

PDAI: A Certified Document Infrastructure for Computational Historical Scholarship

From PDF to a Queryable, Audited, and Learning-Enabled Corpus

Gerd Graßhoff 

Humboldt-Universität zu Berlin, Deutsches Museum
gerd.grasshoff@hu-berlin.de

ABSTRACT

Computational epistemology — the systematic study of how knowledge claims in historical science can be evaluated computationally — requires primary sources that are queryable down to the page and sentence. This paper presents PDAI (Portable Document AI), a certified document infrastructure that transforms scanned scholarly editions into structured, queryable corpora with full production audit and learning-enabled quality assurance. The infrastructure has three layers: the PDAI format (a flat, predictable XML directory structure validated against a declared DTD grammar), the production pipeline (a Partitur-governed sequence from ingest through schema-first OCR, pilot-gated batch processing, and a mandatory certify-and-learn stage), and the contextual data stores (corpus DuckDB, lemma parquet postings, gazetteer, LWgrammar lexicon, entity registry) that make a certified page answerable to structured scholarly queries. The paper describes the nine structural categories of the PDAI page grammar and shows through five case studies how they capture the genuine typographic diversity of early modern scholarly editions — epistolary apparatus, astronomical ephemerides, cuneiform transliteration, manuscript catalogue entries, and philosophical correspondence. A page enters the corpus only after the audit gate: every defect is attributed to one of four root-cause layers, registered as a versioned learning case, and closed through an A/B-gated instruction improvement loop. The result is a certified chain from PDF to the ai-viewer endpoint where pages serve as Grounding Evidence for computational epistemological inquiry — citable, provenance-tracked, and regression-anchored.

KEYWORDS digital humanities · document processing · optical character recognition · computational epistemology · PDAI format · schema-first OCR · scholarly infrastructure · large language models · grounding evidence

Table of contents

1 Introduction: Documents as Grounding Evidence	2
2 The PDAI Format	3
2.1 The problem with PDF	3
2.2 The PDAI directory structure	3
2.3 The canonical page DTD	4
2.4 The page language: one core, collection wrappers	6
2.5 PDAI compared with alternative formats	6
3 Five Document Types: The Expressiveness of the Grammar	7
3.1 Euler’s Tidal Theory (E57): mathematical Latin treatise	7
3.2 Kepler’s Correspondence (KGW 13–19): epistolary apparatus	8
3.3 Astronomical Diaries (ADRT 1–7): cuneiform transliteration	8
3.4 Bischoff’s Handschriften-Katalog (medieval-sec): catalogue entries	9
3.5 The Euler-Goldbach Correspondence: segment-level assembly	10
4 The Production Pipeline	10
4.1 Architecture overview	10
4.2 Stage 1: Ingest and classify	11
4.3 Stage 2: Schema derivation	11
4.4 Stage 3: The pilot gate	12
4.5 Stage 4: Schema-first OCR batch	12

4.6 Stage 5: Certify and learn	12
4.7 Stage 6: Corpus completion	13
4.8 Stage 7: The ai-viewer endpoint	13
5 Certify and Learn	13
5.1 The four-layer fault model	13
5.2 The tag hierarchy protocol	14
5.3 Figure completion census	14
5.4 The fault dossier	14
5.5 The learning bridge	15
5.6 The Typst certification report	16
6 The Contextual Data Stores	16
6.1 Corpus DuckDB	16
6.2 Lemma parquet postings	16
6.3 Gazetteer	17
6.4 LWgrammar lexicon	17
6.5 Entity registry and partitur register	17
7 Case Studies at Scale	17
7.1 KGW 15: Certified ephemerides	17
7.2 Pirckheimer (354 letters): correspondence queries at 200 ms	18
7.3 Astronomical Diaries (ADRT 1–7): transliteration at scale	18
8 The Edition Registry and the Viewer Contract	18
8.1 The delivery contract	18
8.2 The edition viewer	18
8.3 Port registry	19
9 Implications for Computational Epistemology	19
9.1 The page as the unit of evidence	19
9.2 Grounding Evidence vs Sekundärevidenz	19
9.3 Reproducibility and provenance	20
9.4 Computational engagement with primary sources	20
10 Conclusion	20
11 Acknowledgments	21
Bibliography	21

1 Introduction: Documents as Grounding Evidence

The history of science is a record of claims. Kepler claims that planetary motion obeys an area law; Euler derives the shape of tidal forcing from a Newtonian potential; the Babylonian astronomical diaries record Jupiter’s daily position in degrees and seconds. These claims are embedded in primary sources — letters, treatises, observation records — most of which reached the digital age as scanned PDFs: visually faithful, computationally opaque.

GROUNDING EVIDENCE

Grounding Evidence is the direct textual or observational basis on which a historical knowledge claim rests — as distinct from secondary literature that interprets or summarises it. A page of Kepler’s correspondence is Grounding Evidence for claims about his reasoning; a monograph about Kepler is Sekundärevidenz. The distinction is operational: PDAI pages are structured to serve as Grounding Evidence, which means they must be citable at the sentence level, provenance-tracked, and answerable to structured queries without paraphrase.

A scanned PDF preserves what a document looks like. It cannot answer the question: *on how many pages of the Kepler correspondence does Kepler mention the law of refraction?* To answer this question computationally,

a system needs not images but structured transcriptions — transcriptions that record not just characters but the elements of scholarly apparatus: headers, sections, paragraphs, marginal notes, footnotes, figures, tables, mathematical expressions, and the epistolary skeleton of a letter. It needs those transcriptions validated against a declared grammar so that queries can be written against a stable schema. And it needs provenance: which model produced this transcription, under which instruction, on which date.

The Portable Document AI (PDAI) infrastructure addresses these requirements. This paper describes its format, its production pipeline, and the certify-and-learn system that makes a PDAI page a trustworthy unit of Grounding Evidence.

Three companion papers carry related material: the orchestration architecture behind the query pipeline is presented in [1]; the linguistic index layer (lemma tables, dependency parses, DuckDB query engine) in [2]; and the research platform that reads PDAI pages as evidence for scholarly operations in [3].

2 The PDAI Format

2.1 The problem with PDF

The PDF standard [4] solves a real problem: document appearance is preserved across platforms. This made it the canonical format for digitised historical collections. But the solution creates a different problem for computational scholarship.

A scanned PDF encodes position on a page, not structure. A footnote marker in the text and its referent at the page foot share no programmatic connection; a column in a table of astronomical data has no type. Even PDFs with embedded text layers from conventional OCR contain errors, lose structure, and cannot distinguish a marginal annotation from the main text. The consequence is that the scholarly question — what does this document say about this topic — can be asked of a PDF only through full-text string search, which neither understands structure nor tolerates the orthographic variation of historical texts.

The TEI guidelines [5] offer rich semantic encoding, but at a cost: TEI markup is written by human editors and is not automatically produced from scans. For large historical corpora — tens of thousands of pages across dozens of editions — TEI encoding is not feasible. What is needed is automatic production of structured XML from page images, at sufficient quality to serve as Grounding Evidence.

