

# Golden Queries in AI Erasmus

*A testbed for scholarly inquiry: query forms, grounding, and the monitor log*

Gerd Graßhoff 

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

## ABSTRACT

A research platform is only as trustworthy as the questions it can be held to. This paper presents the *golden queries* of AI Erasmus — a canonical set of thirteen scholarly questions, each with a declared form of inquiry, a grounding document, and an oracle — and the *monitor log* that makes every answer inspectable. A golden query is a research question whose expected answer is declared in advance, so the pipeline is audited against a reference rather than against a model’s fluency. Two kinds of oracle are in force: a *count oracle*, a verified number the pipeline must match exactly, and a *floor oracle*, a register-grounded minimum a healthy run must clear. We walk the thirteen canonical goldens across four corpora — Euler, Kepler, Pirckheimer, and von Kármán — showing seven forms of inquiry: lemma co-occurrence, attestation, person mention, correspondence roster, letter subject, place-and-person lookup, and concept statements. For each we show how the question is grounded in a document layer, and we reproduce the monitor log — segmented section by section, each paragraph explaining the goal solution with its strengths and limits. We close with the reliability metrics the golden set makes possible: recall of the general filter wherever a sentence-level must-include set exists, the independent DuckDB audit of the ground tier, and the per-stage timing rollup of the 13-golden subset of the 12 August 2026 orchestrator audit. We then show the *learning loop* that closes the circle from a monitor-log deficit to a versioned, A/B-gated patch — worked through a real case in which a misresolved person name was fixed, gated against the full suite, and promoted. The set is designed to grow: new queries are introduced, tested, and promoted as the platform’s scope widens.

**KEYWORDS** digital humanities · golden queries · query grounding · monitor log · PDAI · lemma search · correspondence networks · reliability metrics

## Table of contents

|                                                              |           |
|--------------------------------------------------------------|-----------|
| <b>1 Introduction: The Question of Trust</b>                 | <b>2</b>  |
| <b>2 The Basic Approach: Golden Queries as the Objective</b> | <b>3</b>  |
| 2.1 What a golden query is                                   | 3         |
| 2.2 Why the golden set is the objective                      | 4         |
| 2.3 The pipeline the goldens exercise                        | 4         |
| <b>3 The Query Families and Their Forms of Inquiry</b>       | <b>5</b>  |
| <b>4 The Golden Set: Thirteen Queries, Four Corpora</b>      | <b>5</b>  |
| 4.1 Lemma co-occurrence: R1, R2, K5                          | 5         |
| 4.2 Attestation: R4                                          | 6         |
| 4.3 Person mention: R5, K3                                   | 6         |
| 4.4 Correspondence roster: K1, ground-pirckheimer-1          | 7         |
| 4.5 German letter-count: K2                                  | 8         |
| 4.6 Letter subject: ground-pirckheimer-2                     | 8         |
| 4.7 Place-and-person lookup: K4                              | 8         |
| 4.8 Concept statements: T4-kepler-motrix, T4-euler-aether    | 9         |
| <b>5 Grounding: Documents and the Grounding Locus</b>        | <b>9</b>  |
| 5.1 The grounding layers                                     | 10        |
| 5.2 The independent audit                                    | 10        |
| <b>6 The Monitor Log and Its Mapping Function</b>            | <b>10</b> |
| 6.1 The monitor log, section by section                      | 11        |

|                                                                  |           |
|------------------------------------------------------------------|-----------|
| 6.1.1 §1 — UNDERSTAND: the Phase-0 Partitur, played .....        | 11        |
| 6.1.2 §2 — PARTITUR SELECTION .....                              | 11        |
| 6.1.3 §3 — PARTITUR TEXT .....                                   | 12        |
| 6.1.4 §4 — SIGNALS .....                                         | 12        |
| 6.1.5 §5 — EXECUTION STREAM .....                                | 12        |
| 6.1.6 §6 — ANNOTATED GOAL TREE .....                             | 13        |
| 6.1.7 §7 — UNBOUND EXECUTION .....                               | 13        |
| 6.1.8 §8 — LLM CALLS .....                                       | 14        |
| 6.1.9 §9 — ANSWER / GOAL CHECK .....                             | 14        |
| 6.1.10 §10 — VERDICT .....                                       | 14        |
| 6.2 What the monitor does well, and where it falls short .....   | 15        |
| <b>7 Reliability and Metrics .....</b>                           | <b>15</b> |
| 7.1 Recall of the general filter .....                           | 15        |
| 7.2 The independent DuckDB audit .....                           | 16        |
| 7.3 The per-stage timing rollup .....                            | 16        |
| 7.4 The suite rollup .....                                       | 17        |
| <b>8 The Learning Loop: From Monitor to Promoted Patch .....</b> | <b>18</b> |
| 8.1 Two entry points, one gate .....                             | 18        |
| 8.2 Worked example: ground-pirckheimer-3, Georg Hartmann .....   | 18        |
| 8.3 Discussion: what the loop buys, and what it risks .....      | 19        |
| <b>9 Extending the Set .....</b>                                 | <b>19</b> |
| 9.1 Candidates still under test: terminology-find .....          | 20        |
| <b>10 Conclusion .....</b>                                       | <b>20</b> |
| <b>11 Acknowledgements .....</b>                                 | <b>21</b> |
| <b>Bibliography .....</b>                                        | <b>21</b> |

## 1 Introduction: The Question of Trust

A scholar’s instrument earns trust the way a workshop does: by being held to account. In Erasmus’s printing house the amanuenses copied, collated, fetched, and counted, and the judgement about what a variant meant never left the desk. The modern instrument that AI Erasmus builds [1], [2] staffs those repeatable moves with specialists — but a specialist that answers fluently is not the same as a specialist that answers *right* [3]. The difference is the subject of this paper. Meaning, in the lemma queries that follow, is use under a citable rule [4], [5]: a co-occurrence of *lex* and *natura* is a fact about pages, not a claim that Kepler wrote of a *lex naturae*.

The problem is the asymmetry of failure. A lemma search that returns nothing is honest: “no letters from this sender in this range” is scholarship’s ordinary weather. A generated answer is never empty; pressed for something it does not know, a model produces fluent material with no page behind it, and the failure is invisible precisely where it is worst. The platform’s answer to this asymmetry is *provenance* — every result carries the page addresses it rests on, the method that produced it, and the scope it was computed over. But provenance alone is not enough. A method trace tells you *how* an answer was produced; it does not tell you whether the answer was *right*.

### CONCEPT The golden-query stance

A golden query is a research question together with a form of inquiry, a grounding document, and an oracle. An oracle is the declared expected answer against which a run is checked — not the human label later assigned to the run as a whole. Two kinds are in force: a count oracle (a verified number) and a floor oracle (a register-grounded minimum). The golden set is the platform’s objective: the learning loop maximises the fraction of golden queries whose oracle is satisfied. Without goldens, an answer is judged by fluency; with them, it is judged by accountability.

This paper is the full statement of that testbed. It makes five contributions:

1. **The basic approach.** What a golden query is, how its two kinds of oracle are declared, and why the golden set is the objective of the learning loop (Chapter 2).
2. **The query forms.** The seven forms of inquiry the canonical set exercises — lemma co-occurrence, attestation, person mention, correspondence roster, letter subject, place-and-person lookup, and concept statements — each with its evidence path (Chapters 3–4).
3. **Grounding.** How each question is grounded in a document layer, and what the grounding documents are (Chapter 5).
4. **The monitor log.** The mapping function that turns a run into a readable, sectioned protocol, reproduced and explained paragraph by paragraph, with the strengths and limits of each section (Chapter 6), and the reliability metrics the set makes possible (Chapter 7).
5. **The learning loop.** How an annotated deficit in the monitor log becomes a versioned, A/B-gated patch to the pipeline — the mechanism that actually maximises the fraction of goldens whose oracle is satisfied (Chapter 8). The protocol for extending the set is Chapter 9.

