MDReader as a Reference Application for Categorical Schema Modeling: Typed Markdown Projections, Inspectable Mappings, and Evidence Boundaries

Matthew Long
The YonedaAI Collaboration
YonedaAI Research Collective
Chicago, IL
matthew@yonedaai.com · https://yonedaai.com

23 July 2026

Provisional design and acceptance contract.
The audited repository contains no MDReader package or application result.

Abstract

Markdown is convenient precisely because it leaves many relationships implicit. A heading contains paragraphs by position. A claim may cite evidence in another file. A decision may satisfy a requirement, invalidate a task, or change an architectural dependency without declaring any of those relations in a database schema. Search can recover similar passages, but similarity alone does not expose typed paths, mapping obligations, constraint witnesses, provenance, or the semantic effect of a revision.

This paper specifies MDReader, a reference application for testing how categorical schema models could make those relationships inspectable while ordinary Markdown files remain authoritative. The proposed application projects a corpus into versioned document structure, layers typed overlays for claims, evidence, decisions, requirements, tasks, architecture, and agent context, and exposes mappings among these views. It defines contracts for object inspectors, path and mapping exploration, migration visualization, constraint violations, provenance, historical queries, and semantic diffs. Local execution is specified in terms of transactional SQLite storage, with optional PostgreSQL synchronization governed by an explicit compatibility and conflict protocol. Retrieval is deliberately hybrid: lexical, vector, and typed relationship signals remain distinguishable, and embeddings are never treated as identity, truth, authority, provenance, or proof.

The contribution is a falsifiable application contract, not an implementation report. At the audited repository revision, `apps/mdreader/` is empty and has no package history. There is no Markdown parser, user interface, database backend, server API, search index, provenance executor, semantic-diff engine, browser suite, or synchronization layer. The accepted upstream computational boundary covers finite schemas, typed paths, total mappings, finite instances, and bounded Delta evaluation over a closed fixture corpus. It does not provide an application or database runtime. We therefore present evaluation protocols rather than results: seeded end-to-end tasks, retrieval ablations, independent semantic oracles, accessibility checks against WCAG 2.2, preregistered usability tasks, storage recovery tests, and optional synchronization failure injection. Exact nonclaims, dependency status, and twenty falsifiers make clear what future evidence must establish before MDReader can be called a working reference application.

Contents

1 Introduction

1.1 The gap between finding text and understanding a corpus

Consider a research repository with a design note, an implementation roadmap, a release record, and a review. A text search for "migration" returns all four. A vector search may rank two semantically similar passages near the top. Neither result answers a more precise question: which accepted decision satisfies requirement R7, through which mapping, at which source revisions, and what changed when the migration schema was edited?

That question has structure. Files contain sections and blocks. Some blocks are claims; others are evidence or decisions. A requirement can be linked to a verification artifact. A mapping can send a source relation to a path through a target schema. A revision can preserve the text of a heading while changing the typed object it denotes. A constraint failure should point to a witness, not merely color a node red.

The CatDB research program studies typed, versioned schemas and mappings. Its Markdown context proposes a viewer and search system that makes categorical structure visible without replacing ordinary files with a proprietary database [2]. The program manifest assigns that application to Paper 18, with dependencies on categorical foundations, compilation, queries, migration safety, information loss, provenance, and the Rust runtime [3]. The implementation roadmap places the server API and MDReader in an open future slice [4].

That chronology matters. An application paper can easily slide from "the interface should show a mapping" to "the interface shows a mapping." This paper does not make that slide. It defines what a reference application would have to expose and how to test it. The repository audit found no MDReader package, tracked application file, package manifest, test suite, or history under `apps/mdreader/`. The planned command `npm --prefix apps/mdreader test` exits before testing because `package.json` is absent.

Evidence status. ABSENT/OPEN for the application; ACCEPTED UPSTREAM only for the bounded semantic dependency described in Section 2..

1.2 Why call a proposed interface a reference application?

A reference application is useful when it forces a semantic design to answer operational questions. Where does an object come from? Which source span does an arrow denote? How can a user distinguish a proposed mapping from a validated one? What does an inaccessible graph conceal? How are historical results reconstructed? What happens if indexing stops halfway through a revision?

Here, "reference" names a conformance role, not an implementation quality claim.

Definition 1.1 (Reference application contract)**.** *A reference application contract is a collection*

$$\mathfrak{R} = (\mathcal{S}, \mathcal{P}, \mathcal{O}, \mathcal{M}, \mathcal{V}, \mathcal{E}, \mathcal{D}, \mathcal{Q}, \mathcal{L}, \mathcal{Y})$$

of source, projection, overlay, mapping, provenance, violation, diff, query, local-storage, and synchronization interfaces, together with observable invariants and falsifiers. An implementation conforms only for the capabilities that pass their native gates against a named version of $\mathfrak{R}$*.*

This definition lets the paper make a concrete contribution before claiming a product. It also prevents a mock-up, a JSON example, or a passing semantic library test from being mistaken for end-to-end application evidence.

1.3 Contributions

The paper contributes:

1. an evidence vocabulary separating accepted upstream computation, accepted paper contracts, provisional application requirements, absent capabilities, and nonclaims;
2. a source-preserving projection model for CommonMark or explicitly selected Markdown profiles;
3. versioned typed overlays for claims, evidence, decisions, requirements, tasks, architecture, and agent context;
4. inspector contracts for objects, paths, mappings, migrations, violations, provenance, histories, and semantic diffs;
5. a local SQLite execution protocol and an optional PostgreSQL synchronization protocol;

6. a hybrid retrieval design that explicitly rejects vector similarity as semantic authority;
7. correctness, retrieval, usability, accessibility, performance, and synchronization evaluation plans; and
8. exact dependencies, nonclaims, threats, and falsifiers.

No item in this list is reported as implemented.

2 Evidence boundary

2.1 Five labels

Table 1 defines the labels used throughout the manuscript. They appear near capabilities rather than in a single disclaimer at the end.

Table 1: Evidence labels. Acceptance of a paper and acceptance of code are different facts.

Label	Meaning
ACCEPTED UPSTREAM	A bounded CatDB result has native repository evidence and an accepted release record.
ACCEPTED PAPER CONTRACT	An accepted manuscript supplies definitions or an acceptance contract, but not necessarily an implementation.
PROVISIONAL CONTRACT	This manuscript specifies observable future behavior.
ABSENT/OPEN	The audited repository lacks the capability, artifact, or gate.
NONCLAIM	The manuscript explicitly declines a possible interpretation.

2.2 What the repository contains

The correction audit used repository base commit `e2f7cd70217e079af70c3e2ccaca14e6f548fd23`, dated 23 July 2026. Its working tree also contained uncommitted paper work and later Slice 5 candidate fixtures. The Cargo workspace contains six packages: core identifiers and errors, an intermediate representation, schema operations, mapping operations, bounded migration operations, and a CLI. No tracked file exists beneath `apps/mdreader/`; no package manifest or example corpus exists; and the path has no Git history.

The accepted Slice 4 release is pinned to `056915834fcc2f172a4a72efe3629f4f753e45cf`. Its closed corpus records 33 cases: 29 semantic comparisons, 4 structural comparisons, and zero disagreements [5]. These computations cover finite schemas, typed paths, total mappings, finite instances, and a bounded Delta operation.