2.2 The PDAI directory structure

A PDAI document is a directory on the filesystem. The directory structure separates concerns while maintaining clear relationships between layers:

```
/Volumes/PDAI-1/pdai-collections/<collection>/<biblabel>/
  <biblabel>.pdf          original source PDF
  biblio.xml             bibliographic identity (biblio.dtd)
  pdai-config.json       harvest and processing record
  pages/
    page_NNN/
      page_NNN.png       page image at production resolution
      page_NNN.txt       raw OCR text (fastimage, used as signal only)
      page_NNN.xml       structured PDAI XML (the primary output)
  segments/
    letters/             materialised letter records
    sections/           materialised sections
  structure.xml          document-level assembly
  quality/
    <vol>_defects.json    structural audit results
    <vol>/
      tag_hierarchy_protocol.jsonl structural fingerprints for regression
```

learning_fault_dossier.json	fault cases for the learning loop
audit_report_<vol>.pdf	LettreAI Typst certification report

The separation is principled. Source images occupy `pages/`; OCR outputs are co-located with the page. `segments/` holds assembled records that span multiple pages. `structure.xml` is the document-level view. `quality/` is the audit and learning layer — it accumulates without touching production files.

This structure embodies two design principles: separation of concerns (no processing step overwrites source data) and accumulative growth (the quality folder grows as auditing deepens, without invalidating transcriptions).

2.3 The canonical page DTD

Every PDAI page is validated against `page.dtd` v4.0 (or a collection wrapper of it), maintained at `pdai/pipeline/ai-scholar/src/ai_scholar/schemas/page.dtd`. The DTD is the grammar contract: any query written against the PDAI schema can rely on the elements in Table 1 being present and structurally correct.

The canonical grammar declares nine structural categories:

Table 1 The nine structural categories of the canonical PDai page grammar. Every element in every PDai page is an instance of one of these categories.

Category	Elements	Role in scholarly apparatus
Page frame	<page pidx="...">, <meta>, <header>, <footer>	Page address and processing provenance
Running text	<p> , <section> , <parno> , <item>	Prose, numbered arguments, enumeration
Typography	<hi rend="...">	Emphasis (italic, bold, smallcaps, superscript, subscript, spaced)
Epistolary apparatus	<letter-head> , <sender> , <addressee> , <date> , <place> , <archive> , <letter-closing> <greeting> , <signature> , <address>	Letter structure and archival provenance
Mathematics	<equation display="..."> <math>	LaTeX-formatted inline and display expressions
Figures	<figure ref="..."> , <caption> , <description>	Illustrations with transcribed caption and description
Tables	<table> , <tr> , <th> , <td>	Tabular data (never flattened to <p>)
Marginalia	<marginal side="..." type="...">	Everything printed in a margin; markers line-number, folio, symbol; notes topic, reference, date
Footnotes	<footnotes> , <footnote> , <fn>	Page-bottom apparatus with inline markers

The <meta> element records the provenance of each page:

```
<meta model="qwen3.8:27b-mlx" engine="qwen3.8:27b-mlx"
  instruction_template="kepler-ocr"
  partitur="kepler-ocr@3f400c0df6fa"
  layers="page-language,base,qwen38"
  overlay="kepler-ocr.qwen38.j2"
  lang="orig"
  processed_date="2026-09-28"
  output_schema="kepler-prim/page.dtd v2.0" />
```

This provenance chain is unbroken: every page records which model and which composed instruction produced it. `partitur` is the instruction family and the first twelve hex digits of the SHA-256 of the

composed prompt, `layers` the layers it was composed from (below). A scholar who doubts a transcription can reproduce it with identical configuration. Detecting when reprocessing is warranted — because a model was updated or an instruction improved — is a query over the `meta` element's attributes.

2.4 The page language: one core, collection wrappers

Every PDAI grammar includes one core module, `pdai-core.mod`, which declares what all collections share: the margin, the processing record `<meta>`, figures and the renditions of `<hi>`; tables come from `pdai-table.mod`. The canonical `page.dtd` (v4.0) writes each of its content models as a parameter entity. A collection grammar in `config/page.dtd` is therefore a *wrapper*: it sets the entities it needs, declares only the elements the canonical page lacks, and includes `page.dtd`. The letter page grammar `page_letter.dtd` (v4.2) is such a wrapper too.

METHOD The extension rule

A collection grammar adds to the page language. It never removes or renames a core element or margin type, and it does not introduce record-level assembly into the page grammar. The page grammar records what is printed on this physical page; the record grammar (`letter.dtd`, `catalogue.dtd`, `tablet.dtd`) assembles the logical record from multiple pages.

Collection	Its own elements beyond the page language
<code>kepler-prim</code>	letter head and closing families, <code>equation</code> , <code>lineno</code> , <code>fnmark</code> ; KGW 19: <code>document-entry</code> and margin type <code>doc-id</code>
<code>copernicus-prim</code>	critical apparatus (<code>apparatus</code> , <code>app</code>), table of contents (<code>toc</code> , <code>toc-entry</code>)
<code>diaries</code>	position-ordered metadata blocks with <code>field</code> , <code>witness</code> , <code>coverage</code> , <code>comment</code> , <code>row</code> ; typed <code>section</code>
<code>medieval-sec</code>	catalogue <code>entryblock</code> with its fields, <code>register</code> ; margin type <code>entry-number</code>
letter collections	<code>page_letter.dtd</code> wrappers: <code>letter-part</code> , <code>head</code> , <code>body</code> , <code>closing</code> , <code>enclosure</code> and <code>apparatus</code> families

Margins. Everything printed in a margin is a `<marginal side="..." type="...">` standing directly before the first word of the printed line it is printed beside. The type is one word of a closed list declared in the core: the markers `line-number`, `folio`, `symbol` and the notes `topic`, `reference`, `date`. A collection may add a type only through the core's extension entity. The same list is read from the grammar by the OCR prompt, the normalizer (which maps the aliases `note`, `marginal-note`, `label`, `gloss` to `topic`), the audit (`SCH-MARGINAL-TYPE`) and the viewer, so the four cannot disagree.

Prompts. An OCR prompt is composed in fixed layers: the page-language fragment, rendered with the vocabulary of the collection's grammar; the collection base; an optional volume layer; the engine overlay, which holds only corrections measured for that engine. The composed prompt is the *partitur* stamped on each page (`<meta partitur="..." layers="...">`).

The flattened grammars are deployed with the collection; a census over all 159,170 page files on the production volume found 93,279 valid before and 95,751 after the restructuring, with no document losing a valid page.