The set is designed to grow. New queries are introduced, tested against their oracles, and promoted into the canonical set as the platform’s scope widens. The paper closes on what the golden set changes in the craft of building a scholarly instrument.

## ■ 2 The Basic Approach: Golden Queries as the Objective

### ■ 2.1 What a golden query is

A golden query is a research question together with three things that distinguish it from an ordinary request to the server:

- a *form of inquiry* — the kind of scholarly act the question asks for (attesting a lemma, counting correspondents, extracting statements about a concept);
- a *grounding document* — the corpus layer that can answer it (lemma index, letter register, gazetteer, sentence index);
- an *oracle* — the declared expected answer against which a run is checked.

A *query family* is the grouping of goldens that share one form of inquiry and one evidence path. The form is the act; the family is the set of goldens that exercise it. The oracle is the load-bearing part: it is what turns a query run from a demonstration into a test.

Two kinds of oracle must not be collapsed. A **count oracle** is a verified number, derived by hand from the register, that the pipeline must match exactly — `n_unique_correspondents: 75` for `ground-pirckheimer-1`. A **floor oracle** is a register-grounded minimum, recorded in the golden’s `truth` note, that a healthy run must clear and a broken route must fail — `n_sentences.min: 30` for `R1`, whose register union on 12 August 2026 is 86 sentences. A third thing is not an oracle at all: the **run verdict** (`correct`, `partial`, `incorrect`) that a human labeller assigns to the run as a whole after reading a sample of top sentences. The learning loop maximises the fraction of runs whose oracle is satisfied; the labeller’s verdict is the human check on whether that satisfaction was the right scholarly answer.

The canonical set of thirteen goldens is indexed in a manifest [6]. Each entry names its tier, its corpus, its query, and its role:

**PARTITUR** `goldens/manifest.yaml` / canonical set (13)

```
version: 2
name: euler-kepler-pirckheimer-karman 1
corpora: [enestroem, euler00, kepler-prim, kepler-sec,
           pirckheimer, karman-nachlass]
```

```

goldens:
- id: R1
  tier: catalog
  corpus: [enestroem, euler00] 2
  query: "Sentences in Euler Enestroem with lemma lex natura"
  role: lemma cooccurrence (lex + natura)
- id: ground-pirckheimer-1
  tier: ground
  corpus: [pirckheimer]
  query: "How many correspondents did Pirckheimer have between 1520 and 1530?" 3
  oracle: {n_unique_correspondents: 75}
  role: verified correspondence-count ground

```

- 1 Six corpora, four centuries, three languages — one set of operations.
- 2 The catalog tier: lemma co-occurrence over Euler’s Eneström corpus [[7]].
- 3 The ground tier: a verified count — the pipeline must return 75.

PLAYED BY **ai-query golden audit**

The tiers matter, and they must not be read as “shape versus number”. As of August 2026 every canonical golden carries an oracle of one of the two kinds above. The **catalog** tier (R1, R2, R4, R5) holds lemma and person queries whose oracles are floors on sentence counts, grounded in a direct probe of the lemma index. The **ground** tier (ground-pirckheimer-1, -2) holds queries whose oracles are verified facts, derived by hand from the Pirckheimer letter register [8]. The **karman** tier (K1–K5) holds queries over the von Kármán archive [9], with floor oracles on correspondent or letter counts. The **concept** tier (T4-kepler-motrix, T4-euler-aether) holds statement-extraction queries whose oracles are floors on the number of grounded statements. A catalog golden tests whether the pipeline understands the question *and* clears a register floor; a ground golden tests whether it matches a hand-derived number.

## 2.2 Why the golden set is the objective

The learning loop of AI Erasmus maximises the aggregate golden score — the fraction of golden queries answered correctly or partially. This is a deliberate choice of objective. It means the platform is not optimised to sound scholarly; it is optimised to answer the questions a scholar actually asks, as judged against a reference.

The choice has a consequence that is worth stating plainly. A golden query that the pipeline cannot answer is not a failure to be hidden; it is a finding. When the monitor shows a golden returning the wrong subtask, or a specialist returning an empty set where the oracle expects a count, that is the learning loop’s raw material. The golden set turns the pipeline’s mistakes into a ledger that can be read, attributed to a subagent, and patched.

### METHOD The verdict is assigned to the run, not the item

A golden item is an assessment **query run** that a human has evaluated and labelled. The run verdict — correct, partial, or incorrect — is always assigned to the run as a whole, never to individual sentences or documents in the result set, and it is not the oracle. A query returning 500 sentences is still a single run with a single verdict. The labeller reads 5–15 top sentences to form a judgement of the run, not all 500.

## 2.3 The pipeline the goldens exercise

Every golden is run through the same path an ordinary request to the server takes: `orchestrate()` plans the work as an ordered set of smaller steps and solves them one by one. The first step is the Partitur, a fast rule-based analysis that classifies the question, resolves persons, places, and dates, and selects the collection

scope. Later steps depend on what the Partitur and the analyser decided: a correspondence question routes to a register specialist, a lemma question to the index, a person question to the gazetteer.

The golden set is deliberately spread across these paths. Some goldens exercise the fast lemma trail; others exercise the register specialists; still others exercise the entity gazetteer. The spread is the point: a golden set that only tested one path would certify only one path.

3 The Query Families and Their Forms of Inquiry

A scholarly question, phrased in ordinary language, is routed by *decomposition* — the deterministic classification of the question into a family with a declared evidence path. The golden set exercises the families that matter most for historical and conceptual research.

**Table 1** The query families exercised by the golden set: each family is the set of goldens that share one form of inquiry and one evidence path.

| Family                  | Form of inquiry                                         | Example golden                    | Evidence path                                  |
|-------------------------|---------------------------------------------------------|-----------------------------------|------------------------------------------------|
| Lemma co-occurrence     | Sentences where two lemmata appear together             | R1, R2, K5                        | Lemma index (boolean AND)                      |
| Attestation             | Whether a single lemma appears at all                   | R4                                | Lemma index (single lemma)                     |
| Person mention          | Passages where a named person is typed as a person      | R5, K3                            | Person gazetteer + letter bodies               |
| Correspondence roster   | Who wrote to whom, or how many letters, in a year range | K1, K2, ground-pirckheimer-1      | Resolved letter register                       |
| Letter subject          | What a set of letters is about                          | ground-pirckheimer-2              | Letter register + body text                    |
| Place-and-person lookup | Letters bound to a place and a person together          | K4                                | Place gazetteer + letter register              |
| Concept statements      | Grounded statements about a concept, with implications  | T4-kepler-motrix, T4-euler-aether | Sentence index + concept-statements specialist |

Table 1 is a working map, not a closed ontology. Its point is the third column: every family names the layer it reads, so every answer arrives knowing where it came from. The golden set is a sample of these families — a sample chosen to cover the paths the platform must not get wrong.

4 The Golden Set: Thirteen Queries, Four Corpora

This chapter walks the thirteen canonical goldens. For each we state the query, its form of inquiry, its grounding, and its oracle — a count where the oracle is a verified fact, a floor where it is a register-grounded minimum. The monitor log of one of them is reproduced in full in Chapter 6.

4.1 Lemma co-occurrence: R1, R2, K5

The lemma co-occurrence query asks for sentences where two lemmata appear together. It is the workhorse of conceptual research: “where do *lex* and *natura* co-occur?” is the question behind a claim about how a concept is used.

