended changing “some naming conventions” to make the UI more intuitive.

Having the core functionalities side by side with workbenches on the web interface felt disconnected to some participants. P07 mentioned that “[t]hese two areas seem not really connected.” Some participants encountered issues during the setup process. P08 had problems “trying to create/connect to a room via Python,” which in the follow-up meeting they attributed to the assistant not giving accurate answers. This problem was also mentioned by P09, who found that the assistant often hallucinated while they were trying to get information on the SDKs.

Changes were made to address these issues during and after the study. Section 7.2 gives more details on how the platform changed.

Chapter 7

Discussion

This chapter discusses the claim that Atrium lowers the barrier to multimodal context-sensitive applications. As outlined in Chapter 3, the sources of evidence used to examine this claim are demonstration (Chapter 5) and usage (Chapter 6). We weigh the claim against the evidence from those chapters, reflect on the development of the platform through its use, and name the limitations of this approach and what it does not prove.

7.1 Lowering the Barrier

In Section 1.1, four goals were outlined as problems the platform aims to solve. Each of the goals and the features implemented to achieve them is supported by different evidence, from demonstration through example applications and from usage by the study participants.

The goal of understanding the context through the gateway to AI models (Section 4.3) is backed mainly by demonstration through applications like SoundMuse and the text adventure. These were implemented without calling a third-party AI API directly. Live speech transcripts were made accessible, which was also showcased through an example application and the usage of P11, who built an early version

Understanding the
context

of an app that could recognize what is being said. P04, who used the transcription workbench, also showed the usefulness of this speech recognition on real data.

Finding fitting material	Finding fitting material through the shared asset library (Section 4.4) is backed by both usage and demonstration. Figure 6.3 shows that 4 of 8 participants used the library in some capacity. As the library is a central part of Atrium, all three applications that reached a prototype state relied on its functionality in different ways. Similarly, SoundMuse and the text adventure demonstrations were built around this capability.
Syncing devices in real time	Syncing devices and applications in real time through real-time rooms (Section 4.5) is backed mainly by demonstration. Many of the example applications, like the text adventure or the reacting to speech example, heavily rely on this feature to play sound, record audio or show what is happening in the background on the Dashboard.
Recording research data	Recording research data through the persistent event log (Section 4.6) is backed mainly by use of the workbenches. P04 used the transcription workbench on real study data, and we used the survey workbench for evaluating the Atrium platform. Atrium Pong also stores data like scores through sessions, and SoundMuse stores snapshots of the written story. Figure 6.3 shows that sessions were used by 3 of 8 participants in some capacity.
Library and hosted tools used most	A pattern is evident from these four goals. Study participants mostly interacted with the library and the hosted tools. Other features were incorporated much more rarely. Some projects that intended to use these features stalled, such as those of P11 and P08. Reasons for this were often named as the limited time or the projects being too ambitious for the study period.
Evidence of reduced effort from the SoundMuse rebuild	The demonstration evidence through rebuilding SoundMuse shows that moving functionality to the shared backend requires much less code for the same functionality in similar applications. The rebuild has around 65% fewer lines overall, and the parts that Atrium now provides went from 14,360 to 560 lines. Atrium takes over the parts of

the application that made up the barrier as defined in Section 1.1. Although the lines-of-code metric can be used to approximate the effort it takes to build such an application, other factors, such as lessons from the first implementation and developing the platform itself, also contributed to reducing this metric. The Atrium platform was also built with SoundMuse in mind, so other applications will likely fit the platform less well, and the metric should be seen as an upper bound for the reduction.

Usage evidence points in a similar direction to the demonstration evidence, since 5 of 6 participants answered that their project would have taken them much more effort without Atrium. It can also be argued that Atrium lowers the barrier to starting a project like this, since 5 of 6 participants answered that they probably or definitely would not have attempted to build their project without Atrium in the first place. Further pointing towards this conclusion are comments from participants like P01, who noted that they “wouldn’t have thought about it” if it were not for the Atrium platform and its capabilities, and P09, who noted that without Atrium “the cost of starting my small projects would be too much.” Participants were prompted to create something with Atrium, which made their projects naturally a good fit for the platform. That 2 participants created working prototypes in under 5 hours and 1 participant in under 10 hours also points towards a reduction of effort needed to get something up and running.