2.5 PDAI compared with alternative formats

Feature	PDF	TEI-XML	PDAI
Visual fidelity	Excellent	Variable	Good (via images)
Semantic structure	Minimal	Excellent	Excellent

Feature	PDF	TEI-XML	PDAI
Automatic production from scans	No	No	Yes
AI processable	Poor	Moderate	Excellent
Progressive enhancement	No	Manual only	Automated agents
Provenance tracking	None	Manual	Automatic per page
Schema validation	No	DTD/Schema	DTD throughout
Interactive scholarly viewer	Reader apps	Custom	Built-in
Paragraph-level citation	No	Manual	Manifest-driven

PDAI does not replace TEI for critical editions requiring diplomatic accuracy at the level of individual manuscript readings; nor does it replace PDF for visual reproduction. Its position is the gap between the two: automatic production at scale, with sufficient structural precision for computational scholarly inquiry.

3 Five Document Types: The Expressiveness of the Grammar

The nine structural categories of the PDAI grammar are not abstractions: they are derived from the actual typographic practice of historical scholarly editions. This section shows the grammar at work across five document types that represent the range of PDAI collections.

3.1 Euler's Tidal Theory (E57): mathematical Latin treatise

Leonhard Euler's *Inquisitio Physica in Causam Fluxus ac Refluxus Maris* (1740, Euler Archive E57) is a 105-page Latin mathematical treatise in 8 chapters with 130 consecutively numbered paragraphs [6].

A page from this treatise activates four structural categories: running text (<p>, <parno>), mathematics (<equation>), figures, and footnotes.

```
<page pidx="025">
  <meta model="qwen3.5:cloud" lang="la"
    output_schema="page" validation_status="valid" />
  <header>CAPUT TERTIUM</header>
  <main>
    <parno>47.</parno>
    <p>Quantitas vis sollicitantis $F$ quae lunae attribui debet<fn>1</fn>,
      si distantia lunae a terra $D$ ponatur, sequenti formula exprimitur</p>
    <equation display="block">
      
$$F = \frac{3M\rho}{2D^3} \left( 3\cos^2\phi - 1 \right)$$

    </equation>
    <p>ubi $M$ denotat massam lunae, $\rho$ radium terrae, $\phi$ distantiam
      a polo...</p>
    <figure ref="fig_p025_n_1">
      <caption>Fig. 4: Schematismus virium lunarium</caption>
      <description>Diagram showing the geometry of lunar tidal forcing.
        A circle labelled T (Terra) at centre. The Moon M is positioned
        to the right. Force vectors F are drawn tangentially at three
        surface points, with angles phi marked. A vertical line through
        the poles divides the diagram symmetrically.</description>
    </figure>
  </main>
  <footnotes>
    <footnote data-fn="1">Cf. Newton, Principia Mathematica, Lib. III, Prop.
    XXXVI.</footnote>
```

```
</footnotes>
</page>
```

The `<description>` element is not a transcription of printed text — it is a model-generated geometric description of the diagram, authored to enable query and display. This is a PDAI-specific enrichment: the figure census (see § 5.2) measures whether every figure has reached this level of completeness.

3.2 Kepler’s Correspondence (KGW 13–19): epistolary apparatus

The Kepler Gesammelte Werke correspondence volumes (KGW 13 through 19) contain Kepler’s letters from 1590 to 1630 [7]. Their structure is epistolary: formal headers, archival references, letter bodies, and closing signatures. The `page_letter.dtd` variant captures this:

```
<page pidx="038">
  <meta output_schema="page_letter" model="qwen3.8:27b-mlx" lang="de" />
  <letter-head>
    <reference>19. KONRAD VISCHSES AN KEPLER</reference>
    <sender>Konrad Vischses</sender>
    <addressee>Johannes Kepler</addressee>
    <place>Tübingen</place>
    <date>14./24. Mai 1595</date>
    <archive>Pulkowo, Sternwarte, Kepler-Hs. III, f. 42r</archive>
  </letter-head>
  <main>
    <p>Salutem a Deo patre per Christum filium eius unicum Dominum nostrum
      <fn>1</fn>. Literas tuas sex ante hebdomadas accepi ...</p>
    <marginal type="folio">42r</marginal>
  </main>
  <letter-closing>
    <greeting>Vale in Christo</greeting>
    <place-date>Tubingae raptim 14./24. Mai 1595</place-date>
    <signature>Tuus M C Vischses</signature>
  </letter-closing>
  <footnotes>
    <footnote data-fn="1">Formel aus dem Briefeingang; vgl. KGW 13, Nr. 19.</
  footnote>
  </footnotes>
</page>
```

The epistolary apparatus — `<sender>`, `<addressee>`, `<date>`, `<archive>` — enables queries that the text alone does not support: *in how many letters between 1610 and 1615 does Kepler discuss refraction?* This requires structured access to both the letter header (for the date filter) and the letter body (for the subject). After segmentation and stanza ingest, the PDAI query layer answers this query in under 200 ms across 1,200 letters.

3.3 Astronomical Diaries (ADRT 1–7): cuneiform transliteration

The *Astronomical Diaries and Related Texts from Babylonia* [8] are a continuous record of celestial and terrestrial observations spanning 750–60 BCE. The PDAI `diaries` collection covers volumes I through VII (~6,500 pages). Their format is not epistolary but tabular-transliteration: Sachs and Hunger’s edition prints the transliteration of cuneiform tablets alongside scholarly commentary.

The `diaries` grammar adds position-ordered metadata blocks and typed sections to the page language; a transliteration page reads:

```

<page pidx="042">
  <meta output_schema="diaries/page.dtd v2.0" model="qwen3.8:27b-mlx" lang="akk" />
  <header>DIARY -651 B</header>
  <main>
    <section type="lunar">
      <p style="tr"><marginal side="left" type="line-number">1'</marginal>[Month I,
        (the 1st of which was identical with)] the 30th (of the preceding month).
        [Night of the 1st,] overcast. [The river level] measured 2 cubits 8
        fingers.</p>
      <p style="tr"><marginal side="left" type="line-number">2'</marginal>Month II,
        (the 1st of which was identical with) the 30th, evening watch, the moon
        was 1 cubit in front of <hi rend="italic">Virginis</hi>.</p>
    </section>
  </main>
</page>

```

Marginal line numbers (`<marginal type="line-number">`) are separated from the text without breaking the reading order. They attach to the line they label without disrupting paragraph flow — a requirement that initially produced VIS-MARGINAL-GUTTER defects (the line number overflowed into the main text column) before the margin rules of the shared stylesheet were calibrated. The tag hierarchy protocol (§ 5.3) detected the regression when the CSS fix was first applied to a subset of pages.

■ 3.4 Bischoff's Handschriften-Katalog (medieval-sec): catalogue entries

Bernhard Bischoff's *Katalog der festländischen Handschriften des neunten Jahrhunderts* (3 volumes, 1,641 pages) is a dense scholarly catalogue: each entry identifies a Latin manuscript, its present location, foliation, script type, and Bischoff's attribution. The entry structure extends the page language with the catalogue's own `<entryblock>` and its fields (`<shelfmark>`, `<foliation>`, `<contents>`, `<script>`, `<literature>`, `<origin>` ...).