OPERATION

Lemma co-occurrence

189 ms (partitur 88.8 ms)

Q

Sentences in Euler Enestroem with lemma lex natura

|   |                          |                                                                                                               |
|---|--------------------------|---------------------------------------------------------------------------------------------------------------|
| 1 | <b>fasttext-classify</b> | classifies the question as a co-occurrence query — intent=cooccurrence, scope=sentence                        |
| 2 | <b>collection-select</b> | fixes the scope to the Eneström corpus — the score's iron rule keeps scope terms out of any content filter    |
| 3 | <b>stanza-filter</b>     | attests the lemmata lex and natura in the lemma index and retrieves every co-occurring sentence with its page |
| 4 | <b>assemble</b>          | groups the evidence into an answerable shape, every item with a locus                                         |

→ a set of sentences where lex and natura co-occur, each with its page address — the raw material of a claim about usage

The grounding is the lemma index: `lemma_index.parquet`, built by Stanza under Universal Dependencies [10], [11] and stored in DuckDB for interactive query [12]. The oracle for R1 is a floor, not a single exact count: the pipeline must produce `lemma_ncf = [[lex],[natura]]` with the right collections and at least 30 co-occurring sentences. The register truth of 12 August 2026 is 86 sentences (enestroem 70 in 34 documents, eulerOO 16 in 7). R2 is the same concept scoped to Kepler [13], testing fan-out across `kepler-prim` and `kepler-sec`, with a floor of 15 against a register truth of 45 in `kepler-prim` and 0 in `kepler-sec`. K5 is the same shape over the von Kármán archive, exercising the general filter on a corpus without a `letter_pages` table, with a floor of 100 against a register truth of 1,426 sentences.

The form's specific difficulty is the *iron rule*: a collection name must never enter the lemma filter. “Lex natura in Kepler's writing” must produce `lemma_ncf = [[lex],[natura]]` with `collections=[kepler-prim]`, never `lemma_ncf = [[lex],[natura],[kepler]]`. The golden R2b is the regression test for exactly this disambiguation — the “Lex natura in Kepler's writing” bug that once treated the corpus scope as a co-occurring lemma.

## 4.2 Attestation: R4

The attestation query asks whether a term appears at all. It is the negative control of scholarship: “does vacuum appear in Euler's writings?” is a question whose honest answer may be “no”.

| OPERATION Attestation                     |                          | 16.9 s (full harness)                                                                |
|-------------------------------------------|--------------------------|--------------------------------------------------------------------------------------|
| Q Does vacuum appear in Euler's writings? |                          |                                                                                      |
| 1                                         | <b>fasttext-classify</b> | classifies the question as an attestation query — intent=attestation, scope=sentence |
| 2                                         | <b>stanza-filter</b>     | probes the lemma vacuum against the lemma index of the Eneström corpus               |
| 3                                         | <b>assemble</b>          | returns the verdict — found or absent — with the evidence                            |

→ a verdict on whether vacuum is attested, with the sentences that attest it — or an honest empty answer

The form's specific virtue is that it tests the pipeline's honesty. An attestation query whose answer is “absent” must return an empty result with its trace, not a plausible guess. The golden R4 allows `min_rows: 0` — the pipeline is *expected* to be able to answer “no”. This is the amanuensis stance applied to the negative case: the empty answer is reported, not papered over.

## 4.3 Person mention: R5, K3

The person-mention query asks where a person is named in a text. It is the first step of prosopography: “where is Tycho mentioned as a person in Kepler?” is the question behind a reading circle or a reception study.

| OPERATION Person mention | 43.9 s (full harness) |
|--------------------------|-----------------------|
|--------------------------|-----------------------|

**Q** *Where is Tycho mentioned as a person in Kepler?*

- 1 **fasttext-classify** classifies the question as an entity query — intent=entity, scope=sentence
- 2 **person-classify** resolves the name Tycho against the person gazetteer and types it as a person
- 3 **gazetteer-entity** maps the surface form to the registry entity and probes the corpus
- 4 **lemma-execute** retrieves the sentences where the person is named, with their pages

→ the passages where Tycho is named as a person, each with its page — the raw material of a prosopographical claim

The form’s specific difficulty is *typing*. “Tycho” must be resolved as a person, not treated as a bare lemma. The golden R5 tests that the gazetteer-entity subagent appears in the trail and that the entity is typed correctly. K3 is the same shape over the von Kármán archive: “who mentions Einstein in his letters to Karman?” routes to the `mention-in-letters` specialist, which reads the letter register and the letter bodies.

#### 4.4 Correspondence roster: K1, ground-pirckheimer-1

The correspondence-roster query asks who wrote to whom, within a range of years. It is the heart of network research: “who corresponded with Karman between 1930 and 1940?” is the question behind a correspondence network.

**OPERATION** Correspondence roster

370 ms (full harness)

**Q** *How many correspondents did Pirckheimer have between 1520 and 1530?*

- 1 **fasttext-classify** classifies the question as a correspondence-count query — intent=roster
- 2 **person-classify** resolves Pirckheimer against the person gazetteer and sets `wanted_role=correspondent`
- 3 **date-classify** extracts the year range 1520–1530
- 4 **correspondence-roster** reads the resolved letter register, filters by year, and counts distinct correspondents

→ 75 distinct correspondents between 1520 and 1530 — the oracle the pipeline must match

The form’s specific virtue is that it has a *count oracle*. The ground golden ground-pirckheimer-1 pins the answer at 75 distinct correspondents, derived by hand from the register. An independent DuckDB query of `letters_resolved` — `COUNT(DISTINCT other_id)`, excluding the main person and unknown senders, years 1520–1530 — returns the same 75. That query is independent of the orchestrator and of any model call; it is not independent of the register schema, and the audit notes record that it matches the specialist’s year-window predicate. What it certifies is that the number 75 is a fact about the table, not an artefact of the pipeline’s routing.

K1 is the same form of inquiry over a different corpus and a floor oracle. The query is “*Who corresponded with Karman between 1930 and 1940?*” The register truth of 12 August 2026 is 494 raw names / 493 canonical ids in that window; the floor is 200, set so that canonicalization merges never fail a healthy run and a broken route cannot clear it. The 12 August 2026 orchestrator audit returned 491 distinct correspondents and an oracle verdict PASS.

4.5 German letter-count: K2

K2 is the correspondence-roster form asked in German, as a letter count rather than a who-list: “*Wie viele Briefe schrieb Millikan an Karman?*” It is the golden the previous walkthrough omitted, and it is in the canonical thirteen.

OPERATIONGerman letter-count

1,678 ms (12 August 2026 audit)

QWie viele Briefe schrieb Millikan an Karman?

1fasttext-classify

classifies the German question as a correspondence-count query — intent=roster

2person-classify

resolves Millikan and Kármán against the person gazetteer

3correspondence-roster

reads the resolved letter register and counts letters from Millikan to Kármán

→ n\_letters in the band 3–20 — the floor-and-ceiling oracle that rejects both a zero-hit failure and a corpus-wide inflation

The register truth of 12 August 2026 is 4 letters under a strict sender-Millikan / recipient-Kármán filter, and 16 Millikan-sent letters overall, many with an empty recipient. The oracle band 3–20 accepts either strict or recipient-lenient counting. The audit run returned `n_letters = 10` and an oracle verdict PASS. The form’s specific difficulty is the German cue path together with a directed count: the pipeline must not treat the question as an English who-list and must not inflate the count to the whole archive.

4.6 Letter subject: ground-pirckheimer-2

The letter-subject query asks what a set of letters is about. It is the synthesis question: “what is the subject of the letters from Schöner between 1520 and 1527?” is the question behind a summary of a correspondence.

OPERATIONLetter subject

10.1 s (full harness)

QWhat is the subject of the letters from Schöner between 1520 and 1527?

1fasttext-classify

classifies the question as a letter-subject query — intent=roster

2person-classify

resolves Schöner and sets wanted\_role=correspondent

3date-classify

extracts the year range 1520–1527

4letter-subject

reads the register, retrieves the letters, and summarises their subject

→ a summary of the letters’ subject — the oracle requires the topic saphea (the astrolabe) to appear

The form’s specific difficulty is that the oracle is a *loose substring* check, not an exact number. The expected answer requires `n_letters: 11` and a `subject_summary` containing “saphea” — the astrolabe topic that dominates the Schöner correspondence. The loose check is deliberate: prose synthesis is not expected to be byte-identical between runs, but the load-bearing fact (the topic) must appear.

