

Large Language Models for Research Data Workflows

Sessions 3 and 4 · Hands-On Practice and Independent Work

Summer School Musicology 2026

Gender – Knowledge – Mobility

Digital Perspectives in Musicology

Lecture Notes

- [Dr. Christopher Pollin, Digital Humanities Craft](#)
- AI assisted: GPT-6, Claude Fable 5.1

Sessions 3 and 4 · Large Language Models for Research Data Workflows

Companion slides:  Summer School Musicology 2026 · Session 3

Sessions 3 and 4 · Guided Practice and Independent Work

Agenda and Learning Objectives (Session 3 & 4)

- Use LLMs for coding and build small research tools through Promptotyping.
- Understand AI agents and the role of an AI harness; practise context, knowledge and agentic engineering.
- Explore M³GIM, try the small CSV exercise, then develop a digital edition or research dashboard around your own question.

Work in progress. These AI-assisted lecture notes combine existing explanations with bullet-point outlines. {{todo}} marks sections still to be developed.

AI Agents Existed Long Before LLMs

- An AI agent perceives an environment, selects actions and pursues goals with some degree of autonomy. Agent research predates language models.
- Autonomy concerns acting without continuous supervision; reactivity concerns environmental changes; proactiveness concerns initiating goal-directed action; social ability concerns interaction with people or other agents.
- The slide links Wiener's Cybernetics, Wooldridge and Jennings, AlphaGo, multi-agent hide-and-seek and Voyager. Voyager is the LLM-based example; these examples use different architectures.

Cybernetics and Critique (Nosthoff and Maschewski)

The slide "Kybernetik und Kritik" presents a critical perspective on cybernetic ideas in digital platforms. It asks how feedback, prediction and self-regulation become instruments of governance. Treat the claims as an attributed theoretical position.

- Discuss the reduction of human behaviour to predictable patterns, the amplification of platform feedback and the preference for system stability over political contestation.
- Connect this critique to research practice by asking who sets the goals, which feedback counts and who can inspect or change the system.
- Reading and discussion materials:
<https://www.suhrkamp.de/buch/anna-verena-nosthoff-kybernetik-und-kritik-t-9783518300794> ; <https://youtu.be/BZo-3ni4paU> ; <https://youtu.be/cvZW3TD2jNE> ;
<https://fredturner.stanford.edu/books/counterculture-cyberculture-stewart-brand-whole-earth-network-and-rise-digital-utopianism>

LLM-based AI Agent

An LLM-based agent pursues a goal through repeated actions and observations. It may inspect a directory, read an XML file, write code, run a command and decide what to do next from the result. A tool error or a browser observation can change the next action. An explicit planning step or persistent memory may support this process, but neither should be treated as a universal requirement of every agent.

- Tool calls are examples of actions. Execution results inform the next step; available tools, permissions and stopping conditions bound autonomy.
- The workshop cycle is inspect, build, check and revise. Terms such as AI agent and agentic AI differ across publications.

AI Harness

The harness is the software around the model that manages its interaction with the environment. It supplies working context, exposes tools, executes permitted actions, maintains task state and returns results. The model produces text and tool calls. The environment contains the files, terminal processes and browser state on which those tools operate. The researcher specifies the intended result and judges its relationship to the sources.

A model can generate browser code without observing its rendered result. Browser inspection is only available when the harness actually provides an appropriate tool. If it does not, open the view yourself and return the relevant observation. Ask the agent to distinguish checks it executed from checks it merely recommends. A statement that something works needs evidence from the corresponding behaviour.

- Claude Code is the demonstrated harness. Equivalent environments are suitable when they can inspect the project files and run the required tools.

Research Data Workflows in M3GIM

The M³GIM examples explain the relationship between archival sources, structured descriptions and research interfaces. Source images show the material document. Transcriptions record a reading. Metadata and encodings describe selected features and relationships. A browser view determines which of those features a reader can inspect and how they are connected.