A single catalogue entry spans between one and several pages; the page grammar records the entry block, and `catalogue.dtd` assembles the complete entry:

```

<page pidx="203">
  <meta output_schema="medieval-sec/page.dtd v2.0" model="qwen3.8:27b-mlx"
  lang="de" />
  <header place="MÜNCHEN">203</header>
  <main>
    <entryblock position="1" type="main" state="complete">
      <number value="2341">2341</number>
      <shelfmark>München, BSB, Clm 14742</shelfmark>
      <foliation>ff. 1–48</foliation>
      <contents><author>Isidorus Hispalensis</author>, <title>Etymologiae</title>,
      lib. I–III.</contents>
      <script>Bairisch-alemannische Minuskel</script>
      <literature>Bischoff 1980, Nr. 3241</literature>
      <origin place="Regensburg" date="IX. Jh., 2. Hälfte">— Regensburg, IX. Jh., 2.
      Hälfte —</origin>
    </entryblock>
  </main>
</page>

```

The `<shelfmark>` element enables queries by library and signature across the full catalogue: *how many manuscripts does Bischoff attribute to Regensburg between 820 and 860?* This query is answered by the stanza

entity layer (which recognises place names) and the letter register (adapted for catalogue IDs), not by string matching.

■ 3.5 The Euler-Goldbach Correspondence: segment-level assembly

The correspondence between Euler and Christian Goldbach (1729–1764) comprises 196 letters spanning the development of number theory, analysis, and mathematical physics [9]. This edition demonstrates segment-level processing: letters span multiple pages, and the query layer must be able to treat a complete letter as a unit.

The segmenter agent analyses all page XML, identifies letter boundaries, and produces `manifest.json`. A materialisation pass then assembles complete letter XML from the relevant page blocks. The `<ai-query-chat>` web component in the viewer uses the manifest to navigate directly to any of the 196 letters.

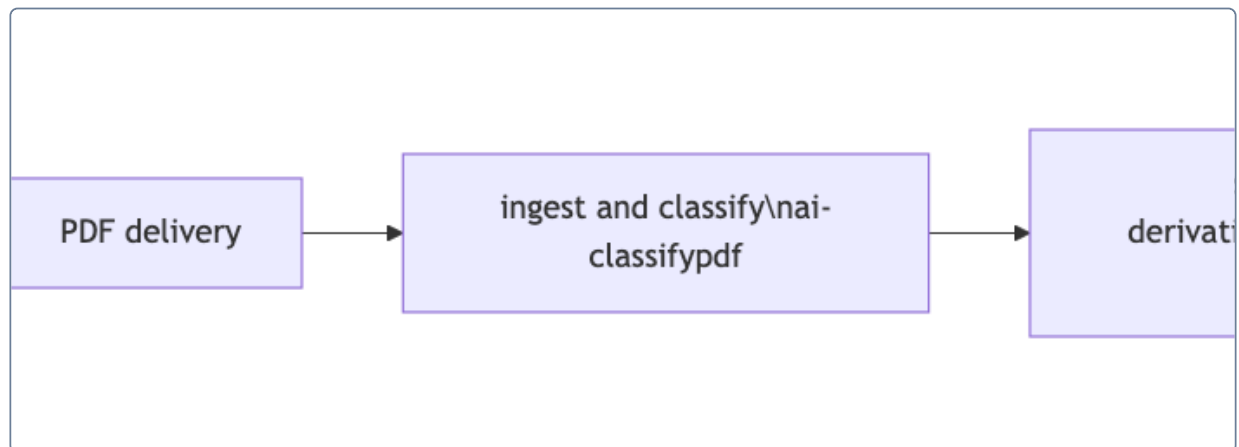
```
{
  "segments": {
    "letters": {
      "items": [
        {
          "id": "42",
          "from": "Goldbach",
          "to": "Euler",
          "date": "1742-06-07",
          "place": "Moskau",
          "pages": [87, 88, 89],
          "incipit": "Ich habe neulich ein Theorem gefunden ..."
        }
      ]
    }
  }
}
```

Letter 42 — Goldbach’s letter to Euler of 7 June 1742 — is the letter in which Goldbach states the conjecture that bears his name. Its first page is `page_087.xml`; the segmenter identifies the boundary at `page_089.xml`. A query for *letters in which Goldbach discusses prime sums* retrieves this letter at the sentence level, within 100 ms, without requiring a model to read 196 letters.

■ 4 The Production Pipeline

■ 4.1 Architecture overview

The PDAI production pipeline is a Partitur-governed sequence of seven stages, formalised as `pilot-gated-ocr-pipeline v1.1.0`, registered at goal path `pipeline/ocr/pilot-gated` in the central Partitur register. The Partitur is a plan with declared free inputs (`collection`, `volume`, `primary_model`, `dtd_path`); concrete values are supplied through a setting file before execution. The Conductor never executes an unbound Partitur.



Each arrow is a concrete command or service call. Each node is a system component with defined inputs, outputs, and failure modes.

4.2 Stage 1: Ingest and classify

The source PDF or scanned image folder is ingested by `ai-classifypdf ingest`. Page images are extracted at production resolution (typically 300 DPI for printed text, 400 DPI for manuscripts). The document is classified into one of four material classes:

- `printed-scholarly`: the standard path, using `qwen3.5:cloud` as OCR model
- `special-script`: Fraktur, Greek, cuneiform transliteration — routed to a learned script template via the script gate in `ai-scholar`
- `handwritten-archival`: personal correspondence at scale — `kimi-k2.7-code:cloud`
- `catalogue`: structured entry format — escalates to the schema-first derivation

Classification produces `ocr_classification.json` and `collection_profile.yaml`, which the OCR stage reads automatically. Classification is **always before OCR**; no production OCR runs without a completed `ocr_classification.json`.

4.3 Stage 2: Schema derivation

For formalised editions, the schema derivation reads the printed front matter of the volume and derives the record schema before any OCR. Every schema element is graded by evidence:

- `front-matter`: cited from the title page or the edition’s introduction — authoritative
- `facsimile-only`: observed in page images but not described in the front matter — provisional
- `unattested`: hypothesised but not observed — must not be included

The derivation produces `config/page.dtd`, `config/<record>.dtd`, `config/audit_rules.yaml`, and `config/page.css`. These four files constitute the schema package. No production OCR runs without a complete schema package.

METHOD Schema before OCR

The schema is derived from the printed document **before** OCR is run. An element found in the printed front matter is treated as authoritative and included; an element observed in the facsimile but not described is provisional; an element neither described nor observed must not be invented. This discipline prevents schema drift: the OCR model is not free to introduce structural categories that the edition’s editor did not declare.

4.4 Stage 3: The pilot gate

For any document of more than 50 pages, the pilot gate is mandatory before batch OCR is unlocked. Five pages are drawn as a reproducible random sample spanning the beginning, middle, and end of the document, with diversity across layout types: running text, headers, tables, figures, and indexes.