4.7 Place-and-person lookup: K4

The place-and-person query combines a place and a person: “letters sent from Pasadena by Millikan” is the question behind a study of where a correspondent wrote from.

OPERATIONPlace-and-person lookup

21.3 s (full harness)

QLetters sent from Pasadena by Millikan

|                                                                  |                              |                                                                          |
|------------------------------------------------------------------|------------------------------|--------------------------------------------------------------------------|
| 1                                                                | <b>fasttext-classify</b>     | classifies the question as a correspondence query                        |
| 2                                                                | <b>person-classify</b>       | resolves Millikan and sets wanted_role=correspondent                     |
| 3                                                                | <b>place-classify</b>        | resolves Pasadena against the place gazetteer                            |
| 4                                                                | <b>correspondence-roster</b> | reads the register, filters by place and person, and returns the letters |
| → the letters sent from Pasadena by Millikan, each with its page |                              |                                                                          |

The form’s specific difficulty is the *joint resolution* of place and person. The golden K4 tests that Pasadena resolves as a place and Millikan as a person, and that the two bind together in the register filter. The partitur benchmark pins the expected binding: `place_binding: pasadena` and `letter_ent_name: millikan`. The register truth of 12 August 2026 is 13 letters under a sender-Millikan / place-Pasadena filter, 19 if Millikan is allowed in either role; the oracle band is 5–25. The known limitation, recorded in the manifest, is that “sent BY Millikan” direction is not enforced.

#### 4.8 Concept statements: T4-kepler-motrix, T4-euler-aether

The concept-statements query asks what an author *says* about a concept, and what those statements imply. It is a different form of inquiry from lemma co-occurrence: R1 returns sentences where *lex* and *natura* co-occur; T4-kepler-motrix asks for grounded statements about the *vis motrix* and their implications. The retrieval unit is the statement, not the sentence, and the specialist is `concept-statements`.

| OPERATION Concept statements                                                                                                                                               |                           | 3,342 ms (12 August 2026 audit)                                                                      |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------|------------------------------------------------------------------------------------------------------|
| Q What statements does Kepler make about the vis motrix, and what do they imply?                                                                                           |                           |                                                                                                      |
| 1                                                                                                                                                                          | <b>fasttext-classify</b>  | classifies the question as a concept-statements query                                                |
| 2                                                                                                                                                                          | <b>collection-select</b>  | fixes the scope to kepler-prim — Kepler is corpus scope, not a co-occurring lemma                    |
| 3                                                                                                                                                                          | <b>concept-statements</b> | segments grounded statements about vis motrix from the sentence index and records their implications |
| → at least five grounded statements from at least three documents — the floor oracle against a register truth of 145 motrix-anchored sentences in 15 kepler-prim documents |                           |                                                                                                      |

T4-euler-aether is the same form in German, over Euler: “*Welche Aussagen macht Euler über den Äther?*” It exercises the German cue path (“Aussagen über X”) and Latin-corpus seed resolution from a German concept name. The register truth of 12 August 2026 is 269 `aether%` sentences across 15 enestroem documents; the floor is five statements from three documents. The 12 August 2026 audit returned oracle PASS for both T4 goldens (T4-kepler-motrix: 7 statements, 26 documents; T4-euler-aether: 45 statements, 5 documents).

One limit belongs in the same paragraph as the result. The floor oracle checks counts, not the prose of the synthesis. In the same audit run the T4-kepler-motrix synthesis still spoke the roster register’s language (“No persons found in the kepler-prim collection”) even as the specialist returned seven statements. That mismatch is a finding for the learning loop, not a silent pass.

## 5 Grounding: Documents and the Grounding Locus

Every golden query is grounded in a document layer. The grounding document is the layer that answers the question — the lemma index for a co-occurrence query, the letter register for a roster query, the gazetteer for a person query. The grounding *locus* is the specific address — collection, doc\_id, page index — that a claim rests on.

## 5.1 The grounding layers

The platform's substrate is the PDAI page: one raster image and one structured transcription under a single address [14]. Above the pages, derived layers accumulate:

- **The lemma index** (`lemma_index.parquet`) — one row per token, built with Stanza, stored in DuckDB [15]. Grounds co-occurrence and attestation queries over the Eneström, Kepler, and Euler corpora [7], [13], [16]. LWgrammar writes OED-form usage entries from that index [4], [17].
- **The letter register** (`letters_resolved`) — the edition's metadata of sender, recipient, and date per letter. Grounds correspondence roster and letter-subject queries [8], [9].
- **The person and place gazetteers** — canonical entities with their variant forms. Ground person-mention and place-and-person queries.
- **The stanza index** (`stz_sentences`) — the sentence layer, with dependency parses [18]. Grounds the sentence-level evidence and the concept-statements family.

Each golden names its grounding layer in its definition. R1 grounds in the lemma index; ground-pirckheimer-1 grounds in the letter register; R5 grounds in the person gazetteer. The grounding is what makes the oracle checkable: the independent audit re-runs the query directly against the layer, bypassing the pipeline, and compares.

### METHOD The grounding contract

Every answer carries its grounding locus: the collection, the doc\_id, and the page index it rests on. The chain from answer back to page is never broken. This is what allows the golden set to print numbers and mean them — and what allows the independent audit to check them.

## 5.2 The independent audit

The ground-tier check the golden set offers is an *independent audit* of the register, not a second run of the pipeline. For ground-pirckheimer-1 the audit runs `COUNT(DISTINCT other_id)` over `letters_resolved` and confirms the count oracle of 75. For ground-pirckheimer-2 it confirms 11 letters and the saphea topic. The audit does not call the orchestrator and does not call a model. It does use the same table, and for the year window it uses the same predicate as the specialist — the audit notes say so. What it answers is whether the oracle is a fact about the source layer. What it does not answer is whether the specialist's code is free of that predicate.

ground-pirckheimer-3 is in that audit and not in the canonical thirteen. It is the candidate used as the worked example of the learning loop in Chapter 8: four correspondents mention Georg Hartmann in letters to Pirckheimer, against thirteen index-cited letters. The suite table in Chapter 7 therefore reports two canonical ground goldens; Table 3 reports three rows because the candidate was audited alongside them.

## 6 The Monitor Log and Its Mapping Function

The monitor log is the readable record of a golden run. It is produced by a single script, `query_monitor.py`, which plays the Phase-0 Partitur once, selects the answering Partitur from the register, hands the understanding to the orchestrator, and writes the shared trail as numbered sections §1–§10. The mapping function is the heart of it: it turns the raw event stream and the goal tree into a sectioned protocol that a reader can follow.

### METHOD The mapping function

The monitor is not a separate diagnostic system and not a second pipeline. It is the readable imprint of the one execution path. The runner plays Phase 0 once, selects the answering Partitur, and writes the shared trail as numbered sections. The iron rule: never invent or paraphrase a monitor log. The log is the deliverable.

The sections are the anatomy of one answer. §1 shows the Phase-0 Partitur being played — the classification, the person and place resolution, the collection selection. §2 shows the Partitur selection from the register. §3 prints the full goal text of the chosen Partitur. §4 shows the signals — persons, years, collections. §5 is the execution stream, the chronological protocol of every event. §6 is the annotated goal tree, joining the plan to the run. §7 lists unbound execution. §8 shows the LLM calls. §9 is the answer and goal check. §10 is the verdict.

## 6.1 The monitor log, section by section

We reproduce the monitor log of a golden run — the mention-in-letters query over Pirckheimer — and explain each section, with its pros and cons. The query is:

Which correspondent mentions Georg Hartmann in his letters to Pirckheimer?