- The JSON-LD excerpt associates Potsdam with record m3gim-data:NIM_003_1_1 through a performance-place role. Read the role with its record and source evidence.
- A catalogue or interface count does not automatically measure performances or journeys. The complete published graph and the twelve-statement starter serve different exercises.
- Single-record example with context:
<https://chpollin.github.io/summer-school-musicology-2026/downloads/m3gim-jsonld-example.jsonld>

Hands-On: Exploring M3GIM

1. Open <https://dhcraft.org/m3gim> and inspect the prototype, which remains a work in progress.
2. Connect a place, an operative role and a date in one documented example. Identify one person or institution in Malaniuk's artistic network.
3. For each question, cite the supporting record and distinguish what is answerable from missing evidence or functionality.
4. Turn one observed gap into a requirement for a research tool.

Hands-On: AI Harness and M3GIM Data

1. Create m3gim-explorer, then open it with File → Open Folder in VS Code. Start a new harness conversation for that workspace.
2. Select the model and effort specified for the exercise. The slide currently names Sonnet 5 with Medium or High effort; available settings depend on the harness.
3. Download m3gim-mobility-starter.csv into the folder and confirm it appears in the Explorer. Download: <https://chpollin.github.io/summer-school-musicology-2026/downloads/m3gim-mobility-starter.csv>
4. Run Prompt 1, observe file access and tool execution, and check one statement and its date interpretation against the CSV.
5. Run Prompt 2 in the same conversation. Open the local URL and inspect the resulting interface yourself.

Prompt 1: Explore the Data

Explore the CSV in this folder. Use Python to inspect its structure, values and missing data. Summarise what it represents, including patterns and uncertainties. Briefly explain what you ran and show the key results. Keep the CSV unchanged. Do not build a website yet. I have no coding experience; use simple, concise language.

Prompt 2: Build a First View

Build a simple static website from the inspected CSV. Use plain HTML, CSS and JavaScript without a framework or build step. Choose one suitable interactive visualization and show sources and record details on selection. Do not install anything. Use a clear, compact interface with short labels. Start a local server, give me the URL and explain how to start and stop it myself.

- Preserve source IDs and original values; keep uncertainty visible. The twelve rows represent statements from six documents and three places, not twelve independently established events. Distinguish document dates from event dates. Place mentions do not confirm attendance or travel routes, and gender must not be inferred from names.
- The slide permits implementation without an agent inspecting the frontend. You still need to check the browser result. Compare a displayed record, its source and an uncertain date against the CSV.

Open the Website Locally

- Live Server: with the VS Code extension installed, right-click index.html and choose Open with Live Server.
- Harness: ask it to start a local server and provide the browser URL.
- Python: in the project folder run `python -m http.server 8000`, or `python3` if required. Open <http://localhost:8000> and stop the process with Ctrl+C.

Workspace and Claude Code in VS Code

- Claude icon: open the harness; Explorer: inspect project files; prompt field: give the current task.
- The terminal displays the working folder. Confirm it points to m3gim-explorer before running commands.
- Project files persist on disk. A file becomes model context only when the harness reads or supplies it.

Prompt, Context, Knowledge and Agentic Engineering

Knowledge Engineering creates and maintains explicit project knowledge. In this exercise, record the research purpose, the meaning and relationships of the data, implementation decisions and unresolved questions. Context Engineering selects from this maintained knowledge for the current task. Agentic Engineering organises the tool-supported work and its inspection. The starter and final project let you practise these activities with your own research requirement.

Context and Agentic Engineering in Practice

LLMs have become increasingly capable of generating code and carrying out extended software-development tasks.[8] Their effective and responsible use, where such use is warranted, requires more than selecting a capable model or formulating an effective prompt.[9] Results also depend on the information available to the system, the organisation of the work performed by LLM-based AI agents, and the technical environment through which they interact with a project. These interdependent informational, organisational, and technical concerns are addressed here through Context Engineering, Agentic Engineering, and AI harnesses.

Context Engineering refers here to the systematic selection, organisation, maintenance, and provision of the information an LLM-based system requires for its work. It extends Prompt Engineering, understood here as the iterative development of prompts through changes to their content or the techniques applied to them (Schulhoff et al. 2024), from individual prompts to the wider informational environment in which they are interpreted (Mei et al. 2025). Within Promptotyping, this environment is organised around maintained project knowledge that records the project's current understanding of its data and purpose, together with decisions and unresolved questions relevant to implementation. Context Engineering does not consist in placing all available material into a context window. It involves maintaining an inspectable body of project knowledge from which task-specific context can be selected and refreshed.