The current working tree is not that release checkout. Its all-fixtures command reports 42 cases: the same 29 semantic comparisons, 13 structural comparisons, and zero disagreements. Nine later Slice 5 candidate fixtures for schema diff and information-loss analysis account for the larger structural total. Their presence changes the current corpus, not the accepted Slice 4 release, and does not supply MDReader application evidence. The current `cargo test --workspace --all-features` run also passes, but only for the semantic workspace.

Structural cases are important. Schema-diff and information-loss documents can be decoded and compared structurally without executing their meaning. A future interface must show that distinction. A polished diagram cannot upgrade structural compatibility into semantic evaluation.

2.3 What the repository does not contain

A focused workspace and lockfile audit found no SQLite or PostgreSQL driver, Rust HTTP server, browser framework, desktop shell, or Playwright dependency. Negative dependency search is not a proof about all possible code, but it agrees with the empty application path and open roadmap status.

Table 2: Current capability boundary relevant to MDReader.

Capability	Status	Consequence for this paper
Finite schemas, paths, mappings	ACCEPTED UPSTREAM	May define semantic examples within the accepted profile.
Finite instances and bounded Delta	ACCEPTED UPSTREAM	May be displayed only after a future API reproduces the accepted library result.
Schema diff and information loss	structural only	The interface must not label decoded shapes as computed analysis.
Query IR or compiler	ABSENT/OPEN	Historical and typed queries are contracts, not results.
SQL, SQLite, PostgreSQL	ABSENT/OPEN	Storage and sync sections are protocols.
Provenance executor	ABSENT/OPEN	Provenance records are proposed data and validation surfaces.
Server or application API	ABSENT/OPEN	No library/API equivalence result.
MDReader UI and E2E suite	ABSENT/OPEN	No usability, accessibility, or browser claim.
General Sigma, Pi, or Kan migration	ABSENT/OPEN	Migration views must display unsupported operators.

2.4 Paper dependencies

The manifest names Papers 1–4, 9, 13, and 15 as dependencies. At the correction-audit cut, Papers 1–4, 9, and 13 are internally accepted manuscripts. Paper 15 remains unaccepted. Manuscript acceptance is distinct from acceptance of an executor, application, or full Rust runtime. In particular, the accepted migration, integration, provenance, and information-loss manuscripts do not imply that any corresponding engine exists.

Table 3: Dependency ledger.

Item	Status	Permitted dependence	Open condition
P1	accepted paper	schema, path, and mapping vocabulary	no application implementation follows from the paper
P2	accepted paper	physical-plan contract and its limits	no accepted P2 vertical slice, query engine, or database backend
P3	accepted paper	typed migration-map definitions and obligations	no migration executor, loss detector, diff engine, or backend runner

Item	Status	Permitted dependence	Open condition
P4	accepted paper	compositional integration contract and safety conditions	no integration operator, factorization checker, connector, or identity resolver
P9	accepted paper	information-loss questions and obligations	no accepted loss-analysis runtime
P13	accepted paper	provenance record and explanation contract	no accepted provenance executor
P15	provisional paper	potential Rust application boundary	only a bounded semantic slice is accepted separately
Slice 14	open roadmap	future server and MDReader gates	package, API, E2E, equivalence, and review artifacts absent

If a dependency changes, the corresponding MDReader schema, stored derived data, and acceptance fixtures must be versioned or invalidated.

3 Ordinary Markdown as source authority

3.1 Profiles, revisions, and spans

CommonMark supplies a precise core syntax [6]; GitHub Flavored Markdown adds tables, task-list items, strikethrough, and autolinks [9]. A real corpus may also contain front matter, mathematical extensions, HTML, wiki links, diagram blocks, or local conventions. Calling all of these "Markdown" hides meaningful parser choices.

Definition 3.1 (Observed source revision)**.** *An observed source revision is a tuple*

$$r = (i, w, p, h, b, \pi, \nu, t, d),$$

where i is a stable source identifier, w a workspace identifier, p the observed relative path, h a content hash, b the exact bytes, π a parser profile, ν a parser version, t observation metadata, and d parse diagnostics.

The content hash helps detect byte equality. It is not by itself the human identity of a document. A file can move without becoming a new intellectual object, and two copied files can have the same bytes without having the same role. Identity policy must be explicit and testable.

A span is a half-open byte interval plus derived line and column coordinates:

$$s = (r, [a, b), \ell_a, c_a, \ell_b, c_b).$$

Byte offsets are authoritative for exact recovery from b. Lines and columns are presentation aids that depend on newline and Unicode conventions.

Invariant 3.2 (Source preservation)**.** *For an observation and projection operation with no separately authorized write action,*

$$\text{bytes}_{after}(r) = \text{bytes}_{before}(r).$$

Every projected node span must resolve to the exact bytes used to derive that node.

3.2 Projection is derived data

For a selected parser profile, let

$$\mathcal{P}_{\pi,\nu}(r) = I_{\mathcal{D}}$$

produce an instance of a document schema $\mathcal{D}$. A minimal $\mathcal{D}$ has objects

$$\mathsf{Doc}, \mathsf{Sec}, \mathsf{Block}, \mathsf{Link}, \mathsf{Task}$$

and arrows such as

$$\mathsf{Sec} \xrightarrow{\text{document}} \mathsf{Doc}, \quad \mathsf{Sec} \xrightarrow{\text{parent}} \mathsf{Sec}, \quad \mathsf{Block} \xrightarrow{\text{section}} \mathsf{Sec}, \quad \mathsf{Link} \xrightarrow{\text{block}} \mathsf{Block}.$$

Each object also maps to Span and Revision. Order is explicit rather than inferred from database row order.

Parse recovery can be useful, but it changes the evidence state. If an unterminated fence is recovered as one code block, the interface must show the diagnostic and the profile that produced it. A user should be able to view the raw bytes alongside the projected node.

Protocol 3.3 (Projection conformance)**.** *A conforming projection suite shall:*

1. *freeze parser profile and version;*
2. *include CommonMark examples, selected GFM extensions, and repository dialect fixtures;*
3. *compare object kinds, hierarchy, order, content, destinations, and exact spans to an independent expected-results manifest;*
4. *test malformed and recovered syntax;*
5. *test Unicode, mixed newlines, large blocks, and embedded HTML;*
6. *prove source preservation by pre/post byte hashes; and*
7. *retain old projections for historical queries or mark their migration explicitly.*

Evidence status. PROVISIONAL CONTRACT. No parser, watcher, projection store, or projection fixture suite is present..

4 Typed overlays without hidden rewriting

4.1 From blocks to assertions

A paragraph does not announce whether it is a claim, evidence, decision, requirement, task, or background. A heading hierarchy alone cannot supply that meaning. MDReader therefore separates source projection from typed overlay.

Let $\mathcal{O}$ contain objects

$$\mathsf{Claim}, \mathsf{Evidence}, \mathsf{Decision}, \mathsf{Requirement}, \mathsf{Task}, \mathsf{Component}, \mathsf{AgentContext}$$

with relations such as

$$\mathsf{Evidence} \xrightarrow{\text{supports}} \mathsf{Claim}, \qquad \mathsf{Decision} \xrightarrow{\text{satisfies}} \mathsf{Requirement}, \qquad \mathsf{Task} \xrightarrow{\text{implements}} \mathsf{Decision},$$

$$\mathsf{Component} \xrightarrow{\text{dependsOn}} \mathsf{Component}, \qquad \mathsf{AgentContext} \xrightarrow{\text{selected}} \mathsf{Block}.$$

An overlay instance is not recovered solely by parsing Markdown. It is a set of versioned assertions anchored to projected objects or spans.