### 6.1.1 §1 — UNDERSTAND: the Phase-0 Partitur, played

The first section is the richest and the longest, deliberately, because Phase 0 carries the entire routing decision. It shows the fixed Partitur `ai-query-understand` being played — not as a Python call sequence, but as a score execution. The section shows the setting (the context the query was asked in), the goal text verbatim from the register, the stages played with their guards and budgets, the full rendered instruction of the classifier, the attribute glossary, and the routing hints.

The stages played tell the story of the classification:

|                     |                            |          |      |         |
|---------------------|----------------------------|----------|------|---------|
| ✓ fasttext-classify | pdai:aiq_classify          | 1502.6ms | 2500 | ok      |
| ✓ person-classify   | pdai:aiq_person_classify   | 0.1ms    | 80   | ok      |
| ✓ place-classify    | pdai:aiq_place_classify    | 0.0ms    | 50   | ok      |
| ○ date-classify     | pdai:aiq_date_classify     | 0.0ms    | 200  | skipped |
| ✓ collection-select | pdai:aiq_collection_select | 4.9ms    | 40   | ok      |

The classifier returns `subtask_type=mention-in-letters` with confidence 0.95. The person classifier resolves nine persons. The date classifier is skipped — there is no temporal token in the query. The collection selector fixes the scope to `pirckheimer`.

**Pro.** The section makes the routing decision fully inspectable. A reader can see which classifier answered, with what confidence, and which stages were skipped and why. The SHA check confirms the disk YAML matches the register.

**Contra.** The section is long, and it repeats textbook material on every run — the specialist catalogue and the full instruction. For repeated runs this is redundant. The specialist definition is also slightly inaccurate for some subtasks: it says a specialist answers “from a structured register, not from lemma retrieval”, which is not true of `terminology-find`, which attests lemmata over the lemma index.

### 6.1.2 §2 — PARTITUR SELECTION

The second section shows the register choice. Candidates come only from `partitur_register.duckdb.partiturs`, filtered by goal-path prefix and term score. Skills are not searched.

- goal\_path prefix: query/historical/
- excluded kind: understanding
- selected ai-query-phase1 @ query/historical/stage-one (term\_score=1.767)

candidates:

- ai-query-phase1 score=1.767 query/historical/stage-one

**Pro.** The demarcation is stated every time: candidates only from the register, Phase-0 scores never answer. The candidate list shows the alternatives that were *not* chosen.

**Contra.** The section shows the score but not the reasoning. A reader sees that `ai-query-phase1` won, but not *why* the root Partitur beat a dedicated specialist Partitur. The decision should be commented, not just scored.

### 6.1.3 §3 – PARTITUR TEXT

The third section prints the full goal text of the chosen Partitur from the register – not truncated, not paraphrased.

```
Beantworte eine historische Fachfrage aus belegter Evidenz: Artikuliere
zunächst, was die Frage verlangt und welche Dokumente überhaupt als
Antwort infrage kommen (Grounding-Primärquellen vs. Sekundärrepositorien).
Filtere diese Evidenzmenge dann – immer billigst und selektivst zuerst –
auf die minimale Menge relevanter Passagen. Erst dieses kleine Resultat
wird in die volle LLM-Anfrage injiziert. Jede Behauptung im Ergebnis
trägt einen Grounding-Locus (collection, doc_id, pidx).
```

**Pro.** Completeness here is correct. Without §3 the selection is an ID; with it, the goal of the entire execution is readable: specify the evidence set, filter cheaply, inject only the remainder into the LLM, ground every claim.

**Contra.** None material. Full goal text is the right default for this section.

### 6.1.4 §4 – SIGNALS

The fourth section shows the condensed Phase-0 output that all later stages read as a common board – without re-parsing the query.

```
persons      = ['Pirckheimer, Willibald', 'Pirckheimer, Klara', ...,
                'Sauermann, Georg']
years        = {}
collections  = ['pirckheimer']
```

**Pro.** The three signals – persons, years, collections – are the shared board. The section is the right headline.

**Contra.** The section is too flat. The persons list shows only the canonical name strings, not the provenance – the surface term, the entity id, the role match. This is the most common review objection to §4. In this run, the token “Georg” collides with “Sauermann, Georg”, so “Georg Hartmann” is not resolved as a bigram – an entity-resolution gap that the flat list hides.

### 6.1.5 §5 – EXECUTION STREAM

The fifth section is the chronological protocol of every event in the AO\* run: `node_open`, `node_solved`, `subagent_start`, `subagent_end`, `decision`, `route`. It is the timeline.

```
[ 71] step= 7 info  node_open          route      10.6ms  open LEAF:execute mention-
in-letters
[ 72] step= 8 info  subagent_start      mention-in-letters  0.0ms  mention-in-letters
starting
[ 73] step= 8 info  decision            mention-in-letters 221.7ms mention-in-
letters returned n_out=14
[ 74] step= 8 info  subagent_end        mention-in-letters 221.7ms mention-in-
letters done in 221.7ms
```

```
[ 76] step= 8 info  node_open          mention-in-letters  224.2ms  open
LEAF:assemble final answer
[ 77] step= 9 info  subagent_start      assemble  0.0ms  assemble starting
[ 78] step= 9 info  decision            assemble  51.9ms  verdict=found groups=14
```

**Pro.** The stream is the process trace with milliseconds and intermediate decisions. Without it, cache hits, skip reasons, and order would be invisible. Structurally identical runs are collapsed to a single line, which keeps the log readable.

**Contra.** The event data JSON is often truncated to 220 characters — half readable, half redundant with §6. And when the metapartitur cache disagrees with the classifier, the stream shows the contradiction honestly but does not explain it.

### 6.1.6 §6 — ANNOTATED GOAL TREE

The sixth section is the join of the register goal tree (PLAN) and the actual run (RUN). It is the argumentative form — *this* goal was pursued, *that* alternative was rejected, *this* leaf is missing.

```
► [AND] ai-query-phase1#root          PLAN promising  RUN ok
├ ✓ analyse                          PLAN promising  RUN ok  27ms  in=1
out=1
├ ► [OR] ai-query-phase1#answer-from-register  PLAN promising  RUN ok
|   └ ○ correspondence-roster          PLAN promising(unused OR-alt)  RUN
not-taken
|   └ ✓ mention-in-letters              PLAN promising  RUN ok  222ms  in=1
out=14
|   └ ○ source-finder                  PLAN promising(unused OR-alt)  RUN
not-taken
├ ✓ goal-check                        PLAN promising  RUN ok  43ms  in=1
out=1
```

**Pro.** This section shows the OR alternatives that were not taken, the leaves that ran, and the plan-run agreement at the OR node. The `!! PLANNED-NOT-RUN` and `!! CONTRACT-MISS` markers make gaps visible.

**Contra.** The `PLANNED-NOT-RUN` markers mix genuine gaps with substituted paths — a leaf that was skipped because a specialist substituted for it reads the same as a leaf that was genuinely forgotten. The monitor should distinguish “really forgotten” from “substituted by a specialist”.

### 6.1.7 §7 — UNBOUND EXECUTION

The seventh section lists steps that ran but have no register goal node under the answering Partitur — plus the Phase-0 steps bound to the Understand score.

```
bound to the Phase-0 score  (ai-query-understand, played once in §1):
• understand                status=ok ms=1704.41
• classify                  status=ok ms=1507.52
• metapartitur             status=ok ms=2256.84  in=11 out=1
• partitur                 status=ok ms=348.22  in=1 out=11
```

**Pro.** The section makes the seam visible: Phase 0 and the metapartitur are execution reality but not leaves of `ai-query-phase1`. Without §7 they would appear in §6 as ghosts or vanish.

**Contra.** The list should carry a binding recommendation for each unbound step — “belongs to Phase-0”, “engine-internal control node”, or “genuine register gap”.