An AI agent is understood here as an LLM-based system that pursues a goal through a sequence of tool-supported actions and adapts to intermediate results (Sapkota, Roumeliotis, and Karkee 2026; Wang et al. 2024). Rather than responding only to an isolated input, an agent may inspect project resources, act on them, evaluate the results, and continue its work across several steps. Agentic Engineering refers here to the systematic organisation of this extended work. It concerns how tasks are decomposed and coordinated, how agents use tools and respond to intermediate results, when human intervention is required, and how their work is inspected and continued. The term is used broadly because Promptotyping agents work not only on code but also on the documents through which a project is described, directed, and assessed.

An AI harness is the technical environment through which an agent receives context, accesses project resources, uses tools, and obtains feedback. It mediates the relation between the model and the environment in which the work is performed. Agentic capability therefore arises from the combined model–harness–environment system rather than from the model alone (Zhong and Zhu 2026). A harness may provide controlled access to project files, execution environments, tests, and running artefacts while maintaining state and permissions. It thereby supplies the technical conditions under which Context Engineering and Agentic Engineering can be applied.

Promptotyping is not tied to a particular AI harness or degree of delegation. Its earliest cases relied on iterative sequences of prompts, with humans directing each step. Contemporary agentic development systems combine a high-capability LLM with an AI harness that gives the agent controlled access to a persistent project environment.[10] When this environment is maintained as a Git repository, project documents, code, configuration, tests, and references to relevant research data can be kept together under version control. Agentic work can thus draw on and modify a versioned project rather than remain confined to an isolated prompt or code fragment.

As AI agents become capable of carrying out longer sequences of computational work without human intervention,[11] the knowledge needed to guide them must remain accessible throughout these tasks. This creates a methodological requirement: instructions, decisions, and intermediate results must be maintained beyond individual prompts and conversations, so that agents can build on prior work and project contributors can inspect and correct it.[12]

Source Notes for Context and Agentic Engineering

[8] Evidence for this development comes from several complementary forms of evaluation. Kwa et al. (2025) estimate that the human-equivalent duration of predominantly software-oriented tasks completed by frontier agents at 50 per cent reliability approximately doubled every seven months between 2019 and 2025, while cautioning that the external validity of this trend remains uncertain. Benchmarks involving research replication, research

extensions and complex software engineering likewise show capabilities extending beyond isolated code generation, but also substantial remaining limitations (Starace et al. 2025; Deng et al. 2025; Edwards et al. 2026). These evaluations are treated here as indicators of expanding but task-dependent capability, not as measures of general research automation.

[9] Responsible use may include declining to use an LLM where its scholarly value does not justify the associated risks and costs. This discussion focuses on frontier-capability models because extended code generation and agentic software development currently rely largely on capabilities concentrated in computationally intensive systems. This focus does not endorse their concentrated and often proprietary infrastructures, limited transparency and reproducibility, or environmental and material costs, but reflects the need to examine their growing influence on research critically.

[10] Examples available at the time of writing include Claude Code, OpenAI Codex, and the agent functionality provided by Cursor. Their interfaces and architectures differ: agents may operate through a terminal, an integrated development environment, or a remote execution environment.

[11] Kwa et al. introduce the task-completion time horizon to measure the duration, expressed in human expert completion time, of tasks that AI agents can complete at a given level of reliability. Across tasks drawn primarily from software engineering, machine-learning research engineering, cybersecurity, and general reasoning, they find that the 50 per cent task-completion time horizon of frontier models increased exponentially between 2019 and 2025, approximately doubling every seven months. The measure refers to the human time required to complete a task rather than the agent's actual runtime. The authors explicitly caution that the external validity of this trend remains uncertain. See Thomas Kwa et al., "Measuring AI Ability to Complete Long Tasks", arXiv:2503.14499 (2025).