Definition 4.1 (Overlay assertion)**.** *An overlay assertion is*

$$o = (j, \tau, a, \rho, \sigma, q, g, c),$$

where j is an identity, τ a type, a source anchors, ρ its relations, σ status, q authority, g generating activity and agent, and c creation and schema-version metadata.

The status set includes `suggested`, `reviewed`, and `approved`. It also records `rejected`, `superseded`, and `invalidated`. The authority field distinguishes an author's assertion, a reviewer decision, a deterministic extractor, and a model suggestion.

Invariant 4.2 (No authority laundering)**.** *A machine-generated suggestion cannot become an approved overlay merely because it is displayed, indexed, copied, or used by another machine operation. The approving transition must identify its actor, input revision, time, and resulting overlay revision.*

4.2 Anchors can go stale

An exact span may fail after editing. A heading-derived anchor may collide. A content fingerprint may match copied text. The application should retain multiple anchor signals and report how an anchor was resolved:

- exact source revision and byte span;
- stable projected-object identifier;
- heading or structural path;
- local context fingerprint; and
- explicit manual reattachment history.

No heuristic reattachment should be silent. A low-confidence anchor is a violation or review item, not a normal resolved edge.

Example 4.3 (A changed requirement)*.* A roadmap block at lines 120–126 is typed as requirement R_7. The file is edited, moving the block to lines 133–139 and changing "may" to "must." The text-level diff shows the word change. The overlay history shows whether the old requirement was revised, superseded, or replaced. A semantic diff then reports which decisions and tasks still claim to satisfy R_7. Merely finding the new word "must" cannot answer that question.

Evidence status. PROVISIONAL CONTRACT. There is no accepted overlay extractor, editor, authority workflow, or anchor-repair implementation..

5 Mappings as inspectable application objects

5.1 A mapping must expose its assignments

Functorial data migration uses mappings between schemas to induce data migration operations [21]. In an application, the first duty is more modest: show what a selected mapping says and which obligations justify its status.

For source schema S and target schema T, a CatDB mapping $F : S \to T$ sends source objects to target objects and source arrows to typed target paths. If

$$\mathsf{supports} : \mathsf{Evidence} \to \mathsf{Claim}$$

is mapped to

$$\mathsf{Evidence} \xrightarrow{\text{anchor}} \mathsf{Block} \xrightarrow{\text{section}} \mathsf{Sec} \xrightarrow{\text{documentsClaim}} \mathsf{Claim},$$

the user needs the full path, not the vague caption "evidence maps to claims."

Protocol 5.1 (Mapping inspector)**.** *For mapping $F : S \to T$, a conforming inspector shall expose:*

1. *identities and versions of S, T, and F;*
2. *every object assignment;*
3. *every arrow and attribute assignment;*
4. *each mapped path with endpoint typing;*
5. *source equations and their mapped obligations;*
6. *totality, coverage, validation, and execution status;*
7. *unsupported or structurally decoded portions;*
8. *links to affected source and overlay objects; and*
9. *the library, API, and fixture revision behind a computed result.*

The normalized record shall be available as structured text. A graph is a secondary presentation, not a separate authority.

5.2 Graph, table, and path views

Figure 1 sketches three synchronized views. The graph provides orientation. The assignment table exposes completeness. The path panel explains one selected arrow and its obligation. Selection in any view must identify the same normalized mapping record.

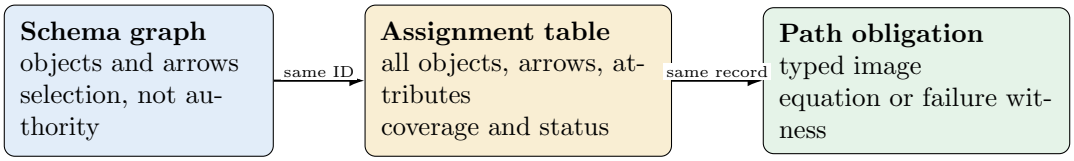

Figure 1: Proposed synchronized mapping inspector. No such interface is present at the audited revision.

The graph must not hide parallel arrows, path length, unmapped source components, or equation failures. Color cannot be the only carrier of status. A screen-reader user needs a traversable object list, assignment table, and path sequence with equivalent operations.

5.3 Computed, validated, and proposed are different states

A mapping record can pass JSON decoding, pass structural validation, pass endpoint typing, pass equation obligations, or execute a migration. These states are not interchangeable. The application should expose a monotone evidence ladder:

$$\text{observed} \preceq \text{decoded} \preceq \text{typed} \preceq \text{validated} \preceq \text{executed},$$

where an implementation may skip no required evidence while keeping the higher label. A failed or stale prerequisite invalidates the higher state.

The current accepted semantic slice can validate mappings within its bounded profile. A future server result would still need to demonstrate that its normalized request and response match the library computation.

6 Migration visualization

6.1 Show the operator and evidence status

Calling every schema change a "migration" conceals which operation is meant. A migration view shall name:

$$(S, T, F, \Omega, I_{in}, I_{expected}, I_{observed}, e),$$

where S, T, F identify the schemas and mapping, Ω is the operator, the three I's distinguish input, expected, and observed instances, and e records evidence status.

For bounded Delta in the accepted profile, the view may compare expected and observed target-shaped data after a future API exists. For Sigma, Pi, general Kan extensions, physical database migration, SQL generation, transaction cutover, rollback, or information-loss execution, the current status is unsupported.

Table 4: Fields required in a migration presentation.

Field	Required presentation
Identity	source schema, target schema, mapping, operator, and versions
Scope	selected instance or corpus revision and affected object set
Expectation	independently specified expected result or declared absence
Observation	normalized computed result with supplying runtime revision
Risk	unsupported components, possible loss, failed obligations, and stale dependencies
Comparison	object-level additions, removals, changed values, and witnesses
Status	proposed, structural-only, validated, computed, disagreed, or failed
Provenance	activity, inputs, software identity, configuration, time, and review transition

6.2 Visualization must not simulate execution

An animation can make a proposed mapping look operational. To avoid that, unexecuted transitions should use static planned-state conventions and an explicit label. Sample data generated for explanation must remain distinguishable from repository data and computed output. If a future animation interpolates between before and after states, the intermediate frames are presentation, not database states.

Information loss also requires discipline. The accepted Paper 9 can inform questions and vocabulary, but the audited runtime has no accepted loss-analysis executor. The migration view can display a declared risk or an externally supplied assessment with provenance. It cannot present a computed loss certificate until a native gate establishes one.

Evidence status. PROVISIONAL CONTRACT for all application views. ACCEPTED UPSTREAM is limited to bounded Delta library and fixture computation, not UI or API execution..

7 Constraint violations as records

7.1 A red badge is not a witness

Validation often collapses to a Boolean or a toast. That is inadequate for a reference application. A violation must be a versioned record that explains what failed and points to the data needed to check the explanation.