Usage evidence points
the same way: less
effort with Atrium

Pointing against this evidence is P08, who had trouble with the initial setup, especially when the assistant hallucinated and gave wrong advice. For their smaller-scope project, where they intended to play relaxing sounds on some input, they noted that learning how to query an API like Freesound directly would have been a bit less effort for them. It is also important to note that these answers are self-reported, based on a hypothetical, and there was no control group. These limitations are further discussed in Section 7.3.

Some evidence against
reduced effort

Still, both types of evidence point in the same direction and indicate that Atrium lowers the barrier to this class of multimodal context-sensitive applications by reducing the effort

Atrium lowers the
barrier but has a
learning curve

and lowering the initial hurdles to creating such an application. However, it does not remove them entirely, and it has a learning curve of its own that needs to be overcome by new users.

7.2 Development Through Use

Feedback changed
docs and interface

Throughout the study and beyond, the feedback of the participants was incorporated into the platform. Much of this feedback was directed at the learning curve mentioned in Section 7.1. Core capabilities remained largely unchanged, while the documentation and user interface changed a lot during the study. This section covers some of the bigger changes that were informed by usage, though these are not exhaustive.

Tutorials, SDK pages
and consistent IDs

Due to feedback from multiple participants, the guide and documentation pages were expanded over time to include tutorials and explanations for various parts of the platform, like the collection and library hierarchy or the reacting to speech workflow. Feedback from P08 prompted the addition of separate pages for the SDK, something that only existed for the API before. Inconsistencies between surfaces were also fixed, like the IDs that were exposed through the SDK and API not matching those visible in the web interface, which used more human-readable, short IDs, while the SDK exposed the UUIDs that were only intended for internal use.

Better tools against
hallucinations

The assistant and its capabilities changed a lot during the study. It gained multiple tools to help it identify correct paragraphs in the documentation and SDK references and retrieve correct information. This was done in direct response to P08 and P09, who both encountered hallucinations in relation to the SDK. Also important for the assistant was the selection of the underlying LLM. Using a locally hosted model caused these hallucination issues due to the smaller context size. For general use, the option to use an external provider like OpenAI was added with a warning not to send sensitive information, which noticeably im-

proved the answers but is a trade-off between answer quality and privacy.

The layout of features in the web interface was changed multiple times due to feedback. Based on feedback from multiple participants, such as P04, projects gained the ability to hide specific features to make the interface less cluttered in cases where only a subset of them would be used. As a consequence of feedback from P02, the project libraries also gained the ability to enrich directly from their populate page, which previously was only possible in the shared library. The SDK surface stayed shared-library-only so as not to encourage enriching a project library with material that could benefit every application. Since custom enrichment pipelines are possible in code, it is still possible to enrich on any level, but global enrichment was kept as the default.

Hideable features and
direct enrichment

The workbenches, especially the transcription workbench, gained the highlighting and comment features based on feedback from P04 during the initial meeting and their needs for transcribing and analyzing their own data.

Highlights and
comments added to
transcriptions

The same principle that motivated changes from user feedback also motivated changes to and extraction of features from our own applications. Developing SoundMuse exposed gaps in the library retrieval pipeline for modalities that are embedded in multiple embedding spaces. Features needed for the text adventure, such as playing music through a browser and adding tool calls to the agent framework, were directly extracted from its development into the SDK according to the design commitment of extraction from real use (Section 4.1).

Own applications also
drove changes

Changing the platform in response to how it was used is part of lowering the barrier. The initial setup and orientation in particular needed this feedback. The fast turnaround of platform features was only possible because the platform was implemented by a single developer with the help of AI coding agents (Chapter 3). Additional upkeep will be required after the thesis due to the nature of these kinds of platforms.

7.3 Limitations

No control condition, self-reported answers	Due to the nature of the study design, there were no fixed tasks per participant and no control condition. The without-Atrium answers were self-reported and based on a hypothetical. The number of participants was limited, and even more so for participants who were able to answer questions about what they built with Atrium. Eleven participants attended the first meeting, 8 the follow-up, and only 6 answered the without-Atrium questions. The time participants spent with Atrium was also limited. Most participants spent under 5 hours with the platform in total, which gives no indication of long-term use and left features like rooms and the gateway to AI models underexplored. The platform changed during the study, meaning no two participants saw the exact same platform, which was intended under the formative framework of the study.
Participants mostly from our own chair	Participants were predominantly from the same chair as us and recruited by us, which can inflate answers such as whether they would recommend the platform.
We were developer, evaluator and interviewer	We were developer, evaluator and interviewer at once, and built both versions of SoundMuse, which can bias both the design and the interpretation of the results.