[12] Chen et al. conceptualise long-horizon machine-learning research engineering as repeated implementation, experimentation, and refinement across heterogeneous stages. Their results indicate that maintaining durable project artefacts across agent roles and invocations contributes substantially to performance. This supports treating long-horizon agentic work as a problem of maintaining cumulative and inspectable project state rather than of local reasoning alone. See Guoxin Chen et al., "Toward Autonomous Long-Horizon Engineering for ML Research", arXiv:2604.13018 (2026).

Knowledge Documents

A knowledge document is a structured record of externalized knowledge, distilled from sources, data or experience for reuse by people and AI agents. Its claims remain traceable to evidence, with assumptions and uncertainties made explicit.

- Dual-readable: understandable to people and usable as context by AI agents.
- Compact: synthesises information while preserving meaning, provenance and uncertainty.

- Portable: uses open, plain-text formats such as Markdown for retrieval, linking and versioning.
- Curatable: supports review, correction and enrichment as evidence and understanding develop.

The context window holds the information available for a model step. Longer input can make relevant information harder to retrieve or use, depending on the task and model. Context engineering selects and organises that information. A useful summary preserves the decisions, evidence references and unresolved questions needed for the next step. Compression should retain those distinctions.

What Humanities Researchers Do with Data

- Unsworth's scholarly primitives describe discovering, annotating, comparing, referring, sampling, illustrating and representing.
- TaDiRAH groups activities under capture, creation, enrichment, analysis, interpretation, storage, dissemination and meta-activities.
- Rhee describes historians' interactions with sources, including chaining, browsing, extracting, verifying, orienting and constructing contextual knowledge.
- Use these vocabularies to name the research operation your proposed tool should support. Capturing a passage, comparing statements and interpreting a relationship require different evidence and interfaces.

Research Questions

- RQ1: How did singers shape Graz's music and theatre culture, and how did mobility contribute to professionalisation and networks?
- RQ2: Which narrative and aesthetic structures were influenced by migration, and how did they contribute to transforming opera?
- Choose a tractable question within this wider context. Identify the source statements needed and distinguish what the selected corpus can show from what remains unanswered.

Scholar-Centred Design and Requirements Engineering

- The slide "Scholar-Centred Design und Requirements Engineering" connects knowledge acquisition with requirements. Workshops, expert interviews and literature review help articulate researchers' practices.
- A persona describes a relevant research perspective. A user story connects a role, an action and a purpose: "As a ..., I want to ..., so that I can ...".
- The Norton financial-history examples concern transaction networks, personal and institutional relationships, change over time, community comparisons and participation in credit and trade. They illustrate how requirements depend on a research domain; they are not additional datasets required for this workshop.
- For a musicological example: as a researcher, select a documented assignment to inspect its source quotation and determine what kind of evidence supports it. Use only attributes supported by the data.
- Source: <https://resolver.obvsg.at/urn:nbn:at:at-ubg:1-220602>

Promptotyping

- The two Promptotyping slides introduce the method and its project workflow. Promptotyping iteratively turns research resources and maintained project knowledge into digital research artifacts through context engineering.
- Read the method discussion:
https://lisa.gerda-henkel-stiftung.de/digitale_geschichte_pollin
- Inspect the method repository:
<https://github.com/DigitalHumanitiesCraft/Promptotyping>
- Keep the research question, source interpretation and implementation decisions inspectable. A revision may reveal a coding defect or a requirement that needs clarification.

Promptotyping Project: A Small Digital Edition or Research Dashboard

1. Open a project folder in your AI harness. With a chat interface, upload the relevant files and save generated documents and code into your folder manually.
 2. Choose one path: PDFs and available transcriptions → TEI XML → a small edition; or the full published M³GIM dataset → a research dashboard.
 3. State your research question and intended result. Explain what users should be able to investigate and identify a source example with which to check it.
 4. Inspect the actual files and create knowledge/data.md, knowledge/research.md and knowledge/specification.md. The project uses three documents.
 5. Plan the first small milestone before implementation. Agree one main view, one core interaction and acceptance checks based on real source records.
- Technical scope: a local static website using only plain HTML, CSS and JavaScript. No libraries, frameworks, package installations, build tools, backend, database or external services. Serve local assets through a simple local server.
 - Edition source PDFs:
<https://chpollin.github.io/summer-school-musicology-2026/downloads/m3gim-fulltext/m3gim-pdf.zip> ; reuse checked transcriptions and TEI where available.
 - Dashboard data:
<https://chpollin.github.io/summer-school-musicology-2026/downloads/m3gim-dataset.jsonld> ; source revision, licence and scope:
<https://chpollin.github.io/summer-school-musicology-2026/downloads/m3gim-dataset-source.txt>
 - The complete published graph contains 4,321 top-level nodes, including 1,007 records. These counts are not performance or person counts. Nested statements retain source relationships; inspect their meaning before aggregating them.