Definition 7.1 (Violation record)**.** *A violation record is*

$$v = (k, \phi, c, x, w, a, s, \eta, u),$$

where k is a stable code, ϕ a phase, c the constraint identity, x the subject, w a witness or counterexample, a source anchors, s severity and lifecycle status, η supplying runtime and schema metadata, and u a human-readable explanation.

Phases include parsing, projection, schema typing, mapping validation, instance validation, overlay consistency, provenance completeness, storage, sync, and presentation accessibility. A repair suggestion is a separate overlay with its own author, status, and provenance.

Example 7.2 (Mapped equation failure)*.* Suppose a source schema declares $f; g = h$. A mapping sends $f; g$ to target path $p; q; r$ and h to s, but the accepted equality profile cannot establish $p; q; r = s$. The violation should record the source equation, both mapped paths, endpoint types, equality procedure and version, and the exact failure classification. "Mapping invalid" is not enough.

7.2 Lifecycle

A violation is observed against named revisions. It can be acknowledged, assigned, resolved, waived, superseded, or invalidated by a dependency change. Resolution requires a new validation activity; changing the UI state alone does not resolve the underlying constraint.

Invariant 7.3 (Witness fidelity)**.** *Selecting a violation shall expose a witness that independently rechecks the reported failure against the same normalized inputs. If no compact witness exists, the record shall say which procedure and logs must be rerun.*

8 Provenance and historical reconstruction

8.1 Provenance is not a tooltip

PROV-O distinguishes entities, activities, and agents and supplies relations for derivation, use, generation, association, and attribution [13]. Database provenance research further shows how annotations can explain the contribution of input tuples to query results [10]. MDReader does not need to claim complete semiring provenance in its first profile, but it must avoid using the word "provenance" for a bare file path.

A minimal result record includes:

- source and overlay revision identities;
- projection, mapping, query, or validation activity identity;
- software revision, configuration, parser profile, and model version;
- start and completion time;

- generated normalized result and diagnostics;
- responsible human, service, or model agent;
- review and authority transitions; and
- completeness limits and unavailable inputs.

Invariant 8.1 (Historical reconstruction)**.** *For a result labeled reproducible, the recorded inputs, versions, configuration, and deterministic seeds shall be sufficient to recompute the normalized result, or the record shall identify the external dependency that prevents recomputation.*

This requirement does not promise byte-identical browser rendering or deterministic output from an unpinned external model. It demands an honest account of what can and cannot be reconstructed.

8.2 Historical queries

A query result is meaningful only against a corpus and overlay revision. The query record should therefore bind:

$$q = (\text{text or typed request}, R, O, M, C, \theta),$$

where R is the source-revision set, O overlays, M mappings, C constraints, and θ retrieval or execution configuration.

Versioning systems such as Decibel and OrpheusDB study dataset branching and relational version management [11, 14]. Those systems are related design evidence, not MDReader components. The reference application can begin with immutable logical revisions in local SQLite, but it must make retention, compaction, branching, and deletion policy explicit.

Evidence status. ACCEPTED PAPER CONTRACT for provenance vocabulary; PROVISIONAL CONTRACT for the application; ABSENT/OPEN for a provenance executor and historical-query engine..

9 Semantic diff

9.1 Six layers of change

A text diff is necessary but not sufficient. MDReader should separate:

1. source-byte changes;
2. parse-tree and span changes;
3. schema and typed-object changes;
4. mapping assignment and obligation changes;
5. overlay and authority changes; and
6. provenance, constraint, and result invalidation.

Let two application snapshots be A_0 and A_1. A semantic diff is not a single set subtraction but a structured report

$$\mathcal{D}(A_0, A_1) = (D_s, D_p, D_\tau, D_m, D_o, D_v, D_r),$$

with source, projection, type, mapping, overlay, validation, and result components.

Identity matching is itself evidence. Exact stable IDs are strongest. Path, heading, content, and neighborhood heuristics may propose matches, but the report must identify the method and ambiguity. Schema-evolution research illustrates how frequently real systems change in nontrivial ways [8]; it does not solve Markdown identity automatically.

Protocol 9.1 (Seeded semantic-diff oracle)**.** *A future diff evaluation shall create a corpus pair with independently listed changes:*

1. *move a file without editing it;*
2. *rename a heading while retaining a stable source identity;*
3. *split one section into two;*
4. *change a claim without changing its evidence;*
5. *change a mapping path while preserving endpoints;*
6. *introduce and then repair a constraint violation;*
7. *approve one machine suggestion and reject another; and*
8. *change a parser or schema version without changing source bytes.*

The oracle shall specify expected matches, additions, removals, changes, invalidations, and uncertainty. Precision and recall shall be reported per layer, not only as a global score.

Evidence status. PROVISIONAL CONTRACT. The current accepted slice structurally compares schema-diff-shaped fixtures but does not execute semantic diff..

10 Local SQLite execution contract

10.1 Why a local relational store is a reasonable target

The source corpus should remain ordinary files. Derived structure still needs transactional identity, references, history, search, validation state, and jobs. SQLite is a plausible local reference store because it provides transactions, relational constraints, and an optional full-text index in one file [23, 25]. Plausible is not implemented.

Figure 2 separates source authority from derived state. The database may cache bytes for a revision, but it does not silently replace the file as authoring authority.

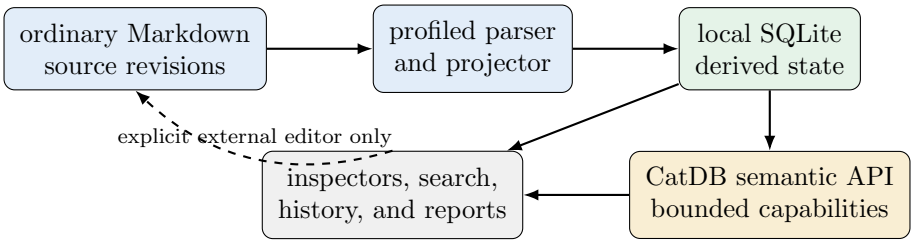

Figure 2: Proposed local architecture. The dashed path is not an implicit write-back feature. None of these application components exists in the audited repository.

10.2 Logical tables

The contract does not freeze a physical schema, but a conforming design needs equivalents of:

- workspace, source, source revision, and source span;
- projected object, attribute, relation, and parser diagnostic;
- overlay assertion, anchor, relation, status transition, and review;
- schema, mapping, assignment, path, equation, and validation;
- query, retrieval candidate, rank feature, result, and explanation;
- provenance entity, activity, agent, and edge;
- violation, witness, acknowledgement, and resolution;
- logical application revision and parent relation;
- background job, attempt, lease, cancellation, and diagnostic; and
- FTS document plus embedding metadata, if those capabilities are enabled.

SQLite foreign-key enforcement must be enabled for every connection because the documented default is disabled [22]. STRICT tables can reduce accidental type flexibility where the representation permits them [24]. Neither mechanism replaces application invariants, schema migration, or acceptance tests.

