One Commit, Three Engines: Bounded, Verifiable Multi-Engine Transactions in a Single Process

Hyphae Research Foundation

Manuscript v0.1 — September 4, 2026 — record state per claim in Table 1 (class 3 throughout)

Every number traces to a receipt file in the HYPHAE repository at source commit `24bce1a`; the mapping is Appendix A.

Abstract

An application that needs relational tables, a key–value cache, lexical search and vector search today runs four systems, each holding its own version of the same facts, and reconciles them in application code with dual writes, sagas and background re-indexing. We describe HYPHAE, a local-first data engine in which one process owns one data directory and runs all four engines over one catalog, one write-ahead log, one multi-version snapshot space, one commit sequence number (CSN) and one recovery path. A transaction that touches SQL, a keyspace structure and a search document is one commit, not a coordinated sequence: it becomes visible at exactly one CSN in every engine, or in none. We make three claims and bound each with a receipt. (1) *Cost scales with what a transaction touches, not with what the directory holds*: on dedicated hardware the cross-engine transaction commits in 1.02 ms at the median against 46.6 ms for the materialized path, and its page reads (9) and log appends (3) are flat from 1 to 1,024 prior versions of the same row. (2) *The commit protocol is model-checked*: a TLA$^+$ specification of cross-engine commit and recovery was exhaustively checked by TLC (79,063,806 states generated, 43,885,299 distinct, depth 30) against six invariants, and the run was reproduced with identical counts. (3) *Results can be proved to a third party who trusts neither the machine nor the network*: eligible reads emit a canonical proof and witness that a verifier re-executes offline against an independently held 32-byte anchor. We report the engine's per-engine hot paths against SQLite, DuckDB, Redis and Tantivy on the same host and publish where it loses—SQLite's point read is 11× faster and Tantivy's durable ingest 41× faster—because the contribution is the integration, not per-engine speed. All receipts, harnesses and the formal model are public.

1 Introduction

The relational database stopped being the only store a long time ago [28]. A typical application now keeps its authoritative rows in one system, a denormalized copy in a key–value cache, a re-indexed copy in a search engine and an embedded copy in a vector store. Each system has its own notion of "current". Nothing outside application code knows that a row, a cached key, an index entry and an embedding describe the same object, so the application reconciles them by hand—dual writes, compensating sagas [10], change-data-capture pipelines, cache invalidation [14]. Multi-model databases [18] and polystores [8] attack the query side of this problem; the commit side—one durable decision that all engines observe at once—remains the application's problem.

HYPHAE takes the opposite position for the local-first case: one process, one data directory, and one commit sequence for every engine. The engines are not sidecars behind a facade; they share the catalog, the write-ahead log (WAL) [21], the multi-version root set, the commit scheduler and a stable object-identifier namespace. A transaction stages a SQL insert, a structure mutation and a search document, and commits once. Readers see all of it at one CSN or none of it.

This paper is not a claim that HYPHAE is faster than a specialized engine at that engine's own hot path. It is a claim about *integration with bounded cost and verifiable outcomes*, and it is deliberately measured in the way the project's publication policy requires: every number below is a receipt bound to an exact source commit and a named host class, and losses are printed with the same prominence as wins. Concretely we contribute:

- A description of the shared substrate (§2) and of the *delta all-engine transaction*, whose staging and commit resolve only the identities a transaction touches (§2.2).
- A TLA$^+$ model of cross-engine commit and recovery, checked exhaustively by TLC (§3).

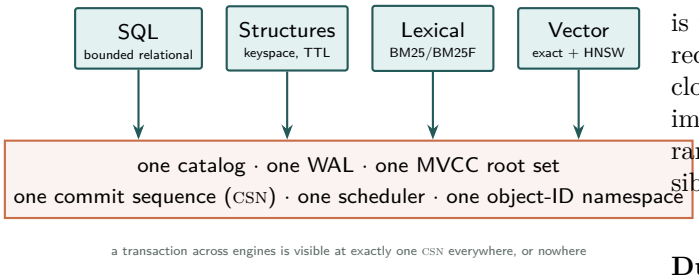

Figure 1: Four engines, one substrate. There is no per-engine commit path and no internal network hop between engines.

- Result proofs and witnesses that a third party verifies offline against an externally trusted anchor, under the rule that *self-consistency is not trust* (§2.3).
- A dedicated-hardware evaluation (§4): the cross-engine transaction against the materialized path; flat scaling in version depth; the durability ablation across strict, group and memory classes; honest per-engine comparisons; a document-scale ladder to one million documents with a bit-identical model-scorer oracle; and the crash, isolation and determinism evidence.
- A statement of what the system is not (§5), because a bounded engine that fails closed is only useful if the boundary is documented.

2 Design

2.1 One process, one directory

HYPHAE is a Rust engine that opens one data directory under an exclusive operating-system lock. Inside it live a bounded relational core, a native keyspace/structure engine (strings, counters, hashes, lists, sets, sorted sets, streams, TTL), and a search engine with lexical (BM25/BM25F [27]) and vector (exact and HNSW [19]) indexes. Figure 1 shows what they share. The shared parts are not an integration layer over independent stores; they are the only storage path any engine has.