Templates for Knowledge Documents

data.md

- Subject and sources: identify the selected materials, source IDs, filenames and provenance.
- Model: describe documents, pages and passages for an edition, or the relevant entities and relations for a dashboard.
- Limitations: record uncertain readings, missing values and the scope of evidence. Place mentions do not establish attendance.
- Project-specific source selection and findings: {{todo}}

research.md

- Question: state what you want to investigate. Example: which places are mentioned in the selected documents, and in what context?
- Method: check transcription against the PDF and read place mentions in their surrounding passages. Adapt the method to the dashboard's source data when choosing that path.
- Interpretation: distinguish explicit source statements from inferences and record unresolved questions.
- Your research question and method: {{todo}}

specification.md

- Requirements: define observable behaviour. An edition might display transcription beside its source and let users inspect a selected passage.
- Scope and constraints: one main view, one core interaction and the local, dependency-free HTML/CSS/JavaScript setup.
- Acceptance criteria: passages or records link to evidence; uncertain readings remain visible; original files and source IDs are preserved.
- Agreed first milestone and source-based acceptance example: {{todo}}

Organise Your Project Folder

- sources/: unchanged originals, such as PDFs or the downloaded dataset.
- data/: prepared research data, including transcriptions, TEI or processed dashboard data. Create only the subfolders needed.
- knowledge/: data.md, research.md and specification.md. Update these documents as understanding and requirements change.
- website/: index.html, style.css and script.js, plus any local assets required by the first version.

The HTML file supplies the page structure, CSS controls its appearance and JavaScript supplies interactive behaviour. Keep generated application files distinguishable from the source materials. Relative file paths allow the project folder to remain usable when moved to another computer.

Implement, Verify and Refine

Run the prompts one at a time and review each result before continuing. Begin only after the first milestone and its acceptance criteria have been agreed.

1. Implement: Implement the agreed milestone using knowledge/. Keep original sources unchanged. Start the website locally and give me the browser URL.
2. Verify: Check the result against knowledge/specification.md and the source data. Report what passed, failed or remains unverified. Give me a short checklist for scholarly review.
3. Fix and refactor the code: Fix confirmed errors first and verify the fixes. Then simplify the code and remove duplication without changing behaviour. Repeat affected checks.
4. Update and refactor the knowledge: Update knowledge/ with what we learned. Remove duplication and flag contradictions or outdated assumptions. Keep data findings in data.md, research decisions in research.md and requirements in specification.md. Distinguish verified findings from open questions.

5. Plan the next iteration: Compare the current result with the research question and requirements. Propose the next small milestone and explain what it would improve.

A data-checking script can be useful if your chosen task introduces structured records. Its rules must follow the actual fields and allow legitimate missing values. Check it with one known valid input and a deliberately incomplete copy. A successful structural check establishes only the conditions implemented in the script; source interpretation requires comparison with the original evidence.

Document an observed defect with the selected document, the expected result and what appeared instead. Give this evidence to the agent and request a focused correction. Distinguish checks executed by the agent from your own browser inspection and from checks that remain open.

Present Your Promptotype

- Explain your data and goals. Demonstrate the supported research operation with one source example.
- Describe what was useful or problematic when working with LLMs and agentic systems.
- Explain how far you could act as the expert in the loop by understanding, judging and articulating project knowledge.
- Discuss what you learned about investigating mobility, networks and gender, and how you might use the approach in your own research.
- State unresolved limitations and checks that remain open. Keep the application, source references and project knowledge together.