The pilot sample is audited in three independent passes:

1. **DTD validation:** every pilot page must be well-formed XML and valid against `page.dtd`. Zero structural blockers are required.
2. **Declarative rule conformance:** `audit_rules.yaml` checks are applied — vocabulary sets, table cue rules, forbidden tokens. Zero blockers required.
3. **Visual facsimile comparison:** the rendered page in `ai-viewer` is compared against the facsimile scan. VIS-* defects (table flattened to prose, marginal gutter overflow, hyphenated word split across two `<p>` elements) are recorded and attributed to one of the four root-cause layers.

If any blocker is raised, the improvement loop runs before batch unlock: the Jinja2 prompt overlay is revised, the normalizer regex is updated, or `page.css` is calibrated. The sample is re-audited until 100% PASS (0 blockers).

4.5 Stage 4: Schema-first OCR batch

The full volume is processed by the schema-first runner. The primary model on this hardware configuration (M1 Ultra Mac Studio) is `qwen3.8:27b-mlx`, a 27-billion-parameter multimodal model running on Apple MLX. Its validated parameter set is:

Parameter	Value	Rationale
<code>num_batch</code>	512	Reduces prefill latency from 26 s to 0.1 s
<code>num_ctx</code>	8192	Sufficient for one PDAI page
<code>num_predict</code>	2048	Avoids truncation on dense tables
<code>reasoning</code>	false	Required: thinking leaks into output
Workers	1	Optimal: 2 workers do not increase throughput on this hardware

Fallback: `kimi-k2.7-code:cloud` via Ollama's `:cloud` routing, which matches Gemini quality on dense scholarly text and cuneiform transliteration at 5–15× lower throughput. For handwritten archival correspondence at scale, `kimi-k2.7-code:cloud` is the primary model.

The OCR runner produces one `page_NNN.xml` per page, writes a JSONL telemetry file, and serves a live auto-refreshing dashboard at `http://localhost:<port>/ocr-monitor?volume=<vol>`. Volume succession chains run automatically (`chain_kepler.sh`: KGW 13 → 14 → 15 → 16 → 17 → 18), each volume executing its own pilot gate independently before batch unlock.

4.6 Stage 5: Certify and learn

The `certify-and-learn` stage is the post-batch entry in the Partitur and is **not optional**. The command is:

```
pdai-audit --collection <name> --vol <vol> --all --emit-annotations
```

`--all` runs five sub-stages in sequence. Each is described in detail in § 5.

The certification criterion is: `blockers == 0 AND vis_blockers == 0`. A volume that fails certification is not registered in the edition registry and does not appear in the `ai-viewer` endpoint.

4.7 Stage 6: Corpus completion

After certification, `pdai-complete --collection <name>` runs the indexing stages in dependency order:

1. Structure: `ai-scholar segment + materialise` → `structure.xml`, `segments/`
2. Stanza ingest: `ai_lemma_index stanza_ingest` → corpus DuckDB (sentences, entities)
3. Lemma index: `ai_lemma_index build_index` → `lemma_index.parquet` (postings)
4. Gazetteer: `ai_lemma_index build_gazetteer` → gazetteer entries in DuckDB
5. LWgrammar: `ai_lemma_index lwgrammar_build` → `<collection>_lwgrammar.duckdb`
6. ChromaDB: `ai_lemma_index chroma_build` → per-collection BGE-M3 vector store

The `pdai-complete status` command writes `status.json` to the collection root on PDAI-1. This file is the readiness manifest: a collection entry shows which stages are complete and whether the collection is ready for query.

4.8 Stage 7: The ai-viewer endpoint

A certified, indexed collection is served by the `ai-viewer` instance for its edition at its registered port. The viewer provides:

- **Dual-pane split view:** facsimile image alongside the structured transcription, sharing one page address (`pidx`)
- **Segment navigation:** the segment dropdown (letters, sections, paragraphs) is driven by `manifest.json` and allows direct navigation to any letter or section without knowing which page it occupies
- **Schema-aware rendering** for every collection: the shared book sheet (`pdai-page.css` and genre sheets) and the renderer `repro-xml.js`, which preserves empty elements and marks every element with `data-el` and every attribute as `data-<attribute>`; the collection `page.css` is the last sheet and sets measures and typography. Margins are placed in the margin at the height of their line
- **The `<ai-query-chat>` web component:** a Shadow DOM component, self-contained with all CSS bundled inline, that embeds the full query interface (query, evidence, solution path) within the viewer page, firing `navigate` events to open the relevant page in the viewer
- **The integrated `/audit` page:** a live quality-scoring panel for the collection's golden test set, visible in every viewer instance

5 Certify and Learn

5.1 The four-layer fault model

Every defect identified during `pdai-audit` is attributed to one of four root-cause layers. This attribution is the routing signal for the learning loop: a fault attributed to Layer 1 triggers a patch to the engine overlay of the composed prompt; a fault attributed to Layer 3 triggers a patch to the stylesheet (shared geometry in `pdai-page.css`, measures in `page.css`).

METHOD Four-layer root-cause attribution

Every fault case is attributed to one of four layers.

Layer 1 — Prompt / Vision LLM: the composed prompt (page language, collection base, engine overlay) governing the OCR model's output. Faults here: missing `<description>` in figures, wrong element choice for a structural pattern, `<think>` leakage, table flattened to prose.

Layer 2 — Runner Normalizer: the Python `_clean_xml()` function that post-processes raw model output. Faults here: unjoined hyphen at line end, unclosed tag from token limit, HTML entity in text content.

Layer 3 — Typographic CSS: the shared `pdai-page.css` and the collection's `config/page.css` governing display in `ai-viewer`. Faults here: a marginal in the text column, line-break fragmentation visible in rendering, wrong element spacing.

Layer 4 — DTD / Record Schema: the grammar files. Faults here: element not declared, attribute value outside the closed vocabulary, `<marginal>` type unrecognised.

A fault attributed to the wrong layer cannot be resolved: a CSS fix for a prompt omission will not affect newly generated pages, and a prompt fix for a CSS gutter overflow will not affect existing transcriptions.

5.2 The tag hierarchy protocol

The tag hierarchy protocol is the zero-regression anchor of the pipeline. For every page in the volume, the protocol logs a content-free structural fingerprint: the S-expression tree of XML element names and nesting order, with no text content, and a SHA-256 hash of that S-expression.

```
page_042: (page (meta) (header) (main (parno) (p) (equation) (figure (caption)
(description)) (p) (p)) (footnotes (footnote)))
sha256: a3f7b91c4d8e2f50...
```

When a prompt overlay is revised, `diff_tag_protocols(before, after)` compares structural hashes across all pages of the volume in under 50 ms. A fix that correctly joins hyphenated words across `<p>` boundaries shows no structural change. A fix that inadvertently drops all `<table>` elements on 34 pages shows as a mass structural delta: those 34 pages lose the `(table ...)` node from their S-expression hash.

The protocol is therefore a necessary condition for promoting an instruction change: any change that alters the structural fingerprint of a page that was previously correct must be investigated before promotion.