Protocol 10.1 (Atomic corpus revision)**.** *For a logical revision R_n, the local implementation shall:*

1. *observe and hash the intended source set;*
2. *begin a database transaction;*
3. *record projection, overlay reattachment, validation, index, and provenance updates under a non-visible candidate revision;*
4. *either commit the complete candidate and atomically advance the visible revision pointer, or roll it back;*
5. *retain diagnostics for a failed attempt outside the logical revision; and*
6. *recover after process termination without exposing a partially committed revision.*

Large indexing work may use smaller physical transactions, but the application still needs a logical completeness marker. A query must not unknowingly mix old projections with new overlays.

10.3 File changes and write-back

File observation should debounce unstable saves, identify renames separately from delete/create when justified, and recheck hashes before commit. Symlinks, ignored files, generated content, submodules, permission failures, and case-folding behavior require explicit policy.

This contract contains no automatic write-back. If a later application edits Markdown, it must preview the exact patch, ask for explicit authorization, check that the source revision has not changed, write atomically, and record provenance. Those requirements belong to a separate accepted capability.

Evidence status. PROVISIONAL CONTRACT. No SQLite dependency, schema, migration, or recovery test is present..

11 Optional PostgreSQL synchronization

11.1 Optional means the local application still works

A local reference application should remain useful on an airplane, in an isolated repository, or when a shared service is unavailable. PostgreSQL synchronization is therefore a negotiated extension, never a prerequisite for opening, indexing, inspecting, or querying a local corpus.

Logical replication can stream data changes, but PostgreSQL documentation states that schema definitions are not replicated and that schema mismatch can stop replication [17, 19]. Conflicts also require visible handling [18]. A checkbox labeled "sync" is not a protocol.

Protocol 11.1 (Synchronization capability)**.** *Before exchanging data, peers shall negotiate:*

1. *application, storage-schema, projection-schema, and overlay-schema versions;*
2. *workspace identity and authorization;*
3. *synchronization direction and table or object scope;*
4. *stable identity and deletion semantics;*
5. *conflict detection and resolution policy;*
6. *DDL migration order and compatibility window;*
7. *retry, deduplication, ordering, and checkpoint behavior;*
8. *encryption, credential storage, and audit policy; and*
9. *a completeness statement for the resulting local revision.*

An incompatible peer shall fail closed without corrupting local state.

11.2 Conflict examples

Two peers can approve different statuses for the same overlay, reattach an anchor to different spans, delete and edit the same assertion, or create different schema versions under one logical name. Last-write-wins may be appropriate for some cache metadata and dangerous for authority transitions. The policy should be typed by record class and should retain both inputs in the conflict record.

Tests must inject connection loss before, during, and after remote commit; repeat deliveries; out-of-order deliveries; local process termination; schema mismatch; authorization revocation; and remote constraint failure. After each test, local operation and provenance reconstruction must still work.

Evidence status. PROVISIONAL CONTRACT and optional. No PostgreSQL driver, sync implementation, protocol, or failure-injection result is present..

12 Retrieval that is not just vector search

12.1 Four distinguishable stages

Exact terms matter in technical corpora. A commit hash, theorem name, CLI flag, error code, or schema object may be poorly served by an embedding. Lexical ranking such as BM25 remains a useful baseline [20]. Approximate nearest-neighbor structures such as HNSW can efficiently produce

vector candidates [15]. Rank fusion can combine independent rankings [7]. Typed relationships can then expand from a claim to evidence, from a requirement to a decision, or from a mapping to affected paths.

The proposed retrieval path keeps these stages visible:

1. exact and lexical candidate generation;
2. embedding candidate generation with model, version, and source revision recorded;
3. typed relationship expansion under named schema and mapping versions;
4. filtering, reranking, context assembly, and explanation.

Every result should expose which stage contributed it and why it survived filtering. Scores from unlike systems need not share a probabilistic meaning.

Invariant 12.1 (Similarity has no semantic authority)**.** *Vector similarity shall not establish object identity, logical implication, truth, authorship, approval, provenance, constraint satisfaction, or authorization. It may propose candidates for a separately identified operation.*

12.2 The hybrid hypothesis is falsifiable

The context design rejects a vector-only product. This paper agrees for contractual reasons: vector retrieval cannot replace typed inspection and authority. It does not claim that hybrid ranking always improves relevance. A frozen ablation might show vector-only ranking equal or superior for a particular query set. That result should narrow the ranking hypothesis while leaving the source, mapping, provenance, and constraint contracts intact.

Protocol 12.2 (Retrieval ablation)**.** *Evaluate lexical-only, vector-only, typed-relationship-only, and hybrid configurations against the same corpus revisions and judgments. Report recall at fixed cutoffs, nDCG, MRR, latency distributions, index size, explanation accuracy, and provenance-link accuracy. Freeze candidate pools, model versions, seeds, filters, and tie-breaking. Disclose assessor protocol, missing judgments, query construction, and multiple-comparison treatment.*

Evidence status. PROVISIONAL CONTRACT. No FTS, embedding, relationship expansion, reranker, judgment set, or retrieval result is present..

13 Evaluation program

13.1 Correctness before preference

The first evaluation is a seeded corpus with independent expected results. Fixtures should include headings, nested blocks, tables, task lists, code, links, claims, evidence, decisions, requirements, tasks, architecture, mappings, violations, history, and deliberately malformed syntax.

Tests should call the public application boundary and compare results to the semantic library boundary. Reusing the same code to generate fixtures and evaluate them creates common-mode confidence, so the expected-results manifest must be independently reviewed.

13.2 Visualization validation

Munzner's nested model separates domain problem, data and operation abstraction, visual encoding and interaction, and algorithm validation [16]. MDReader should use the same discipline:

Table 5: Core end-to-end tasks and primary oracle.

Task	Oracle
Open source from every projected object	exact revision, byte span, and expected source bytes
Inspect a claim and supporting evidence	independent overlay and relation manifest
Trace a mapping assignment	normalized mapping plus typed path obligations
Open a constraint violation	seeded subject, constraint, witness, and source anchors
Compare two revisions	layer-specific semantic-diff manifest
Reproduce a historical query	pinned corpus, overlays, configuration, and normalized result
Interrupt indexing	visible revision remains complete; candidate rolls back or resumes under protocol
Disable networking	local corpus remains usable
Resolve an optional sync conflict	both inputs, policy, resolution, and audit record retained

- interview and task evidence checks whether mapping and provenance questions matter to intended users;
- fixture review checks whether document and categorical abstractions represent those questions;
- controlled tasks test whether graph, table, and path encodings support the intended operations; and
- profiling tests layout, indexing, diff, and interaction algorithms.

A fast force-directed graph does not validate the abstraction. A preferred color palette does not establish task effectiveness.

13.3 Usability protocol

ISO 9241-11 frames usability through effectiveness, efficiency, and satisfaction in a specified context of use [12]. A future study should preregister:

- participant population and relevant Markdown, database, and category theory experience;
- recruitment, compensation, sample size rationale, exclusions, and stopping rule;
- frozen tasks, datasets, training, counterbalancing, and baseline order;
- a plain file and text-search baseline, a conventional Markdown or vector-search baseline, and the conforming application;
- primary completion and critical-error endpoints;
- secondary time, navigation, confidence, and satisfaction endpoints;
- missing-data and multiple-comparison treatment; and
- debriefing, anonymization, and artifact release.

The System Usability Scale can provide a compact descriptive measure [1]. It cannot replace observed task completion or establish why a mapping was misunderstood.