Chapter 8

Summary and Future Work

In this chapter, we summarize the contributions and findings of this work and give an outlook on where the Atrium platform could go next.

8.1 Summary and Contributions

This thesis laid out the problem of researchers reimplementing similar features for the same class of application, requiring a large amount of development effort and time for tasks not directly related to their research. Then we defined the four main goals that, when achieved, could lower the barrier to creating these multimodal context-sensitive applications.

Problem and four goals

We introduced and implemented Atrium as a shared backend with the ability to act as a gateway to AI models and a shared asset library, to synchronize devices through real-time rooms and to record research data in a persistent event log, stored as sessions.

Four capabilities in one platform

Atrium was then evaluated based on demonstrations as well as usage in an exploratory and formative study. Feed-

A lower barrier with a learning curve

back was incorporated continuously while the platform was tested. The findings from both of these sources of evidence indicate that Atrium does lower the barrier to developing multimodal context-sensitive applications, but it has a learning curve of its own. Of the 6 participants who answered, 5 said that their project would have taken them much more effort to create without Atrium, and all 8 follow-up participants would or probably would recommend the platform for similar projects. Three participants intend to keep actively using Atrium, two of them building their thesis on it.

Available at i10 and
self-hostable

Atrium is available as an artifact contribution to anyone at the i10 chair. It is possible for the platform to be hosted elsewhere, and the process is documented as part of the codebase. Also contributed are example applications, especially the new version of SoundMuse, which can be used as a starting point for future research in this area.

The effectiveness of Atrium as an aid for other research and long-term projects still has to be proven, but Atrium was designed from the very beginning to be extensible and malleable to future requirements.

8.2 Future Work

Atrium as a platform is designed to be extended and to act as a basis for future research in the domain of context-sensitive applications.

More modalities

Future work could add more modalities to the platform. Groundwork for this has already been laid, but it would need specialized embedding models. The Blender plugin showed how Atrium currently can bridge the gap to other modalities via text, but native support for them is also feasible in the future. More platforms and programming languages could be natively supported through an SDK, making it easier to develop 3D scenes with an engine like Unity or iOS applications with Xcode and Swift.

More SDKs and
platforms

Atrium could be extended to bring AI features to aid researchers. The in-platform assistant is already home to tools that can help with creating surveys and analyzing transcripts, though these need additional development and testing to be useful. Maintenance of the platform will likely be an ongoing task. Many of the features are easily configurable on the platform, but future developments in AI and changing requirements over time could make it necessary to further expand the feature set of the platform to incorporate and support more kinds of applications.

AI features for
researchers

Ongoing maintenance
needed

Larger-scale evaluations of the platform itself could also be future work, ideally with data from using it in real use cases and research projects.

Many interesting areas of research could utilize Atrium’s capabilities. These could include tabletop role-playing games that react to the context of the game with dynamic visual and audio cues, shape-changing interfaces as input and output for context, additional modalities in Sound-Muse such as 3D scenes, and creativity studies in general.

Future research areas

Overall, we hope that Atrium provides a solid foundation for context-sensitive applications, so that those building them can focus on their research questions rather than on the infrastructure behind them.

Appendix A

Study Materials

A.1 Interview Guide, First Meeting

The first meeting took about one hour per participant and closely followed this guide.

Consent (5 min)

- Framing: plan for the hour; the study evaluates the system, not the participant.
- Consent form handed out, questions answered, recording started in the transcription workbench.

Interview (10 min)

- Q1 Experience doing projects with AI and/or multiple modalities (audio, image, text, video).
- Q2 If you want to use a new (software) tool, what do you do first?

Q3 Did you ever want to use multiple devices for a project at the same time, for example a phone and a laptop synced?

Showcase (10 min)

- Library page, search and search across modalities.
- Story demo (text adventure).
- Phone and code connected to the Dashboard.