Appendix

The Second Brain Vault as a Working Environment for Humans and Agents

- Obsidian stores Markdown files in a local vault. HOME and Maps of Content connect knowledge areas in the demonstrated Second Brain Vault.
- Projects, concepts, routines and agent rules remain addressable as plain text. Navigation supports retrieval; Git and document rules support versioning and maintenance of canonical knowledge.
- The slide shows an existing working environment. Adopting its full organisation is not required for the three-document project exercise. <https://obsidian.md/>

MCP, RAG, Skills and Project Instructions

- MCP connects a harness to external tools and resources; RAG retrieves source material for a model's context.
- Skills package reusable instructions and supporting resources. CLAUDE.md supplies project instructions for Claude Code; AGENTS.md supplies instructions for compatible agents.
- Check which files your harness actually loads. Merely placing a file in a project does not establish that it was read.

Two Agentic Research Data Workflows & Epistemic Infrastructures

- Printed sources: OCR, layout, TEI and named entities. Frontend: <https://chpollin.github.io/zbz-ocr-tei/> ; repository: <https://github.com/chpollin/zbz-ocr-tei>
- Manuscripts: HTR with Gemini 3.1 Flash Lite and curation, as presented on the slide. Frontend: <https://chpollin.github.io/szd-htr-ocr-pipeline> ; repository: <https://github.com/chpollin/szd-htr-ocr-pipeline>
- Compare how each workflow connects inputs, generated readings and scholarly correction. These are examples for discussion, not extra mandatory implementations.

Optional Preparation: PDF Conversion

- The optional Python-in-VS-Code and AI-harness packages contain the same PDF conversion script and seven PDFs. Run one yourself or delegate it, then compare image counts, order, orientation and readability against the originals.
- Follow each package README. PDF preparation can require Python dependencies; this does not introduce dependencies into the final static website.
- Python package: <https://chpollin.github.io/summer-school-musicology-2026/downloads/python-vscode.zip> ; harness package: <https://chpollin.github.io/summer-school-musicology-2026/downloads/ai-harness.zip>

Optional Edition Guidance: Inspect Materials and TEI Display Rules

The following material expands the edition path. Select the guidance relevant to the agreed first milestone; it does not prescribe additional views or interactions.

- Optional reference transcriptions, TEI and PNG images:
<https://chpollin.github.io/summer-school-musicology-2026/downloads/m3gim-fulltext/m3gim-next-session.zip>

Begin with an inventory of the files actually supplied. Establish which image, transcription and XML represent the same document or page. Prefer explicit identifiers and references in the materials. A filename resemblance alone may be ambiguous. Report missing files or uncertain matches before building navigation.

Check whether the separate transcription and the text encoded in TEI agree. If they differ, record the difference and decide which version the reading view will display. Keep both originals accessible. The assignment does not authorise silently correcting a transcription or inventing a relationship between files.

TEI records textual structure and editorial information. The viewer needs explicit display rules for the elements present in the packet. The following mappings illustrate decisions to inspect; they do not claim that every element occurs in the supplied files.

- `<p>` represents a paragraph and can become a paragraph in the reading view.
- `<lb/>` marks the beginning of a source line. A source-oriented view can display it as a line break.
- `<pb/>` marks a page beginning in TEI generally. In this exercise's profile, one `<pb type="scan">` identifies one scan, including a double-page spread. Resolve its `facs` path to the corresponding PNG.
- `<unclear>` contains an uncertain reading when present. In this exercise's introductory TEI profile, uncertainty is retained as literal symbols such as `word[?]` and `[...]`. Preserve and explain those symbols in the viewer.

Inspect attributes as well as element names. For example, a `facs` value may refer to another TEI element rather than directly to an image file. A page break does not by itself prove which image belongs to that page. Elements that express a supplied reading, a gap or an editorial choice also require deliberate display rules if they occur. Parsing well-formed XML does not establish that those rules are faithful to the encoding.

A working page is the starting point for checking the viewer. Compare its behaviour with the agreed requirements and the source material.