### 6.1.8 §8 — LLM CALLS

The eighth section lists every model invocation with its phase tag, model, stage, milliseconds, error, and prompt head.

```
n_calls = 2 (phase-0=1 execute=1) active_model=gemini-3.5-flash-lite
[00] phase=phase-0 model=gemini-3.5-flash-lite backend=gemini stage=fasttext-
classify ms=1464.2 err=None
[01] phase=execute model=gemini-3.5-flash-lite backend=gemini stage=metapartitur
ms=2207.9 err=None
```

**Pro.** Cost, model choice, and prompt provenance are all visible. The prompt head proves that no inline prompts were used — the SKILL.md path is in the text.

**Contra.** For audits the response head is often missing — the monitor shows the prompt but not the first characters of the model’s answer.

### 6.1.9 §9 — ANSWER / GOAL CHECK

The ninth section condenses the answer side: specialist fields, counts, goal-check score, and violated stage contracts.

```
specialist          = mention-in-letters
total_count         = 4
assemble_verdict    = found
goal_satisfied       = True
goal_score           = 0.7
stage contracts      = 5/6 stages met
  - correspondence-roster: FAILED
    roster_has_correspondents: contract check failed
top_entries:
  - Schöner, Johann (count=8)
  - Savorgnanus, Petrus (count=3)
```

**Pro.** The goal gate answers “was what was asked delivered?” The contracts from the Partitur register check the blackboard keys.

**Contra.** The section must show the answer itself more strongly — the term list with its grounding loci and viewer URLs — not just the metrics. And the contract failures are often schema noise: contracts belonging to the *correspondence-roster* specialist fail on a *mention-in-letters* run, because the goal check evaluates all register contracts, not just those of the active specialist.

### 6.1.10 §10 — VERDICT

The tenth section is one line of truth.

```
resolution = CLOSED (partitur=ai-query-phase1, leaves=30, missing=0)
execution  = 31 steps, 96 events, 2 llm calls, 3953 ms
join       = ran=24 planned-not-run=4 ran-not-planned=6 phase-0=2
answer     = count=4 goal_satisfied=True goal_score=0.7
```

**Pro.** The closing judgement without scrolling. `CLOSED` means goal resolution had no missing leaves — independent of whether the answer satisfied the goal.

**Contra.** The distinction between resolution-closed and goal-satisfied is correct and important, but it should be clearer in the badge — `[closed]` versus `[goal-fail]`.

## 6.2 What the monitor does well, and where it falls short

The monitor is a very good *observation* instrument and not yet a complete *explanation* instrument. It does well at: one source of truth (script output, not ad-hoc dump); Partitur before execution (text, choice, signals, then stream); the PLAN × RUN argumentative form; the specialist path without fake validation; and making term learning visible.

It falls short at: §4 persons without provenance (two “Schöner” entries unexplained); no answer evidence with `collection/doc_id/pidx`; the metapartitur-cache-versus-classify contradiction shown but not commented; and contract failures of the wrong specialist cluttering the goal check. These are the next sharpenings, in priority order:

- §4 persons as structured rows with surface term, entity id, and role;
- a cache-conflict line when metapartitur and classifier disagree;
- a §9 evidence block with grounding loci;
- a contract filter restricted to the active specialist;
- splitting `PLANNED-NOT-RUN` into `substituted` versus `missing`.

## 7 Reliability and Metrics

The golden set makes reliability measurable. Three metrics matter most: the recall of the general filter, the independent DuckDB audit, and the per-stage timing rollup.

### 7.1 Recall of the general filter

The general filter is the recall-first stage of the funnel: it gathers sentence and page candidates from the selected collections with *no false negatives* (false positives are acceptable, because the next stage interprets). The false-negative audit checks that every sentence the oracle says must be included is in fact included.

**Table 2** The general-filter false-negative audit of 8 August 2026, covering the then-canonical eleven goldens. Recall is 1.0 on every golden for which a sentence-level must-include set was stated. T4 is not in this audit.

| Golden               | Verdict | Filter n | Must-include | FN | Recall             |
|----------------------|---------|----------|--------------|----|--------------------|
| R1                   | PASS    | 5032     | 86           | 0  | 1.0                |
| R4                   | PASS    | 2        | 2            | 0  | 1.0                |
| R2                   | PASS    | 2407     | 45           | 0  | 1.0                |
| R5                   | PASS    | 1656     | 1228         | 0  | 1.0                |
| ground-pirckheimer-1 | SKIP    | 362      | —            | —  | no sentence oracle |
| ground-pirckheimer-2 | PASS    | 82       | 29           | 0  | 1.0                |
| K1                   | SKIP    | 1500     | —            | —  | unknown            |
| K2                   | SKIP    | 2181     | —            | —  | unknown            |
| K3                   | SKIP    | 1543     | —            | —  | unknown            |
| K4                   | SKIP    | 3599     | —            | —  | unknown            |
| K5                   | SKIP    | 3171     | —            | —  | unknown            |

Table 2 reports a scoped result, not a suite-wide one. On 8 August 2026 the recall-first filter dropped none of the sentences the must-include sets required, for the five goldens that have such a set (R1, R4, R2, R5,

and ground-pirckheimer-2’s saphea pages). The must-include cardinalities are the register probes recorded in the manifest `truth` notes: R1 union 86, R2 kepler-prim 45, R5 the sampled floor against a 5,146-sentence tycho% truth. The six SKIP rows are goldens for which “which sentences must the filter not drop?” does not apply: ground-pirckheimer-1 is a correspondent count against `letters_resolved`; K1–K5 retrieve letters or persons, not sentences. The two T4 goldens, promoted after this audit, have not yet been run through the sentence-level false-negative script.

7.2 The independent DuckDB audit

The independent audit queries the source layer without calling the orchestrator. Table 3 is the audit of 2026-08 against `letters_resolved` / `letter_pages` / `stz_sentences`.

Table 3 The independent DuckDB audit of the two canonical ground goldens and the Chapter 8 candidate.

| Golden               | Status    | Verdict | Summary                            |
|----------------------|-----------|---------|------------------------------------|
| ground-pirckheimer-1 | canonical | PASS    | oracle=75 duckdb=75 letters=354    |
| ground-pirckheimer-2 | canonical | PASS    | letters=11/11 saphea at-tested     |
| ground-pirckheimer-3 | candidate | PASS    | correspondents=4, index_letters=13 |

The two canonical ground goldens match their count oracles: 75 correspondents and 11 letters with the saphea topic. The third row is the candidate used in Chapter 8, not a third canonical ground golden. The audit shares the register schema and, for the year window, the specialist’s predicate; it does not share the orchestrator, the Partitur, or any model call.

7.3 The per-stage timing rollup

The golden set also makes the *cost* of an answer measurable. Table 4 rolls up `timing_by_subagent` over the thirteen canonical goldens in the orchestrator audit of 12 August 2026 (`orchestrator_audit.json`, `generated_at 2026-08-12T15:55:40Z`). Unnamed internal steps are omitted.

**Table 4** Per-stage timing rollup over the thirteen canonical goldens in the 12 August 2026 orchestrator audit.

| Stage                 | n  | ms    | n_out | Role                       |
|-----------------------|----|-------|-------|----------------------------|
| understand            | 13 | 9043  | 17    | Phase-0 classification     |
| normalize             | 13 | 6840  | 30    | Query normalisation        |
| terminology-find      | 6  | 10684 | 0     | Term attestation           |
| concept-statements    | 5  | 4215  | 1087  | Statement extraction       |
| correspondence-roster | 5  | 2258  | 429   | Register roster            |
| context-gather        | 13 | 1642  | 17    | Session and corpus context |
| general-fast-filter   | 13 | 1227  | 21154 | Recall-first gather        |
| postprocess           | 13 | 789   | 253   | Answer writing             |
| assemble              | 13 | 718   | 422   | Evidence grouping          |
| mention-in-letters    | 3  | 712   | 11    | Mention lookup             |
| letter-subject        | 3  | 633   | 11    | Letter summary             |
| goal-check            | 13 | 297   | 13    | Goal verdict               |
| analyse               | 13 | 179   | 13    | Query-object parsing       |
| partitur              | 13 | 113   | 104   | Fast rule-based analysis   |
| route                 | 13 | 70    | 13    | Retrieval planning         |

Table 4 shows where the time went in that run. The two heaviest named stages are `terminology-find` and `understand`. The register specialists are in the low seconds: `correspondence-roster` 2,258 ms across five goldens, `mention-in-letters` 712 ms across three. These suite totals are not the same measurement as the monitor log in Chapter 6: that log is a single `mention-in-letters` run (`ground-pirckheimer-3`) that returned `n_out=14` in 221.7 ms, a different query and a different day.