Access (10 min)

- Sign in or create an account.

A1 Free exploration of the platform (nudged towards projects, documentation or the assistant if not found).

A2 Transcription and survey tools.

A3 Where would you look to learn more?

Path (15 min)

- Choice between building an application and testing the tools.
- Brainstorming ideas for the participant's own project.
- Optional: help with setup, open questions.

Handover (5 min)

- Questionnaire to be filled in after the meeting.
- Follow-up meeting arranged.

A.2 Interview Guide, Follow-Up Meeting

Since use of Atrium was individual per participant, this guide was only loosely followed during the follow-up meeting.

Participants who built something

- Setup: framing (evaluation of the system, not the participant), consent still valid, recording started.
- Showing what they built: the participant presents their project.
- Walkthrough: where did you start, which example or guide page did you use, where did you diverge from it?
- Two moments: the positive and the negative moment from the questionnaire, step by step.
- Without Atrium: how would you have built this without Atrium? Would you have started at all?
- Closer look at one feature the participant used heavily.
- Close: what would you take out, and anything the questionnaire did not ask.

Participants who did not build anything

- Setup, as above; not having built anything is explicitly fine.
- Where you stopped: what was the last thing you did in Atrium, and what would have come next?
- What you explored: which parts, in which order?
- Your idea: which parts of Atrium would you use, and where do you think you would get stuck?

- Feedback on the guide pages, the web interface and the assistant.
- What kept the project from happening?
- Close, as above.

A.3 Questionnaire, First Meeting

*Questions marked with * were required. Pages are separated by headings.*

Welcome

A short questionnaire on your relevant background for the Atrium user study. Your data will be used as outlined in the consent form you have signed or received during the meeting. You may withdraw your participation from the study at any time by contacting us. Contact: Anders Petruschke, anders.petruschke@rwth-aachen.de

☐ I agree to my data being collected as described on the consent form*

Who are you?

1. Participant ID* (*prefilled from the invitation link*)

2. Gender* (*single choice*)

- ☐ Woman
- ☐ Man
- ☐ Non-binary
- ☐ Prefer not to say
- ☐ Self-describe

3. Please specify *(short text)*

Only shown if "Self-describe" was selected.

4. Age *(number)*

5. Which best describes you right now?* *(single choice)*

- ☐ Bachelor's student
- ☐ Master's student
- ☐ PhD student / research assistant
- ☐ Professional software developer
- ☐ Other

6. Please specify* *(short text)*

Only shown if "Other" was selected.

7. Your field of study or degree program *(short text)*

About our meeting**8. Which of the tools you saw today could you imagine using yourself? (tick all that apply)** *(multiple choice)*

- ☐ Transcription workbench (record, transcribe, edit, export)
- ☐ Observation dashboard
- ☐ Player (layered, reactive audio, like in the story demo)
- ☐ Surveys (like this one)

☐ None of them, honestly

9. If you ticked something: what would you use it for? (one line is plenty) *(free text)*

Programming background

10. How many years of experience do you have programming, counting your studies? *(number, 0–60)*

For the two scales below: 1 = never used, 2 = can follow examples, 3 = can build small things independently, 4 = comfortable, use it regularly, 5 = expert.

11. Python proficiency* *(5-point scale)*

never used ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 expert

12. TypeScript / JavaScript proficiency* *(5-point scale)*

never used ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 expert

More background

13. Other programming experience *(free text)*

14. Domain experience. Tick everything you have used in a real project, not just read about. *(multiple choice)*

- ☐ Audio programming / DSP
- ☐ Machine learning / computer vision
- ☐ LLM APIs
- ☐ Real-time or networked apps (WebSockets, multiplayer, streaming)
- ☐ Sensors / embedded
- ☐ Game development
- ☐ Creative coding (Processing, TouchDesigner, Max/MSP, Pd, ...)
- ☐ Web frontend
- ☐ None of these

15. Which operating systems do you use for programming? (*multiple choice*)

- ☐ macOS
- ☐ Windows
- ☐ Linux
- ☐ Other

AI coding assistants

16. How do you use AI assistance when coding? (tick all that apply) (*multiple choice*)

- ☐ Autocomplete in my editor (Copilot-style suggestions)
- ☐ A coding agent that works in my project (Claude Code, Cursor, ...)
- ☐ A separate chat I copy-paste into (ChatGPT, Claude, ...)
- ☐ I don't use AI assistance