- Correspondence. Select each supplied document and check that its image, text, identifier and XML link belong together. Check that switching documents does not leave content from the previous selection.
- Faithfulness. Compare a passage with its source. Check wording, reading order and the agreed treatment of lines, pages and editorial markers. Confirm that uncertainty remains visible.
- Interaction. Open a larger image and the original XML. Check document selection and keyboard navigation. Try an input with a missing reference in a separate test copy if the material allows it.
- Evidence. Ask which files changed, which checks the agent executed and which results you verified manually. A successful XML parse proves a different property from a correct image–text pairing.

Use an observation that another person could reproduce. The example below describes the form of useful feedback; replace its identifiers with an actual case from your viewer.

When I select the second document, the image changes but the transcription still belongs to the previous selection. Both should update to the selected document. Inspect the selection handler and the file mapping, fix the mismatch and check the switch in both directions. Keep the source files unchanged. Report the cause and the checks you actually performed.

After the revision, repeat the affected check. Keep a short account of the input relationship, one display decision, the observed defect and the verified change. This connects the agent's actions to an inspectable research result.

Appendix · Source Checks behind the M³GIM Example

These source-critical examples explain why an interface must preserve documentary scope and access to evidence. They concern M³GIM programme records and provide background for judging research views. Their occurrence and participation model is not a required input format for the TEI viewer exercise.

Check Names Against the Page

The comparison uses the cast page of source 3 and its catalogue entry.¹ The catalogue forms include Nisson, Ludiwig and Pilz, while the readings established from the programme are Nilsson, Ludwig and Pitz. Comparing the same name components keeps these spelling corrections distinct from a change in display order between surname-first and given-name-first forms. A catalogue record provides useful context, but agreement with it would not have exposed these differences. A documented correction retains the earlier value and the evidence for the revised reading. In this example, the spellings can be checked directly against the original programme page.

One Participation Can Contain Two Stage Roles

The programme for Götterdämmerung on 29 July 1953 assigns Waltraute and the second Norn to Ira Malaniuk. Our model represents this as one participation with two values in stage_roles. The English label Second Norn² is a translation used for display and is distinct from an exact quotation of the programme's wording. Source evidence preserves the original expression and its page reference. This case tests the unit of modelling. Two character assignments belong to one contribution by the same person to the same announced occurrence. The website must retain both roles together, and any participation count must follow that same definition.

Preserving Differences Between Sources

The Malaniuk example shows why source statements need to remain distinguishable. Source 10 includes Waltraute and the second Norn in a dated programme³. The season list Wer singt was omits the second Norn. That absence describes the coverage of the season

¹ Universitätsarchiv der Kunstuniversität Graz, UAKUG/NIM_137 3, scan p. 2. Compare the printed names with catalogue record NIM_137_3_22 in the [M³GIM archive dataset](#). The spellings visible on the programme are Nilsson, Ludwig and Pitz.

² Universitätsarchiv der Kunstuniversität Graz, Ira Malaniuk estate, UAKUG/NIM_137 10, Götterdämmerung programme, scan p. 2 (UAKUG_NIM_005_137_10.pdf). The cast list assigns "WALTRAUTE" and "2. NORN" to Ira Malaniuk. Second Norn is the English display label.

³ Universitätsarchiv der Kunstuniversität Graz, Ira Malaniuk estate. Compare UAKUG/NIM_137 10, scan p. 2, with UAKUG/NIM_137 11, scan p. 3, "Wer singt was?", row "Ira Malaniuk, München". The season list records Fricka, Waltraute and Brangäne; the daily Götterdämmerung programme also records "2. NORN".

list and does not cancel the dated programme's statement. Before resolving a difference, the comparison must establish what each document actually says and the scope of that evidence. The data can then retain both statements and document the omission. A conclusion about whether a person performed on a particular occasion would require evidence beyond this comparison of announced or season-level assignments.