Potential tasks include finding the evidence for a claim, deciding whether a requirement has an accepted verification, tracing a mapping path, explaining a violation witness, and identifying the semantic effects of a revision. The acceptance criterion should be fixed before data collection. A failed criterion is a result, not a reason to relabel the study exploratory after the fact.

13.4 Accessibility protocol

The target is WCAG 2.2 level AA [26]. WAI-ARIA patterns can guide specific widgets, but native HTML semantics are preferred where possible [27]. Required checks include:

- complete keyboard operation and a logical focus order;
- visible focus, skip links, landmarks, headings, and named controls;
- programmatic table headers and relationships;
- contrast, zoom, reflow, text spacing, and reduced motion;
- error identification linked to fields and source locations;
- announcements for asynchronous indexing and validation state;
- no status communicated only by color, position, or animation;
- a complete table or structured-text equivalent for every graph; and
- manual screen-reader paths through source, mapping, violation, provenance, and diff tasks.

Automated scanners catch only part of this contract. Results should name browser, operating system, assistive technology, version, zoom, viewport, and input mode. Passing a scanner is not a WCAG conformance claim.

13.5 Performance and recovery

Local measurements should report cold and warm startup, full and incremental index time, query latency distributions, memory high-water mark, database size, write amplification, and recovery time. Corpora should vary in file count, total bytes, section count, link density, overlay density, mapping size, and change locality. Hardware, storage medium, power mode, operating system, SQLite version, cache policy, and background load must be disclosed.

Synchronization evaluation adds offline duration, backlog, conflict density, round trips, retries, remote version, and network impairment. A local reference result must not be generalized to universal scale or production availability.

14 Acceptance matrix

Table 6 turns the design into staged gates. Each row can be accepted or rejected independently. An application release must publish the matrix with exact source and artifact hashes.

Table 6: Proposed native acceptance matrix.

Capability	Required artifact	Decisive gate	Now
Source observation	watcher and revision fixtures	byte-preservation, rename, partial-save, and failure tests	absent
Projection	pinned parser/profile and expected AST	exact object, hierarchy, order, and span comparison	absent
Overlays	schema, authority workflow, anchor fixtures	no authority laundering; stale-anchor detection	absent
Inspectors	normalized records and UI	every field, path, status, and source anchor reachable in text and graph paths	absent
Mapping	public API plus library comparison	total assignments, typed paths, obligations, and API equivalence	absent
Migration	supported-operator registry and fixtures	expected/observed separation and unsupported-state fidelity	absent
Violations	seeded constraints and witnesses	recall of all seeds and independent witness replay	absent
Provenance	entity/activity/agent records	historical reconstruction and declared completeness	absent
Semantic diff	paired corpora and layer oracles	per-layer precision and recall with ambiguity disclosed	absent
SQLite	migrations, schema, recovery harness	one transaction per logical revision; foreign-key enforcement; versioned upgrade and downgrade policy	absent
Retrieval	judgments and four configurations	frozen ablation with explanation and provenance accuracy	absent
Accessibility	browser UI and test record	automated plus manual keyboard and screen-reader tasks	absent
Usability	preregistration and anonymized data	fixed task criteria and reported failures	absent
PostgreSQL sync	negotiated protocol and fault harness	conflicts, retries, DDL mismatch, auth revocation, and local continuity	absent
Review	exact source range and native logs	independent review closes blockers and majors	absent

The program roadmap proposes commands for application unit tests, browser end-to-end tests, API contracts, library semantic equivalence, cancellation and atomicity, workspace authorization, and application/API comparison. At the audited revision, those commands are intentions rather than runnable gates.

15 Threats to validity and security

15.1 Semantic threats

Dialect ambiguity. CommonMark and GFM do not cover every repository extension. A parser can be correct for one profile and wrong for the corpus. Profiles and diagnostics must be stored with projections.

Identity drift. Paths, headings, and content hashes each fail as universal identity. Stable IDs and heuristic reattachment require explicit lifecycle rules and conflict presentation.

Extraction error. Claims, evidence, decisions, and requirements are interpretive types. Automatic classification can be useful as suggestion but can create false authority.

Shared implementation and oracle. If the UI, API, library, and fixtures derive from one normalization bug, they may agree incorrectly. Independent expected results and adversarial cases reduce this common mode.

Bounded category profile. The accepted runtime uses finite, executable structures. Nothing in this application contract proves general categorical constructions or extends the accepted equality profile.

15.2 Storage and synchronization threats

SQLite constraints do not help if foreign keys are not enabled on each connection. Transactions do not create a complete logical revision unless visibility is managed across batches. Crash recovery can retain an apparently valid but stale search index.

Optional synchronization adds authentication, authorization, confidentiality, integrity, availability, replay, deletion, and tenancy risks. Logical replication is not a substitute for application conflict policy. Repository content may include secrets or personal data; indexing and embeddings can increase exposure. A production system would require a separate threat model, data retention policy, secret scanning, encryption design, and security review. This paper provides none of those results.

15.3 Evaluation threats

Seeded tasks can favor the design. Small expert samples can conceal learnability problems. Familiarity can favor a text-search baseline or the new interface. Relevance judgments can be incomplete. Accessibility results can vary across browser and assistive-technology combinations. Performance can be dominated by corpus shape and cache state. The protocols therefore require frozen baselines, disclosed participant experience, independent oracles, raw measurements, and narrow conclusions.

16 Falsifiers

The following observations reject or narrow conformance. They are not a list of hypothetical bugs to dismiss after implementation.

1. A read-only observation, index, search, or inspection action changes source bytes.

2. A projected span fails to resolve to the asserted bytes and source revision.
3. Recovered syntax is presented as a lossless parse.
4. Stable document identity changes solely because the file moves.
5. A generated suggestion is displayed as reviewed or approved without an authority transition.
6. An inspector disagrees with the normalized record or the public API.
7. A mapping view hides an unmapped object, path image, or failed obligation.
8. A structurally decoded artifact is labeled validated or executed.
9. A seeded constraint violation is missing or exposes an invalid witness.
10. A seeded semantic change is absent from its required diff layer.
11. Provenance cannot identify the source revision, transform, software identity, configuration, and generated result.
12. Vector similarity establishes identity, truth, authority, proof, constraint satisfaction, or authorization in any application path.
13. Interrupted indexing exposes a partially committed logical revision.
14. Local corpus operation requires a PostgreSQL connection or network.
15. Synchronization silently loses, duplicates, or misattributes authoritative data.
16. A normalized API result differs from the accepted library result for the same request.
17. A graph-only task has no complete keyboard and screen-reader equivalent.
18. The workflow misses its preregistered primary usability criterion.
19. A reported result cannot be reconstructed from pinned code, corpus, configuration, and disclosed external dependencies.
20. An upstream schema or mapping contract changes while derived state remains labeled current without migration or invalidation.

A vector-only ranking win is a narrower falsifier. It rejects a hybrid superiority claim for that evaluation, but it does not make vector similarity a mapping, a provenance relation, or a constraint witness.