5.3 Figure completion census

Every `<figure>` element in the volume is audited for three completion components: `@ref` (required attribute linking to the extracted image file), `<caption>` (transcribed printed caption), and `<description>` (a geometric or diagrammatic description authored by the OCR model).

Completion is expressed as four states:

- `complete_full`: both `<caption>` and `<description>` present — standard for figures with a printed caption
- `complete_desc`: `<description>` only — standard when the printed page has no explicit caption (common in the Astronomical Diaries)
- `caption_only`: `<caption>` present but no `<description>` — a repair target; the model transcribed the caption but did not author a description
- `empty`: neither element — a blocker registered as a fault case in the dossier

For a certified volume, every figure must be at `complete_full` or `complete_desc`. The figure census in the LettreAI Typst report shows the distribution across all pages and flags the page indices with `empty` figures for targeted reprocessing.

5.4 The fault dossier

The fault dossier (`learning_fault_dossier.json`) is the machine-readable artefact consumed by the learning loop. It assembles all non-trivial defects into structured `FaultCase` records. Each record carries:

- Collection, volume, and page index (`pid`) as coordinates
- Defect code (`SYN-*`, `SCH-*`, `VIS-*`, or composite)
- A one-sentence fault summary and a precise fault description written to enable a subsequent learning iteration to address the defect without regression
- Root-cause layer (1–4) and the specific system component at fault
- A targeted fix directive naming the file to patch and the rule to add
- A list of regression anchor page indices — pages that passed the audit and must not be degraded by the fix

A typical fault case:

```
{
  "fault_id": "FAULT-kgw15-p234-002",
  "collection": "kepler-prim",
  "volume": "kgw-15",
  "page": "p234",
  "defect_code": "VIS-TABLE-FLATTENED",
  "fault_summary": "Astronomical ephemeris table on p234 rendered as consecutive <p>
elements.",
  "fault_description": "Page 234 contains a 12-column trigonometric table (sinus
versus / sinus rectus at 1° increments). The model rendered all 60 rows as <p>
elements with tab-separated values. The table cue (vertical alignment and column
headers) was not detected. The <table> element with <tr>/<td> structure was not
generated.",
  "root_cause_layer": 1,
  "root_cause_detail": "Prompt overlay does not specify the column-count threshold
for mandatory <table> generation.",
  "targeted_fix": {
    "target_file": "config/kepler-ocr.qwen38.j2",
    "rule": "Add: if page contains a grid with 4 or more aligned columns, always
produce <table><tr><td> structure, never <p> with tab separators.",
    "regression_anchor": "p234"
  },
  "regression_anchors": ["p012", "p067", "p189"],
  "severity": "blocker",
  "xml_evidence": "<p>0°\t0\t0\t0\t...\t</p><p>1°\t175\t1745\t29\t...\t</p>"
}
```

5.5 The learning bridge

When `--emit-annotations` is passed to `pdai-audit`, the learning bridge (`learn_bridge.py`) posts every fault case to the ai-query server's annotation endpoint (`POST /api/annotations`, port 5113). The annotation carries the fault's coordinates, the observed defect, and the targeted fix directive.

METHOD The learning bridge iron rule

Every OCR defect that reaches the fault dossier must enter the annotation system and be addressed through the A/B-gated learning loop. No ad-hoc patch to a prompt or normalizer is permitted outside the versioned cycle. Code-side probes are supporting signal only; the responsible SKILL.md or schema file is the patch target.

The annotation payload is always persisted locally before the HTTP post, so no fault data is lost when the ai-query server is unreachable. Re-ingestion is idempotent: the annotation table deduplicates on (subagent, raw_query).

The learning loop then:

1. Proposes a patch to the instruction file named in `instruction_ref`
2. Stores the patch as a candidate version in `instruction_versions`
3. Re-runs the golden test set on the regression anchor pages before and after
4. Promotes the version if the pass rate improves with no structural regression (verified by `diff_tag_protocols`)
5. Rejects and retains the candidate for forensics if it does not improve

This loop is bounded (at most 5 iterations per fault objective) and terminates at the improvement threshold. It does not run open-ended.

5.6 The Typst certification report

The LettreAI Typst report (`audit_report_<vol>.pdf`) is the human-readable output of the `certify-and-learn` stage. It is generated in the LettreAI theme (Oxford blue, gold, and sandstone palette) and contains:

- **Executive summary:** overall status (CERTIFIED / BLOCKED), page count, well-formed count, blocker count
- **Structural defect register:** all SYN-, CHR-, SCH-* defects tabulated by page and severity
- **Visual defect register:** VIS-* defects with root-cause attribution
- **Figure completion census:** four-state distribution across all figures in the volume
- **Tag hierarchy protocol summary:** structural hash coverage and the deviation count from the baseline
- **Fault dossier:** all fault cases grouped by root-cause layer, with targeted fix directives and regression anchors
- **Recommended action plan:** ordered by layer (Layer 1 first, Layer 4 last), with file paths and rule text
- **CSS coherence:** selector coverage analysis showing elements and attribute values with no CSS rule

6 The Contextual Data Stores

A certified PDAI page is the ground layer. The contextual data stores are what make it queryable. After `pdai-complete --collection <name>`, a collection maintains the following stores:

6.1 Corpus DuckDB

The collection's DuckDB database (at `/Volumes/PDAI-1/pdai-collections/<name>/<name>.duckdb`) is the relational spine of the corpus. Its tables include:

- `documents`: one row per document with bibliographic metadata and `pidx` range
- `letters_resolved`: for correspondence collections — one row per letter with sender, recipient, date, place, and the spanning `pidx` range
- `ent_canonical`: canonical entity forms with their collection-specific aliases
- `stz_sentences`: sentences from the Stanza linguistic parse, keyed by the triple `(doc_id, pidx, sent_idx)` — the sentence key
- `stz_entities`: named entities identified by Stanza/GLiNER, with surface form and canonical link

METHOD The sentence key invariant

A PDAI sentence is keyed by the triple `(doc_id, pidx, sent_idx)`. The `sent_idx` restarts on every page, so a key without `pidx` is not a sentence key — it undercounts sentences and invents cross-page co-occurrences. Every query that refers to sentence position must include `pidx`.

The distinction between correspondence partners (senders and receivers of letters) and persons merely mentioned in text bodies is maintained by the schema: `letters_resolved.sender/.recipient` for the epistolary channel, `stz_entities.text` for body mentions. A query that conflates the two channels produces wrong results.

6.2 Lemma parquet postings

The global inverted index (`lemma_index.parquet`) is the fast filter of the query pipeline. It maps each lemma to the set of postings `(doc_id, pidx, tid)` — document, page index, and token position — where it appears. Postings for a query's conjunctive-normal-form lemma expression are retrieved in milliseconds using DuckDB's columnar scan over the parquet file.