17. Roughly what share of the code you wrote in the last month was first drafted by an assistant? (in percent) (*slider, 0–100 % in steps of 5*)

0 % _____ 100 %

- ☐ I didn't write any code last month

When you get stuck

18. You are using an unfamiliar library and something does not work. Rank what you actually do: 1 for your first move, then 2, 3, and so on.* (*ranking*)

- ☐ Read the official documentation
- ☐ Look for an example / sample project
- ☐ Read the library's source code
- ☐ Search StackOverflow / GitHub issues / the web
- ☐ Ask an LLM or coding assistant
- ☐ Ask a person (colleague, friend, chat channel)
- ☐ Trial and error until it works

When a tool fights you

19. A tool or app you use doesn't do what you expect. Rank what you actually do: 1 for your first move, then 2, 3, and so on.* (*ranking*)

- ☐ Keep trying things until it works
- ☐ Open the built-in help or documentation
- ☐ Search the web for the problem
- ☐ Ask an AI assistant (ChatGPT, ...)
- ☐ Ask a person
- ☐ Give up and work around it another way

Finally

20. Anything else you want to note? (*free text*)

Done

Thank you for participating! You are free to use Atrium in the next couple of weeks as discussed. Please reach out in the meantime if you have questions, ideas or you experience any issues. You can close this tab now.

A.4 Questionnaire, Follow-Up

*Questions marked with * were required. Pages are separated by headings.*

Welcome back

A short questionnaire about the time you spent with Atrium. Your data will be used as outlined in the consent form you have signed or received during the first meeting. You may withdraw your participation from the study at any time by contacting us. Contact: Anders Petruschke, anders.petruschke@rwth-aachen.de

☐ I agree to my data being collected as described on the consent form*

What did you build?

1. Participant ID* (*prefilled from the invitation link*)

2. In two or three sentences, what did you build or try to build with Atrium? (*free text*)

3. Which best describes where it ended up?* (*single choice*)

- ☐ Working, and I am still using it
- ☐ Working prototype, not in regular use
- ☐ Partly working / stalled
- ☐ I explored the platform but did not build anything
- ☐ I used a hosted tool (e.g. transcription) rather than building something

4. Roughly how many hours did you spend with Atrium since the first meeting? *(single choice)*

- ☐ Less than 2
- ☐ 2–5
- ☐ 5–10
- ☐ 10–20
- ☐ More than 20

5. What was your usage connected to? *(single choice)*

- ☐ My thesis
- ☐ A course project
- ☐ A research project
- ☐ Personal curiosity
- ☐ Something else

6. Please specify *(short text)*

Only shown if “Something else” was selected.

What did you touch? (1 of 2)

For each of these, pick the one that fits.

	I used it	I knew about it but didn't use it	I didn't know it existed
7. Python SDK	☐	☐	☐
8. TypeScript SDK	☐	☐	☐
9. The REST API directly	☐	☐	☐
10. The MCP interface	☐	☐	☐
11. The atrium command-line tool	☐	☐	☐

What did you touch? (2 of 2)

	I used it	I knew about it but didn't use it	I didn't know it existed
12. The web platform UI	☐	☐	☐
13. The guide pages	☐	☐	☐
14. The in-platform assistant	☐	☐	☐
15. Hosted tools (transcription, dashboard, ...)	☐	☐	☐
16. Sessions and events	☐	☐	☐
17. Library search	☐	☐	☐
18. Concepts / search-by-examples	☐	☐	☐
19. Projects	☐	☐	☐
20. Surveys	☐	☐	☐

21. Pick one of the things you just marked as not knowing about. Where would you have looked for it? *(free text)*

22. Which single surface did most of your work go through? *(dropdown)*

Options: each of the 14 items above, or "None of them / not applicable".

When you got stuck

Only order the ones that applied to you.

23. Did you write code against Atrium (Python/TS SDK, REST, MCP)?* *(single choice)*

- ☐ Yes
- ☐ No, that didn't come up for me

24. You were writing code against Atrium and something did not work. Rank what you actually did: 1 for your first move, then 2, 3, and so on.* *(ranking)*

Only shown if the previous question was answered with "Yes".