17 Exact nonclaims

At the audited revision, this paper does not claim:

- an implemented MDReader package, user interface, backend, server API, or end-to-end test;
- SQLite storage, FTS5 indexing, PostgreSQL synchronization, or conflict handling in CatDB;
- a working Markdown parser, file watcher, typed overlay engine, inspector, mapping explorer, migration view, provenance view, semantic-diff engine, or historical-query engine;
- hybrid retrieval superiority or vector-search inadequacy for every corpus;

- universal CommonMark, GFM, or repository-dialect compatibility;
- correct automatic extraction of claims, evidence, decisions, requirements, tasks, architecture, contradictions, or entailments;
- that embeddings encode truth, identity, authorship, authority, provenance, constraints, or permission;
- complete provenance, executable information-loss analysis, or executable semantic diff;
- Sigma, Pi, general Kan extension, arbitrary data migration, physical migration, SQL generation, transaction cutover, rollback, or zero data loss;
- production scale, latency, durability, availability, security, privacy, or multi-tenant isolation;
- usability, accessibility, WCAG conformance, responsive-browser support, or assistive-technology compatibility;
- a categorical theorem, formal verification, or a general Rust/Haskell equivalence theorem; or
- acceptance of Paper 18 or Paper 15, Slice 14, an application review, an accepted PDF, or a publication-ready result; the accepted Papers 3 and 4 do not establish application code.

18 Related work

Functorial data migration supplies the categorical background for deriving data movement from schema mappings [21]. This paper does not extend that theory. It asks what an application must show so that a user can inspect a mapping and tell which operations are actually supported.

Provenance semirings provide a rigorous account of how inputs contribute to query results [10]; PROV-O provides an interoperable vocabulary for entities, activities, and agents [13]. The proposed provenance profile is intentionally smaller than a claim of complete query provenance. It focuses first on source, transform, software, configuration, authority, and result reconstruction.

Decibel and OrpheusDB study versioned relational datasets [11, 14]. PRISM and related schema-evolution work motivate explicit change management [8]. MDReader differs in placing ordinary Markdown sources, parser projections, typed overlays, categorical mappings, and application evidence in one inspection contract. That difference is a design target, not a comparative performance result.

BM25, HNSW, and reciprocal-rank fusion represent distinct retrieval tools [7, 15, 20]. The proposed design keeps their signals separate and adds typed relationship expansion. It makes no novel ranking claim without the ablation in Definition 12.2.

Finally, visualization validation, ISO usability concepts, SUS, WCAG 2.2, and WAI-ARIA practices shape the evaluation contract [1, 12, 16, 26, 27]. Citing these standards does not confer usability or accessibility. Only an implemented, tested interface can supply that evidence.

19 Roadmap to a result-bearing paper

A result-bearing revision should proceed in dependency order.

1. Freeze a versioned source/projection/overlay contract and publish a small independent corpus oracle.
2. Add an application package and lockfile without expanding the accepted semantic claims.

3. Implement local SQLite migrations, connection policy, logical revisions, and failure recovery.
4. Expose the accepted semantic library through a versioned API and prove normalized library/API equivalence on accepted and adversarial fixtures.
5. Add source, object, path, mapping, violation, provenance, and diff inspectors with complete textual equivalents.
6. Implement retrieval stages separately, record explanations, and run the frozen ablation.
7. Run browser end-to-end, keyboard, screen-reader, reflow, and responsive checks.
8. Preregister and run the usability study.
9. Add PostgreSQL sync only after local correctness, then publish conflict and failure-injection evidence.
10. Obtain independent native review over the exact source range and resolve blockers before changing the manuscript status.

Each step should add evidence to the acceptance matrix. It should not retroactively reinterpret a previous absence as success.

20 Conclusion

A Markdown corpus can be easy to read and still be hard to reason about. Search retrieves passages. It does not, by itself, show whether an evidence item supports a claim, whether a decision satisfies a requirement, whether a schema mapping preserves an obligation, or which results a revision invalidates. Those questions need typed structure, visible mappings, witnesses, and provenance.

MDReader is specified here as the application that would force those semantics into contact with ordinary work. Source files remain authoritative. Projections and overlays are versioned derived data. Inspectors expose complete assignments and statuses. Migration views distinguish expected from observed results. Violations carry witnesses. Semantic diffs report change by layer. SQLite supplies a proposed transactional local store; PostgreSQL sync remains optional and protocol-bound. Lexical, vector, and typed retrieval signals stay separate, and similarity never becomes semantic authority.

The repository does not yet contain that application. It contains an accepted, bounded semantic core and a roadmap item for the server and UI. The correct conclusion is therefore neither that MDReader works nor that categorical schema modeling has been validated by a user interface. The result of this paper is a conservative acceptance target: observable contracts, exact nonclaims, and falsifiers against which a future implementation can fail clearly or earn a narrower, evidence-backed claim.

References

[1] John Brooke. Sus: A quick and dirty usability scale. In Patrick W. Jordan, Bruce Thomas, Ian L. McClelland, and Bernard Weerdmeester, editors, *Usability Evaluation in Industry*, pages 189–194. Taylor and Francis, 1996.

[2] CatDB Research Project. Immutable context export: Markdown viewer, search, and categorical data architecture. Repository artifact `context/01-markdown-viewer-and-search.md`, 2026. SHA-256 `89dbe785d84f2ec25819fb8d9b3ccc07fa6b45841c17e97f56de6c19978e11ea`.

[3] CatDB Research Project. Paper manifest. Repository artifact `research/paper-manifest.md`, 2026. SHA-256 `bc6cdb5fa6ca411181db8d4a42ecdba47d4542b2615052aaa6c1802f4eb8417a`.

[4] CatDB Research Project. Implementation roadmap. Repository artifact `research/implementation-roadmap.md`, 2026. SHA-256 `66f9c612e6d26790da05923110f7ba1339d1556efe3fa8a4ab65f7bfb75a40af`.

[5] CatDB Research Project. Slice 4 release record: Instances and bounded delta. Repository commit `056915834fcc2f172a4a72efe3629f4f753e45cf`, 2026. Release-record SHA-256 `afe1f08544ffb80b9dadd017a721371292599b863a5e06d92a231f2c90599254`.

[6] CommonMark Project. Commonmark specification, version 0.31.2. https://spec.commonmark.org/0.31.2/, 2024. Accessed 23 July 2026.

[7] Gordon V. Cormack, Charles L. A. Clarke, and Stefan Buettcher. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 758–759, 2009. doi: 10.1145/1571941.1572114.

[8] Carlo A. Curino, Hyun J. Moon, Letizia Tanca, and Carlo Zaniolo. Schema evolution in wikipedia: Toward a web information system benchmark. *Proceedings of the VLDB Endowment*, 1(2):1324–1335, 2008. doi: 10.14778/1453856.1453939.

[9] GitHub. Github flavored markdown spec. https://github.github.com/gfm/, 2026. Accessed 23 July 2026.

[10] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 31–40, 2007. doi: 10.1145/1265530.1265535.

[11] Silu Huang, Liqi Xu, Aaron J. Elmore, and Aditya Parameswaran. Orpheusdb: Bolt-on versioning for relational databases. *Proceedings of the VLDB Endowment*, 10(10):1130–1141, 2017. doi: 10.14778/3115404.3115417.

[12] International Organization for Standardization. Iso 9241-11:2018, ergonomics of human-system interaction, part 11: Usability: Definitions and concepts. https://www.iso.org/standard/63500.html, 2018.