Two counting modes are distinguished:

- `canonical`: the lemma only (as assigned by Stanza's Latin or German model)

- `family`: canonical plus OCR variants registered in the collection's alias table

These modes give systematically different counts and must be declared with every cited frequency. The token-position column in `lemma_index.parquet` is `tid`, not `token_idx`.

6.3 Gazetteer

The gazetteer stores person names, place names, and domain-specific concepts with their collection-specific variant spellings. It is built by `ai_lemma_index build_gazetteer` from the Stanza entity output and the edition's index (when available). The gazetteer serves the `entity-fast-filter` subagent, which filters candidate documents by person or place name across all `entity_variants` spellings before any full LLM pass.

Person names must never appear in `concept_terms` or terminology signals. Persons are on the entity-registry channel only; concept terms are domain vocabulary (mathematical, astronomical, philosophical). This channel separation is enforced by the query pipeline's entity fast-filter, which returns zero results if a person name is submitted to the concept-term lookup.

6.4 LWgrammar lexicon

The LWgrammar (`<collection>_lwgrammar.duckdb`) stores OED-form usage entries for lemmas that appear frequently enough to merit a dictionary treatment (threshold: ~10 occurrences). Each entry has a headword, forms, numbered senses, and dated quotations linked to their `(doc_id, pidx, sent_idx)` sentence keys.

Entries are built by `ai_lemma_index lwgrammar_build`, which calls the allowlisted Gemini model following the OED instructions and the reference `time` entry. The lexicon is the resource for the dictionary operation in AI Erasmus: a scholar who asks how Kepler uses *lex* is asking for the LWgrammar entry for *lex* in the scope `kepler-prim`.

6.5 Entity registry and partitur register

The entity registry (`entity_registry.duckdb`) maintains canonical person and place forms with their collection-specific spelling variants and aliases. It is shared across collections and serves the `entity-fast-filter` and `correspondence-roster` specialists.

The Partitur register (`partitur_register.duckdb`, shared between the ai-query server and the audit CLI) stores goal contracts, plan paths, stage definitions, and result specifications for every registered Partitur. It is opened with short-lived, per-operation connections — a permanent write connection causes silent lock starvation that manifests as annotation and learning-loop failures.

7 Case Studies at Scale

7.1 KGW 15: Certified ephemerides

Kepler's *Gesammelte Werke* volume 15 contains 567 pages of correspondence and astronomical tables. The volume includes 89 figures and a dense population of astronomical ephemerides: trigonometric tables listing *sinus versus* and *sinus rectus* at 1° increments across 12 columns.

After full batch OCR and the `certify-and-learn` stage:

- Structural blockers: 0
- Visual blockers: 0
- Well-formed pages: 567/567 (100%)
- Figure completion: 89/89 at `complete_full` or `complete_desc`
- Fault cases registered: 25 (all Layer 1 prompting improvements for table cue detection; none blocking certification)
- Tag hierarchy protocol: 567 fingerprints; 3 pages showed structural deltas after the overlay improvement, all confirmed correct by facsimile comparison

The 25 fault cases were posted to the ai-query learning loop. The first iteration proposed an overlay rule specifying the column-count threshold for mandatory `<table>` generation. The golden re-run on the 3 regression anchor pages showed 0 blockers after the fix, and the overlay version was promoted.

■ 7.2 Pirckheimer (354 letters): correspondence queries at 200 ms

The Pirckheimer correspondence collection holds 354 letters and 78 persons in `pirckheimer.duckdb` (`letters_resolved` + `ent_canonical` tables). After certification and corpus completion, the query pipeline answers:

Who were Pirckheimer's most active correspondents between 1520 and 1530? → 75 unique correspondents; response time 16.8 s (LLM synthesis over filtered letter set). Golden score: 1.000.

What are the main subjects of Schöner's letters to Pirckheimer between 1520 and 1527? → 11 letters; response time 8.2 s. Golden score: 0.950.

In how many letters is Georg Hartmann mentioned? → 15 letters, 4 correspondents; response time 4.1 s. Golden score: 1.000.

These goldens verify three things: the entity registry correctly maps correspondent name variants; the `letters_resolved` table covers the full date range; and the `correspondence-roster` specialist correctly distinguishes senders/receivers from body mentions.

■ 7.3 Astronomical Diaries (ADRT 1–7): transliteration at scale

The Diaries collection spans 7 volumes and approximately 6,500 pages. It is the largest homogeneous collection in the PDAI workspace. Production OCR runs under `qwen3.8:27b-mlx` with the `diaries-translit-ocr.j2` template.

The unattended volume chain (`chain_diaries_all.sh`) processes ADRT 1 through 7 in succession, each volume executing its pilot gate independently. The live monitor at `http://localhost:5112/ocr-monitor?volume=adrt<N>` reports worker-aware throughput metrics from the runner's InflightTracker.

A critical invariant: when page indices are shifted (e.g., blank flyleaves are removed from ADRT 1), every downstream reference — page folders, `pidx` values in XML, `spread_map.json`, DuckDB tables, and golden queries — must be updated. Shift arithmetic alone is insufficient; an independent correlation control (`verify_pidx_correlation.py`) confirms that claimed OCR content matches expected text at the new `pidx` before the migration is accepted.

■ 8 The Edition Registry and the Viewer Contract

■ 8.1 The delivery contract

A volume is delivered to the ai-viewer endpoint when three conditions are simultaneously satisfied:

1. `pdaï-audit` certification: `blockers == 0 AND vis_blockers == 0`
2. `pdaï-complete` readiness: all corpus stages complete
3. Edition registry entry: volume appears in the collection's `editions.yaml`

These conditions are checked by the `editions_supervisor.py` (launched job `com.ai-platform.editions`), which manages 8 viewer instances under KeepAlive. Killing the supervisor cleanly restarts all children.

■ 8.2 The edition viewer

The `ai-viewer` instance for a collection is a FastAPI server that serves:

- `GET /doc/<alias>/viewer`: the split-view page with facsimile and transcription
- `GET /api/collection/<collection>/page.css`: the collection's overlay stylesheet
- `GET /api/segments`: the manifest-driven segment navigation
- `GET /audit`: the integrated quality-scoring panel for the collection's golden set

The DOM-based renderer (`repro-xml.js`) renders every page. It produces a tree of elements with `data-el="tagname"` and `data-attribute="value"` that CSS can fully target; a marginal is an inline span that the shared sheet moves into the margin while it keeps the height of its line. The `@style` collision that plagued the legacy string transform is absent: every XML attribute is a `data-` attribute on the DOM node.

The `<ai-query-chat>` web component runs in Shadow DOM with all CSS bundled inline (`?inline import` into the shadow root `<style>` tag). This makes the query interface self-contained across all viewer instances — no shared stylesheet required.

8.3 Port registry

Each edition viewer runs at a registered port, mapped by the platform registry:

Collection	Viewer port
Enestroem	5100
Euler OO	5103
Kepler	5104
Copernicus	5105
Waeßler	5106
Euler OO IVA	5107
Pirckheimer	5108
Kirch	5109
Editions landing	5111
Diaries / Medieval-Sec	5112
Kármán archive	5725

The ai-query lab (port 5113) is the query-resolution dev server, not a viewer instance.