- ___ Read the official documentation
- ___ Look for an example / sample project
- ___ Read the library's source code
- ___ Search StackOverflow / GitHub issues / the web
- ___ Ask an LLM or coding assistant
- ___ Ask a person (colleague, friend, chat channel)
- ___ Trial and error until it works

25. Did you use the Atrium web app or one of its hosted tools?* *(single choice)*

- ☐ Yes
- ☐ No, that didn't come up for me

26. The Atrium web app or one of its tools didn't do what you expected. Rank what you actually did: 1 for your first move, then 2, 3, and so on.* (*ranking*)

Only shown if the previous question was answered with "Yes".

- ☐ Keep trying things until it works
- ☐ Open the built-in help or documentation
- ☐ Search the web for the problem
- ☐ Ask an AI assistant (ChatGPT, ...)
- ☐ Ask a person
- ☐ Give up and work around it another way

27. Which of these did you actually reach for while working with Atrium? (tick all that apply) (*multiple choice*)

- ☐ The in-platform assistant
- ☐ The guide pages
- ☐ The example projects
- ☐ Atrium's own source code
- ☐ Asked Anders directly
- ☐ An AI assistant outside Atrium
- ☐ None of these

Two moments

28. Describe one moment where something worked faster or more easily than you expected. What were you doing? (*free text*)

29. Describe one moment where you were properly stuck. What unblocked you, or didn't? *(free text)*

Without Atrium

If you didn't build anything, skip this page.

30. If you had wanted to build this without Atrium, how would you have gone about it? *(free text)*

31. How much effort would that alternative have taken, compared with the time you actually spent with Atrium? *(single choice)*

- ☐ Much less than I actually spent
- ☐ A bit less than I actually spent
- ☐ About the same
- ☐ A bit more than I actually spent
- ☐ Much more than I actually spent
- ☐ I could not have done it at all

32. Which specific pieces would you have had to build or configure yourself? (tick all that apply) *(multiple choice)*

- ☐ Model access and API keys
- ☐ Embeddings and vector search
- ☐ File / asset storage
- ☐ Auth and user accounts
- ☐ Session / event logging
- ☐ A web UI

- ☐ Deployment and hosting
- ☐ None of these

33. If Atrium had not existed, would you have attempted this project at all? (*single choice*)

- ☐ Definitely
- ☐ Probably
- ☐ Probably not
- ☐ Definitely not

34. Why? (*free text*)

35. Did you do anything that you don't think you could have done any other way? What was it? (*free text*)

After the study

36. Do you intend to keep using Atrium after the study? (*single choice*)

- ☐ Yes, actively
- ☐ Yes, if the project continues
- ☐ Probably not
- ☐ No

37. Why? (*free text*)

38. Would you recommend Atrium to someone else working on a similar project? *(single choice)*

- ☐ Yes
- ☐ Probably
- ☐ Probably not
- ☐ No

Gaps

39. Of the parts of Atrium you actually used, was there anything you would take out? Something that felt unnecessary, got in your way, or whose point you never understood. *(free text)*

40. Anything else you want to note? *(free text)*

Done

Thank you! You can close the tab now.

A.5 Consent Form

PARTICIPANT INFORMATION AND CONSENT FORM

Evaluating Atrium: A Toolkit for Multimodal, Context-Sensitive Applications

Principal investigator: Anders Petruschke
RWTH Aachen University
Email: anders.petruschke@rwth-aachen.de

PURPOSE

The study consists of two recorded sessions about two weeks apart, with independent use of Atrium in between. It investigates how developers and students use Atrium, a software toolkit for building multimodal, context-sensitive applications: software that uses AI models to understand live sound, speech, images and other media, and reacts to them. The study examines how well the toolkit and its documentation work in practice, and is conducted as part of a bachelor's thesis at RWTH Aachen University.

The study does not assess your programming ability or technical knowledge. No previous experience with Atrium is required.

VOLUNTARY PARTICIPATION

Participation in this study is voluntary.

You may:

- refuse to answer any question;
- refuse to participate in a particular activity;
- request a break at any time;
- ask for the audio recording to be paused or stopped;
- withdraw from the study at any time without providing a reason and without any negative consequences.