[13] Timothy Lebo, Satya Sahoo, and Deborah McGuinness. Prov-o: The prov ontology. W3C Recommendation, https://www.w3.org/TR/prov-o/, 2013.

[14] Mark Maddox, David Goehring, Aaron J. Elmore, and Aditya Parameswaran. Decibel: The relational dataset branching system. *Proceedings of the VLDB Endowment*, 9(9):624–635, 2016. doi: 10.14778/2947618.2947619.

[15] Yu. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2020. doi: 10.1109/TPAMI.2018.2889473.

[16] Tamara Munzner. A nested model for visualization design and validation. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):921–928, 2009. doi: 10.1109/TVCG.2009.111.

[17] PostgreSQL Global Development Group. Postgresql 18: Logical replication. https://www.postgresql.org/docs/18/logical-replication.html, 2026. Accessed 23 July 2026.

[18] PostgreSQL Global Development Group. Postgresql 18: Logical replication conflicts. https://www.postgresql.org/docs/18/logical-replication-conflicts.html, 2026. Accessed 23 July 2026.

[19] PostgreSQL Global Development Group. Postgresql 18: Logical replication restrictions. https://www.postgresql.org/docs/18/logical-replication-restrictions.html, 2026. Accessed 23 July 2026.

[20] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009. doi: 10.1561/1500000019.

[21] David I. Spivak. Functorial data migration. *Information and Computation*, 217:31–51, 2012. doi: 10.1016/j.ic.2012.05.001.

[22] SQLite Project. Sqlite foreign key support. https://www.sqlite.org/foreignkeys.html, 2026. Accessed 23 July 2026.

[23] SQLite Project. Sqlite fts5 extension. https://www.sqlite.org/fts5.html, 2026. Accessed 23 July 2026.

[24] SQLite Project. Strict tables. https://www.sqlite.org/stricttables.html, 2026. Accessed 23 July 2026.

[25] SQLite Project. Transaction. https://www.sqlite.org/lang_transaction.html, 2026. Accessed 23 July 2026.

[26] World Wide Web Consortium. Web content accessibility guidelines (wcag) 2.2. W3C Recommendation, https://www.w3.org/TR/WCAG22/, 2024.

[27] World Wide Web Consortium. Wai-aria authoring practices guide. https://www.w3.org/WAI/ARIA/apg/, 2026. Accessed 23 July 2026.

A Repository audit commands

The following commands establish the application absence and current semantic boundary without modifying the worktree:

```
git show -s --format='%H %P %cI %s' HEAD

find apps/mdreader -mindepth 1 -print
test -f apps/mdreader/package.json
git ls-tree -r HEAD apps/mdreader
git log --all --oneline -- apps/mdreader

npm --prefix apps/mdreader test

cargo metadata --no-deps --format-version 1 |
  jq -r '.packages[].name'

cargo test --workspace --all-features

cargo run -p catdb-cli -- \
  fixtures compare fixtures/cases \
  --left rust \
  --right haskell \
  --capability-set slice-4
```

The observed application test command exits 254 with `ENOENT` for the `package.json` expected under `apps/mdreader/`. At the accepted Slice 4 release source, the closed-corpus comparison ends:

```
summary left=rust right=haskell capability_set=slice-4
total=33 semantic-compared=29 structural-compared=4 disagreements=0
```

At the correction-audit working tree, the same all-fixtures command ends:

```
summary left=rust right=haskell capability_set=slice-4
total=42 semantic-compared=29 structural-compared=13 disagreements=0
```

The extra nine structural comparisons come from later Slice 5 candidate fixtures. They do not revise the accepted Slice 4 result.

The complete commands, hashes, expected results, and interpretation limits are recorded in `reproducibility.md`.

B Minimum result manifest

A future result release should publish one machine-readable manifest with at least:

```
{
  "contract_version": "...",
  "source_commit": "...",
  "application_package_hash": "...",
  "lockfile_hash": "...",
  "catdb_runtime_commit": "...",
  "corpus_revision": "...",
  "parser_profile": "...",
  "projection_schema_version": "...",
  "overlay_schema_version": "...",
  "storage_schema_version": "...",
  "capabilities": {
    "projection": "accepted|rejected|absent",
    "mapping_inspector": "accepted|rejected|absent",
    "bounded_delta_view": "accepted|rejected|absent",
    "semantic_diff": "accepted|rejected|absent",
    "sqlite_local": "accepted|rejected|absent",
    "postgres_sync": "accepted|rejected|absent"
  },
  "native_gate_artifacts": ["..."],
  "accessibility_record": "...",
  "usability_preregistration": "...",
  "retrieval_ablation": "...",
  "review_record": "...",
  "known_failures": ["..."]
}
```

The manifest is illustrative, not a current repository schema. Its purpose is to make a release's evidence boundary mechanically visible.

C Author evidence snapshot

Table 7: Correction-audit evidence hashes.

Artifact	SHA-256
root `Cargo.toml`	`8d639b43fd8806b8b8a5341c86af626d95cc2bb68ed85d9074c1722caea40a34`
root `Cargo.lock`	`117abb87d3ec22ebfd2ce0f97f80f373b72043601b79464fb259de192c2b9643`
Slice 4 release record	`afe1f08544ffb80b9dadd017a721371292599b863a5e06d92a231f2c90599254`

Artifact	SHA-256
Slice 4 review record	`69664f8f17cbac5a84b36e819c942bb2a26aa10b8ff00c8cd94914543c6289af`
context file 01	`89dbe785d84f2ec25819fb8d9b3ccc07fa6b45841c17e97f56de6c19978e11ea`
paper manifest	`bc6cdb5fa6ca411181db8d4a42ecdba47d4542b2615052aaa6c1802f4eb8417a`
implementation roadmap	`66f9c612e6d26790da05923110f7ba1339d1556efe3fa8a4ab65f7bfb75a40af`
verification status	`3a988f61be0f757f54c0aacc2e7065faf5e7ef10d593a0b93aa922ed8156e648`

Final manuscript and PDF hashes belong in the separate reproducibility record after an isolated build and page inspection.

D Inspector field checklist

Table 8: Minimum fields per inspector.

Inspector	Identity and structure	Evidence and limitations
Source	workspace, stable source ID, path, revision, hash, parser profile, span	diagnostics, observation activity, unavailable bytes, identity method
Projected object	object ID, type, parent/order, attributes, source span	parser version, recovered state, old projection versions
Overlay	overlay ID/type, anchors, relations, status, schema version	author/generator, authority transition, stale-anchor state
Path	source object, arrow sequence, target object, attributes	endpoint typing, normalization procedure, unsupported equality
Mapping	source/target schemas, all assignments, equations	coverage, validation level, structural-only parts, runtime revision
Migration	schemas, mapping, operator, input, expected, observed	support profile, disagreements, possible loss, provenance
Violation	code, phase, constraint, subject, witness, anchors	severity, lifecycle, supplying runtime, repair suggestion authority
Provenance	entities, activities, agents, edges	completeness boundary, external dependencies, reproducibility status
Diff	compared snapshots, identity matches, layer changes	match method, ambiguity, invalidated results, unsupported semantics
Query	request, corpus/overlay revisions, filters, candidates, ranking	model/index versions, stage contributions, explanation, missing judgments