## 7.4 The suite rollup

The thirteen canonical goldens, taken as the subset of the 12 August 2026 orchestrator audit whose ids are in `manifest.yaml`, give the headline numbers:

**Table 5** The thirteen-golden canonical subset of the 12 August 2026 orchestrator audit.

| Metric      | Value              |
|-------------|--------------------|
| Goldens run | 13 (canonical set) |
| Wall time   | 53,370 ms          |
| p50         | 2,847 ms           |
| p95         | 14,647 ms          |
| Oracle PASS | 13                 |
| Oracle FAIL | 0                  |
| Errors      | 0                  |

Every canonical golden in that run carried a count oracle or a floor oracle and received PASS. Floor oracles for the catalog and karman tiers were added in August 2026; an earlier eleven-golden run had reported Oracle N/A for those nine because they then had shape checks only. The same 12 August harness also ran thirteen further candidates (not in the canonical set); those twenty-six rows together recorded 26 PASS, 0 FAIL, 0 N/A, 0 ERROR, wall time 124,425 ms. Table 5 reports the canonical thirteen, not the harness twenty-six.

## 8 The Learning Loop: From Monitor to Promoted Patch

The monitor log and the reliability metrics are diagnostic: they show *that* a golden failed and *where*. They do not, by themselves, fix anything. The learning loop is the mechanism that closes that circle — it turns an annotated deficit in a monitor log into a versioned, A/B-gated patch to the pipeline itself, and it does so without ever trusting the patch until the patch has re-earned the golden set’s verdict.

### 8.1 Two entry points, one gate

The loop has two paths into the same decision logic.

#### METHOD Path A — metric-driven tunable loop

**Trigger:** automatic, or `python -m ai_query.learn.loop run`.

**Target:** a subagent’s tunable YAML (never a `SKILL.md`, never `partitur.yaml`).

**Selection:** the worst-scoring subagent with `support_n > 0` and a tunable file, ties broken by highest `warn+error` count.

**Objective:** `aggregate = mean over subagents with support_n > 0 of their mean_score`.

#### METHOD Path B — annotation-driven instruction/partitur loop

**Trigger:** a user annotates a specific stage of a specific run via the Resolve tab, then `POST /api/learn/from-annotations`.

**Target:** determined by the annotation’s `target` field — a `SKILL.md` instruction, or a subagent’s `partitur.yaml`.

**Objective:** for a `SKILL.md` patch, the subagent’s own score; for a `partitur.yaml` patch, the aggregate score across all subagents, because a `partitur` governs a whole agent, not one stage.

Both paths share the same gate: a patch is **kept** only if the new score exceeds the old one and the aggregate has not regressed by more than a one-percent tolerance. Every patch is snapshotted before it is applied — to `assessment/versions/` for tunables, to `assessment/instruction_versions/` for `SKILL.md` files, and to `assessment/partitur_versions/<partitur_id>/` for `partitur` YAMLs — so a rejected or later-superseded patch can always be restored exactly.

```
# loop.py – the partitur A/B gate
improved      = new_agg > baseline_agg
no_regression  = new_agg >= baseline_agg - 0.01 # 1% tolerance
kept = improved and no_regression
```

### 8.2 Worked example: ground-pirckheimer-3, Georg Hartmann

This is the same query family as K3 in Section 4 — mention lookup inside a correspondence — but a different golden: `ground-pirckheimer-3` asks “Which correspondent mentions Georg Hartmann in his letters to Pirckheimer?” in the Pirckheimer corpus rather than K3’s Einstein-in-Karman query. It is a candidate, not one of the canonical thirteen; Table 3 already records the register fact the pipeline was missing: four correspondents and thirteen index-cited letters.

**The deficit.** The monitor log’s §4 Signals for run `de7b615a0c5040b5` showed the `person-classify` stage resolving

```
persons = ['Pirckheimer, Willibald', ..., 'Sauermann, Georg']
```

Georg Hartmann is absent; Sauermann, Georg is present but wrong. The cause: `person-classify`'s registry lookup matched the single token `Georg` against `Sauermann, Georg` by substring before it tried the whole phrase `Georg Hartmann` against the registry's own `Hartmann, Georg` entry. The baseline scorecard put `person-classify` at `mean_score = 0.60`, the worst of four active subagents — the independent audit (Table 3) already agreed: it finds four correspondents and thirteen index-cited letters, not the single substring match the pipeline was producing.

**The annotation.** A user annotated the run with `target: "person-classify:partitur"`, `persons_should_include: ["Hartmann, Georg"]`, `persons_should_not_include: ["Sauermann, Georg"]`, and a comment naming the substring-priority cause.

**The patch.** The loop's `_propose_partitur_patch()` sent the current `partitur.yaml`, the annotation, and a minimal-change instruction to the LLM. The proposal added one iron rule to the stage goal, one new parameter (`whole_phrase_priority: true`), and one new evaluator (`phrase_match_priority`) — no stage reordering, no other evaluator touched.

**The gate.** Two steps, not one. The fast gate re-ran the three failing-support queries: as recorded in the version row `ann_golden3_001 → person-classify v1.1`, `ground-pirckheimer-3` rose from `mean_score 0.60` to `0.95`; the other two were unchanged. The full-suite step then re-ran the then-canonical eleven goldens and recorded aggregate `0.873` before the patch and `0.905` after. The printed gate in `loop.py` keeps a patch only if the new aggregate exceeds the old one and has not fallen by more than one percent; both conditions held, so `person-classify v1.1` was promoted to `active` and `v1.0` was marked `superseded`. The fast gate does not by itself compare against the whole suite; the full-suite step is what does.

**The result.** Re-running the golden afterward, §4 Signals now reads `persons = [..., 'Hartmann, Georg']` — `Sauermann, Georg` no longer appears. The annotation was marked resolved; the version history records the whole chain (`ann_golden3_001 → person-classify v1.1`, `score_before=0.873`, `score_after=0.905`), so the fix is auditable in exactly the way the golden set demands of the pipeline it patches.

### 8.3 Discussion: what the loop buys, and what it risks

**In favour.** The loop never proposes a patch without a logged deficit to justify it — no speculative tuning. Every change is versioned and trivially reversible (`rollback-partitur person-classify v1.0`). The *full-suite* step of the gate compares against the whole canonical set, not just the failing case, so a patch that fixes one golden while breaking another is caught before promotion. The fast gate, which runs only the failing subagent's support set, does not provide that guarantee. The annotation record and the version row stay linked, so a later reader can trace any live behaviour back to the observation that motivated it.

**Against.** The fast gate runs only the failing subagent's support set (here, three queries) before the expensive full-suite check — a patch could pass the fast gate by overfitting to those three and still fail the full suite, which is why the full-suite step exists, but it means every promotion candidate pays the cost of the whole canonical run. The LLM's patch proposal is bounded by a minimal-change instruction, but "minimal" is judged by the model, not verified structurally beyond "is this valid YAML with the expected top-level keys" — a plausible but subtly wrong patch could still pass validation. And the loop refuses to retry a validation failure automatically: a human must sharpen the annotation and restart it, which is a deliberate brake against runaway iteration, but it does mean a bad annotation can stall the loop rather than self-correct.