DATA COLLECTION

The following data may be collected during the study:

- audio recordings of both sessions;
- written notes made by the researcher;
- your questionnaire and diary answers;
- code and files you write during the study;
- platform usage data (which functions were called, which documentation pages were opened).

Interviews are transcribed automatically by Atrium itself on institute infrastructure; audio is not sent to any external transcription service.

USE OF IDEAS AND MATERIALS

Ideas, code and materials produced during the study may be analysed and reproduced in the thesis, academic publications or presentations. These materials will be associated only with a pseudonym.

The recordings will be used for research analysis and will not be publicly released in their original form. Short quotations from participants may be included in the thesis, academic publications or presentations. Quotations will be pseudonymised and, where necessary, edited to prevent identification without changing their meaning.

Participation does not transfer ownership of your ideas, code or projects to the researcher.

CONFIDENTIALITY AND DATA STORAGE

All personal information collected during the study will be treated confidentially. Recordings and research data will be stored securely on RWTH Aachen University servers. Access will be limited to the researcher and the academic supervisor. You may request the deletion of identifiable data collected from you by contacting degavarelli@cs.rwth-aachen.de.

Please tick each box to confirm your agreement.

☐ I have read and understood the information on this form.

☐ I agree to the sessions being audio recorded.

☐ I agree to the collection of my platform usage data and of the code and files I write during the study.

Participant's name

Participant's signature

Date

Principal Investigator

Date

Bibliography

- [1] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a Better Understanding of Context and Context-Awareness. In Hans-W. Gellersen, editor, *Handheld and Ubiquitous Computing (HUC'99)*, volume 1707 of *Lecture Notes in Computer Science*, pages 304–307, Berlin, Heidelberg, 1999. Springer. doi.org/10.1007/3-540-48157-5_29.
- [2] Panagiota Anastasopoulou, Alastair Porter, and Frederic Font. The Freesound API: Advances in Audio Search and Retrieval. In *Proceedings of the Web Audio Conference 2025*, Paris, France, 2025. doi.org/10.5281/zenodo.17991481.
- [3] Keith Cheverst, Nigel Davies, Keith Mitchell, and Adrian Friday. The Role of Connectivity in Supporting Context-Sensitive Applications. In Hans-W. Gellersen, editor, *Handheld and Ubiquitous Computing (HUC'99)*, volume 1707 of *Lecture Notes in Computer Science*, pages 193–207, Berlin, Heidelberg, 1999. Springer. doi.org/10.1007/3-540-48157-5_19.
- [4] Anind K. Dey. Understanding and Using Context. *Personal and Ubiquitous Computing*, 5(1):4–7, 2001. doi.org/10.1007/s007790170019.
- [5] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction*, 16(2-4):97–166, December 2001. doi.org/10.1207/S15327051HCI16234_02.
- [6] Greda Eshiba Emir. Exploring AI-Curated Context-Sensitive Stimuli for Creative Processes. Master's thesis, RWTH Aachen University, 2025.
- [7] Xianzhe Fan, Zihan Wu, Chun Yu, Fenggui Rao, Weinan Shi, and Teng Tu. ContextCam: Bridging Context Awareness with Creative Human-AI Image Co-Creation. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, pages 1–17, Honolulu, HI, USA, May 2024. ACM. doi.org/10.1145/3613904.3642129.
- [8] Rohit Girdhar, Alaaeldin El-Nouby, Zhuang Liu, Mannat Singh, Kalyan Vasudev Alwala, Armand Joulin, and Ishan Misra. ImageBind: One Embedding