References and Further Reading

- Anthropic (2024). Building Effective Agents. <https://www.anthropic.com/engineering/building-effective-agents>
- Anthropic (2025). Effective Context Engineering for AI Agents. <https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents>
- Zhong, Hailin and Shengxin Zhu (2026). AI Harness Engineering: A Runtime Substrate for Foundation-Model Software Agents. arXiv:2605.13357. <https://doi.org/10.48550/arXiv.2605.13357>
- Sapkota, Ranjan, Konstantinos I. Roumeliotis and Manoj Karkee (2026). AI Agents vs. Agentic AI: A Conceptual Taxonomy, Applications and Challenges. Information Fusion 126, 103599. <https://doi.org/10.1016/j.inffus.2025.103599>
- TEI Consortium. TEI P5 Guidelines, element reference. <https://www.tei-c.org/release/doc/tei-p5-doc/en/html/REF-ELEMENTS.html>
- TEI Consortium. <lb> (line beginning). <https://www.tei-c.org/release/doc/tei-p5-doc/en/html/ref-lb.html>
- TEI Consortium. <pb> (page beginning). <https://www.tei-c.org/release/doc/tei-p5-doc/en/html/ref-pb.html>
- TEI Consortium. <unclear>. <https://www.tei-c.org/release/doc/tei-p5-doc/en/html/ref-unclear.html>
- Unsworth, John (2000). Scholarly Primitives: What Methods Do Humanities Researchers Have in Common, and How Might Our Tools Reflect This? <https://johnunsworth.name/Kings.5-00/primitives.html>
- Borek, Luise, Quinn Dombrowski, Jody Perkins and Christof Schöch (2016). TaDiRAH: A Case Study in Pragmatic Classification. Digital Humanities Quarterly 10(1). <https://tadirah.info>
- Rhee, Hea Lim (2012). Modelling Historians' Information-Seeking Behaviour with an Interdisciplinary and Comparative Approach. Information Research 17(4). <http://informationr.net/ir/17-4/paper544.html>
- Schulhoff et al. (2024). The Prompt Report: A Systematic Survey of Prompting Techniques. <https://doi.org/10.48550/arXiv.2406.06608>
- Mei et al. (2025). A Survey of Context Engineering for Large Language Models. <https://doi.org/10.48550/arXiv.2507.13334>
- Allen et al. (2023). Knowledge Engineering Using Large Language Models. Transactions on Graph Data and Knowledge 1(1), 3. <https://doi.org/10.4230/TGDK.1.1.3>
- Russell, Stuart and Peter Norvig (2020). Artificial Intelligence: A Modern Approach. 4th edition. Pearson. <https://aima.cs.berkeley.edu/>
- Wooldridge, Michael and Nicholas R. Jennings (1995). Intelligent Agents: Theory and Practice. <https://doi.org/10.1017/S0269888900008122>
- Wang et al. (2023). Voyager: An Open-Ended Embodied Agent with Large Language Models. <https://arxiv.org/abs/2305.16291>
- OpenAI (2019). Emergent Tool Use from Multi-Agent Interaction. <https://openai.com/index/emergent-tool-use/>
- Wang, Lei, et al. (2024). A survey on large language model based autonomous agents. Frontiers of Computer Science 18, 186345. <https://doi.org/10.1007/s11704-024-40231-1>
- Starace, Giulio, et al. (2025). PaperBench: Evaluating AI's Ability to Replicate AI Research. Proceedings of ICML, PMLR 267, 56843–56873. <https://proceedings.mlr.press/v267/starace25a.html>
- Deng, Xiang, et al. (2025). SWE-Bench Pro: Can AI Agents Solve Long-Horizon Software Engineering Tasks? arXiv:2509.16941, version 2. <https://doi.org/10.48550/arXiv.2509.16941>
- Edwards, Nicholas, Yukyung Lee, Yujun Audrey Mao, Yulu Qin, Sebastian Schuster and Najoung Kim (2026). RExBench: Can coding agents autonomously implement AI research extensions? Proceedings of ACL, Volume 1: Long Papers, 16380–16417. <https://doi.org/10.18653/v1/2026.acl-long.745>

Xu, W., Huang, C., Gao, S., and Shang, S. (2025). LLM-Based Agents for Tool Learning: A Survey. <https://doi.org/10.1007/s41019-025-00296-9>

LangChain. The Anatomy of an Agent Harness.
<https://www.langchain.com/blog/the-anatomy-of-an-agent-harness>

Willison, Simon (2026). What is agentic engineering?
<https://simonwillison.net/guides/agentic-engineering-patterns/what-is-agentic-engineering/>