9 Implications for Computational Epistemology

9.1 The page as the unit of evidence

The unit of evidence in the PDAI system is the page: one raster image plus its structured transcription, sharing one address (`pid`). Every query resolves to pages; every corpus operation — lemma lookup, entity filter, dictionary construction — takes place over page sentences.

This is an architectural commitment, not a convenience. A claim about what Kepler wrote can be verified in one click: the page address opens the facsimile alongside the transcription, providing the same ground a printed apparatus would offer, reached in milliseconds. A system whose answers ended in paraphrase — summarising what Kepler wrote without a path back to the page — would leave scholarly dispute nowhere to stand.

The audit system enforces this commitment. A page that fails certification is not a page; it is an unresolved production artefact. The `blockers == 0` criterion is not a quality aspiration but a binary gate: either the page's structural transcription is trustworthy, or it is not in the corpus.

9.2 Grounding Evidence vs Sekundärevidenz

The PDAI system distinguishes two kinds of scholarly evidence. *Grounding Evidence* is the direct textual or observational basis of a historical claim: a letter page, an observation record, a theorem statement. *Sekundärevidenz* is the interpretive layer: a monograph's account of what the letter says, a commentary's gloss of the theorem.

PDAI pages are Grounding Evidence by design: they are transcriptions of the original sources, validated against the original typography, with provenance recording the model and instruction that produced them. They do not interpret; they transcribe. The query layer reads Grounding Evidence to answer scholarly questions; it does not substitute interpretation for the source.

This distinction matters for computational epistemology because the two kinds of evidence have different epistemic status. A claim grounded in a letter page can be reproduced and disputed at the level of the facsimile; a claim grounded in a secondary account inherits all the interpretive decisions of that account. The PDAI infrastructure makes Grounding Evidence computationally accessible without collapsing it into secondary interpretation.

■ 9.3 Reproducibility and provenance

Every PDAI page records which model, instruction template, and date produced its transcription. This makes PDAI corpora reproducible in a strong sense: not merely that the data is available, but that the production process can be re-run with identical configuration and verified to produce the same structural output.

The instruction versioning system in the learning loop ensures that a change to an instruction template is recorded with a version number, a change rationale, and the results of the A/B test that promoted it. A scholar who cites results computed over a PDAI corpus can cite the instruction version and the model alongside the document set; a future researcher can verify whether a reprocessed corpus with an updated instruction produces the same structural fingerprints on the regression anchor pages.

This provenance chain is what distinguishes PDAI-certified pages from uncertified OCR output. Uncertified output may be accurate; it does not carry the structural audit, the fault dossier, or the regression guarantee that make it epistemically reliable as Grounding Evidence.

■ 9.4 Computational engagement with primary sources

Traditional scholarship involves locating documents, transcribing relevant passages, translating foreign-language content, manually extracting structured data, and building indices and concordances by hand. With a PDAI-certified corpus, these steps become computational: the corpus is queried by lemma, entity, date, or document type; translation layers accumulate in the `ai/` directory without overwriting transcriptions; entity extraction is a stanza pipeline stage, not a manual operation; indices are built by `bib_build` and `lwgrammar_build`, not by hand.

This does not replace close reading. It augments it. The scholar can still read Kepler’s Latin — and the viewer shows the facsimile alongside the transcription — but they can also query 1,200 letters by correspondent and topic, compare the use of *lex* across the Kepler correspondence and the Euler treatises, and trace the concept of *harmonia* across 26 volumes of *Gesammelte Werke*.

The infrastructure handles the mechanical labour — transcription, validation, indexing, registry maintenance — so that scholarly judgment can be spent where only judgment serves.

■ 10 Conclusion

PDAI is an infrastructure for treating historical scholarly documents as Grounding Evidence in computational inquiry. Its format encodes the nine structural categories of historical scholarly typography in a grammar that is validated per page. Its production pipeline governs the path from PDF to certified corpus through a Partitur that includes a mandatory audit and learning stage. Its contextual data stores — DuckDB corpus, lemma postings, gazetteer, LWgrammar lexicon, entity registry — make certified pages answerable to structured scholarly queries at millisecond latency.

The certify-and-learn system closes the production loop: every defect is attributed to a root-cause layer, registered as a versioned learning case, and resolved through an A/B-gated instruction improvement. The tag hierarchy protocol provides the regression guarantee. The fault dossier provides the precise descriptions that make improvement possible without inadvertent structural degradation.

At present, the PDAI workspace covers approximately 37 collections, 155,000 page XML files, and over 14,000 pages of Kepler alone. Five collections are fully certified and queryable; the remainder are at various stages of production. The Astronomical Diaries collection – cuneiform transliteration at a scale of 6,500 pages – is the current production frontier, testing the pipeline’s handling of non-Western scripts and specialized layout conventions.

The infrastructure is open and documented. The format specification (`pdai/docs/`) and the production pipeline skill (`pdai/skills/pipeline/`) are the primary references. The Quarto documentation book at `pdai/docs/_site/` is generated from the workspace and serves as the canonical reference for anyone working with the PDAI system.

11 Acknowledgments

I thank the libraries and archives whose digitisation programs made the source materials available. The Quarto project enabled the documentation system; DuckDB provided the query foundation; the Ollama project made local inference practical. This work was supported by the Humboldt-Universität zu Berlin (Philosophisches Institut) and the Forschungsinstitut des Deutschen Museums, München.

Bibliography

- [1] G. Graßhoff, “The Partitur: A Conductor Architecture for Multi-Stage Scholarly Query Resolution,” 2026.
- [2] G. Graßhoff, “Querying with Linguistic Trees: Stanza-Backed Lemma Normalisation, Dependency Parses, and DuckDB,” 2026.
- [3] G. Graßhoff, “AI Erasmus: A Platform for a New Era in Digital Humanities,” 2026.
- [4] ISO, “ISO 32000-2:2020 Document Management — Portable Document Format — Part 2: PDF 2.0,” International Standard, 2020.
- [5] TEI Consortium, “TEI P5: Guidelines for Electronic Text Encoding and Interchange.” 2023.
- [6] L. Euler, *Inquisitio Physica in Causam Fluxus ac Refluxus Maris*. Petropoli: Academiae Scientiarum Imperialis, 1740.
- [7] J. Kepler, *Gesammelte Werke, Band XIII: Briefe 1590–1599*. in Johannes Kepler *Gesammelte Werke*. München: C. H. Beck, 1945.
- [8] O. Neugebauer, *A History of Ancient Mathematical Astronomy*, vol. 1. in *Studies in the History of Mathematics and Physical Sciences*, vol. 1. Berlin: Springer, 1975.
- [9] L. Euler and C. Goldbach, *Correspondance mathématique et physique*. Saint-Pétersbourg: L'Académie Impériale des Sciences, 1843.