## 9 Extending the Set

The golden set is designed to grow. New queries are introduced, tested against their oracles, and promoted into the canonical set as the platform's scope widens. The extension protocol is deliberately simple:

1. **Introduce.** A new query is added as a *candidate* — a golden with an expected shape but no verified oracle.

2. **Test.** The candidate is run through the pipeline and the monitor. The trail is inspected for the right intent, the right trail, and a plausible result set.
3. **Verify.** Where the query has a countable answer, the independent DuckDB audit confirms it. Where it has a subject, the oracle is a loose substring check.
4. **Promote.** A candidate whose oracle is verified and whose run is clean is promoted to `status: golden` and joins the canonical set.

The set already shows the pattern. The concept tier — T4-kepler-motrix and T4-euler-aether — was promoted into the canonical thirteen after floor oracles were written from the 12 August 2026 register probes. The karman tier grew from the partitur benchmark’s proven shapes, including the German letter-count K2. Each addition widened the coverage of the query families without breaking the existing goldens.

The extension protocol is the platform’s answer to a real risk: that the golden set becomes a closed exam the pipeline is tuned to pass. The protocol keeps the set open; the independent audit keeps the count oracles as facts about the source layer.

### 9.1 Candidates still under test: terminology-find

The seven query families of Section 4 are not the ceiling. Two candidates still probe a different shape: “Where does Euler discuss force in E57?” and “How does Euler formulate Newton’s law of motion in E57?” — both scoped to Enestroem document E57 (*Theoria motus corporum solidorum seu rigidorum*, 1765). Unlike the lemma and attestation goldens, these ask the pipeline to locate a *concept*, not a single lemma: “force” spans `vis / vires` (Latin), `Kraft / Kräfte` (German), and `force / forces` (French); “Newton’s law of motion” spans `lex motus` and its vernacular equivalents [19].

These candidates are still candidates. Their `expected_answer` is a lower bound on sentence count (`n_sentences.min: 10` and `.min: 5` respectively), not a verified oracle, and their notes say so explicitly: “Promote after GLiNER2 gold control confirms the true positive set.” The intended grounding method is a sentence-level gold control set built by GLiNER2 labelling of E57’s own sentences, against which precision and recall can be measured directly. Recording this step before promotion is the extension protocol applied: a candidate’s status is not upgraded until its verification method has actually run.

## 10 Conclusion

A research platform is only as trustworthy as the questions it can be held to. The golden queries of AI Erasmus are that holding: a canonical set of thirteen scholarly questions, each with a declared form of inquiry, a grounding document, and an oracle — a count oracle where a number was derived by hand from the register, a floor oracle where a register probe supplies a minimum.

The set’s design is its argument. The thirteen goldens span four corpora, three languages, and seven query families — lemma co-occurrence, attestation, person mention, correspondence roster, letter subject, place-and-person lookup, and concept statements. The ground tier matches verified numbers; the catalog, karman, and concept tiers clear register floors. On 8 August 2026 the general filter achieved recall 1.0 on the five goldens that have a sentence-level must-include set. On 12 August 2026 the thirteen canonical goldens all received oracle PASS in the orchestrator audit, in 53,370 ms wall time. The independent DuckDB audit confirms the two canonical ground goldens as facts about `letters_resolved`.

The monitor log makes a run inspectable, section by section. The learning loop turns an annotated deficit in that log into a versioned, A/B-gated patch — fast gate on the failing support set, full-suite step on the canonical set — as the Georg Hartmann case records. The extension protocol keeps the exam open: T4 joined the canonical thirteen; the E57 terminology-find queries remain candidates until their GLiNER2 gold control has run.

The Famulus Handlungsmodell (described in G. Graßhoff [3]) extends the scope of what the goldens audit: they now test not only the filter stage but every Partitur in the iterative expansion loop — the correspondence-roster specialist, the terminology finder, the entity roster. The gate remains the same: no Partitur enters the

repertoire without a PASS on the relevant canon members, measured from a baseline taken *before* the draft. The golden set is the mirror in which every new repertoire entry must see itself.

The goldens can also be read and run as a notebook. The generator of Famulus Notebook writes one block per golden, with its collections and oracle in the preamble and its role, ground, and expected route in the paragraph before it; a run of that notebook is a run of the golden check whose result stays as a document [20].

The workshop in Basel is staffed; the golden set is how we know the staff is doing its job.

## 11 Acknowledgements

Thanks are due to the archives and editorial projects whose editions underpin the golden queries, and to collaborators in the AI Platform project for the index, viewer, query, and orchestrator infrastructure.

## Bibliography

- [1] G. Graßhoff, “The AI Partitur — A Score Language for the Humanities,” 2026.
- [2] G. Graßhoff, “The AI Agent Orchestrator: Partitur, Conductor, and Harvest,” 2026.
- [3] G. Graßhoff, “AI Erasmus — A Platform for a New Era in Digital Humanities,” 2026.
- [4] G. Graßhoff, “Wittgensteinian Grammar and OED Dictionaries for AI,” 2026.
- [5] L. Wittgenstein, *Philosophical Investigations*. Oxford: Blackwell, 1953.
- [6] G. Graßhoff, “The AI Query Harness: Goal Resolution, Specialists, and the Golden Set,” 2026.
- [7] G. Eneström, *Verzeichnis der Schriften Leonhard Eulers*. in Jahresbericht der Deutschen Mathematiker-Vereinigung, Ergänzungsband-IV. Leipzig: Teubner, 1910.
- [8] W. Pirckheimer, *Briefwechsel*. München: C. H. Beck, 1940.
- [9] California Institute of Technology Archives, “Theodore von Kármán Papers: Finding Aid.”
- [10] P. Qi, Y. Zhang, Y. Zhang, J. Bolton, and C. D. Manning, “Stanza: A Python Natural Language Processing Toolkit for Many Human Languages,” in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, Association for Computational Linguistics, 2020, pp. 101–108. doi: 10.18653/v1/2020.acl-demos.14.
- [11] J. Nivre and others, “Universal Dependencies v2: An Evergrowing Multilingual Treebank Collection,” *Proceedings of LREC*, 2020, doi: 10.48550/arXiv.2004.10643.
- [12] M. Raasveldt and H. Mühleisen, “DuckDB: An Embeddable Analytical Database,” in *Proceedings of the 2019 International Conference on Management of Data (SIGMOD)*, ACM, 2019, pp. 1981–1984. doi: 10.1145/3299869.3320212.
- [13] J. Kepler, *Gesammelte Werke*. München: C. H. Beck, 1938.
- [14] G. Graßhoff, “AI Scholar: From PDF to Portable Document AI Infrastructure,” 2026.
- [15] G. Graßhoff, “A Multi-Layer Index and Retrieval Architecture for Historical Document Collections,” 2026.
- [16] L. Euler, *Opera Omnia, Series-IV: Commercium epistolicum*. Basel: Birkhäuser / Springer, 1975.
- [17] Oxford English Dictionary, *The Oxford English Dictionary*, 2nd ed. Oxford: Oxford University Press, 1989.
- [18] G. Graßhoff, “Querying with AI Linguistic Trees: Stanza-Backed Dependency Parses, Lemma Normalisation, and DuckDB,” 2026.
- [19] U. Zaratiana, N. Tomeh, P. Holat, and T. Charnois, “GLiNER: Generalist Model for Named Entity Recognition using Bidirectional Transformer,” in *Proceedings of the 2024 Conference of the North American*

*Chapter of the Association for Computational Linguistics*, Association for Computational Linguistics, 2024.

- [20] G. Graßhoff, “Famulus Notebook: The Notebook as the Principal Medium of Scholarly Querying, Checking, and Development, Federated Across Machines,” *arXiv preprint*, 2026.