- Space To Bind Them All. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15180–15190. IEEE, 2023. doi.org/10.1109/CVPR52729.2023.01457.
- [9] Scott A. Golder and Bernardo A. Huberman. Usage Patterns of Collaborative Tagging Systems. *Journal of Information Science*, 32(2):198–208, April 2006. doi.org/10.1177/0165551506062337.
- [10] Google Research & Nord Projects. Mood Board Search. Experiments with Google, <https://experiments.withgoogle.com/mood-board-search>, July 2022.
- [11] Nestan Gujejani. *SoundMuse: Contextualized Sound Cues for Creative Writing*. Bachelor’s thesis, RWTH Aachen University, 2025.
- [12] Dick Hardt. The OAuth 2.0 Authorization Framework. Request for Comments 6749, Internet Engineering Task Force, October 2012.
- [13] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viégas, and Rory Sayres. Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV). In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2668–2677. PMLR, 2018.
- [14] David Ledo, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg, and Saul Greenberg. Evaluation Strategies for HCI Toolkit Research. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–17, Montreal QC, Canada, April 2018. ACM. doi.org/10.1145/3173574.3173610.
- [15] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, pages 9459–9474, 2020.
- [16] Weixin Liang, Yuhui Zhang, Yongchan Kwon, Serena Yeung, and James Zou. Mind the Gap: Understanding the Modality Gap in Multi-modal Contrastive Representation Learning. In *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*, pages 17612–17625, 2022.
- [17] David Chuan-En Lin, Anastasis Germanidis, Cristóbal Valenzuela, Yining Shi, and Nikolas Martelaro. Soundify: Matching Sound Effects to Video. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–13, San Francisco, CA, USA, October 2023. ACM. doi.org/10.1145/3586183.3606823.

- [18] Nat Sakimura, John Bradley, and Naveen Agarwal. Proof Key for Code Exchange by OAuth Public Clients. Request for Comments 7636, Internet Engineering Task Force, September 2015.
- [19] Jim Sermersheim. Lightweight Directory Access Protocol (LDAP): The Protocol. Request for Comments 4511, Internet Engineering Task Force, June 2006.
- [20] Michael Tschannen, Alexey Gritsenko, Xiao Wang, Muhammad Ferjad Naeem, Ibrahim Alabdulmohsin, Nikhil Parthasarathy, Talfan Evans, Lucas Beyer, Ye Xia, Basil Mustafa, Olivier Hénaff, Jeremiah Harmsen, Andreas Steiner, and Xiaohua Zhai. SigLIP 2: Multilingual Vision-Language Encoders with Improved Semantic Understanding, Localization, and Dense Features. arXiv preprint arXiv:2502.14786, 2025.
- [21] Henrique Schechter Vera, Sahil Dua, Biao Zhang, Daniel Salz, Ryan Mullins, Sindhu Raghuram Panyam, Sara Smoot, Iftekhar Naim, Joe Zou, Feiyang Chen, Daniel Cer, et al. EmbeddingGemma: Powerful and Lightweight Text Representations. arXiv preprint arXiv:2509.20354, 2025.
- [22] Jacob O. Wobbrock and Julie A. Kientz. Research Contributions in Human-Computer Interaction. *Interactions*, 23(3):38–44, April 2016. doi.org/10.1145/2907069.
- [23] Minz Won, Justin Salamon, Nicholas J. Bryan, Gautham J. Mysore, and Xavier Serra. Emotion Embedding Spaces for Matching Music to Stories. In *Proceedings of the 22nd International Society for Music Information Retrieval Conference (ISMIR 2021)*, pages 777–785, 2021. doi.org/10.5281/zenodo.5624483.
- [24] Yusong Wu, Ke Chen, Tianyu Zhang, Yuchen Hui, Marianna Nezhurina, Taylor Berg-Kirkpatrick, and Shlomo Dubnov. Large-Scale Contrastive Language-Audio Pretraining with Feature Fusion and Keyword-to-Caption Augmentation. In *ICASSP 2023 – 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2023. doi.org/10.1109/ICASSP49357.2023.10095969.
- [25] Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 Embedding: Advancing Text Embedding and Reranking Through Foundation Models. arXiv preprint arXiv:2506.05176, 2025.

Index

AI Assistant	31, 54
Artifact Contribution	4, 11
Concept Activation Vector (CAV)	25
Context	7
Context Toolkit	9
Context-Sensitive Application	1, 7
Design Commitments	15–16
Embedding Space	20, 32
Enrichment	22, 32
Evaluation Strategies	11
Exploratory and Formative Study	11, 43
Gateway to AI Models	18–19
Modality Gap	22
Multimodality	8
Persistent Event Log	26–28
Realtime Rooms	25–26
Retrieval-Augmented Generation (RAG)	9, 21
Shared Asset Library	20–25
SoundMuse	2, 35–38
Workbenches	40