Commit sequence. Every committed transaction is assigned one CSN. A snapshot is a CSN plus the set of engine roots published at it. Every read in every engine—a SQL query, a structure read, a search—executes against one snapshot and reports its `visible_csn`, so cross-engine correlation is a comparison of integers rather than a reconciliation.

Isolation. The engine provides snapshot isolation with first-committer-wins over logical write identities, one global commit sequence, and read-your-writes inside a transaction. A version $[begin\_csn, end\_csn)$ is visible when $begin\_csn \leq visible\_csn < end\_csn$; recovery validates the whole version chain and fails closed on discontinuities. Serializable execution is *not* implemented and must not be implied: predicate and range conflicts are not detected, so write skew is admissible exactly as in any snapshot-isolation system [25].

Durability classes. A commit selects *strict* (WAL append, page synchronization and log synchronization before acknowledgement), *group* (batched synchronization across a cohort of concurrent commits) or *memory* (acknowledged without any fsync). An acknowledged memory commit can be lost by a crash but never torn: recovery drops whole commits from the volatile WAL suffix only. The class changes when the acknowledgement is returned, not what is written; recovery replays the WAL to the last published root in every class.

Bounded grammar, fail-closed. The relational core admits a versioned grammar—typed DDL, multi-row `INSERT`, exact-primary-key `UPDATE/DELETE`, indexed `SELECT` shapes, total and grouped aggregates with `HAVING` and grouped `ORDER BY`, `DISTINCT`, `OFFSET`, `BETWEEN`, one indexed inner-join form, one materialized CTE form, two window forms. `LIMIT` is mandatory on scans. A plan that cannot bind to an index does not scan; it fails with a typed error. The same rule holds for search (candidate budgets, at most 250,000 documents per collection on the shipped bound) and for structures (bounded pages and scans). Boundaries are part of the contract, not tuning advice.

2.2 The delta all-engine transaction

The naive way to commit across engines inside one process is to materialize: load each engine's complete state for the touched containers, apply the mutations and publish. Its cost is proportional to what the directory holds. The delta all-engine transaction replaces this with point resolution:

1. *Stage.* The transaction captures an immutable snapshot and keeps three private overlays: relational, keyed by row; structure, keyed by key; lexical, keyed by (collection, document). Staging reads resolve only the exact identities the transaction names, by point reads against the snapshot's roots. No unrelated rows, keys or documents are read or copied.
2. *Commit.* The commit coordinator re-resolves the staged identities against the currently admitted roots, applies the deltas by copy-on-write page mutation, encodes one canonical WAL transaction, applies the selected durability policy, and publishes every changed engine root once, under one CSN.

The contract's invariant is that the number of page reads and log appends per commit depends on the touched identities and the tree height, not on the number of prior versions of those identities nor on the number of unrelated items in the engines. §4.2 measures exactly that.

The search manifest. Release 3.0.0 replaced the last component whose cost grew with the collection: the per-collection document manifest, previously one scalar value rewritten on every ingest (16 bytes per document, i.e. 16 MB per batch at one million documents), is now a header plus 16 kB chunks keyed by an immutable floor. Inserts write the header and the chunks they touch and never rename or delete a chunk (the delta path has no scalar delete); deletes merge adjacent chunks under an invariant that bounds the chunk count to $4 \cdot \text{docs}/1024 + 2$. Legacy manifests remain readable forever and upgrade on the first accepted mutation. This is what made the one-million-document rung measurable (§4.5).

2.3 Proofs and witnesses

An eligible read (catalog listings and descriptions, admitted SQL, product reads) can emit two artifacts: a canonical *proof* (`HYNPRF02`) binding the operation, its result and the snapshot roots, and a *witness* carrying the pages needed to re-execute it. A verifier with only the two files and a 32-byte *anchor* re-executes the operation under bounded reference semantics and compares the exact result. The anchor must come from an independently trusted channel: the design rule is that self-consistency is not trust—a directory that vouches for itself proves nothing, so the proof format carries an `ExternalTrustedAnchor` and the verifier refuses to run without one. This is the authenticated-data-structure lineage [20, 16, 32] applied to a local engine: no outsourced server, but the same asymmetry between the party that produced a result and the party asked to believe it.

3 Formal model

The cross-engine commit protocol—staging, admission against current roots, WAL encoding, root publication and recovery replay—is specified in TLA$^+$ [15] as `HyphaeCommit.tla` and checked with TLC [31]. The model covers concurrent transactions across the engines, crashes that truncate the volatile WAL suffix at an arbitrary prefix-closed point, file-wide fsync, and recovery; memory-durability commits acknowledge without entering the durable set, so the model demonstrates rather than hides that a crash may drop acknowledged memory commits while never splitting one. Six invariants are checked: `TypeOk`, `Atomicity`, `StrictDurability`, `FirstCommitterWins`, `VisiblePrefixComplete` and `CsnBounded`. The exhaustive run recorded in the 2026-08-30 dedicated-hardware receipt (TLC2 2.19, 96 workers, `-deadlock` because the model has intentional terminal states) generated 79,063,806 states, 43,885,299 of them distinct, to depth 30 in 1 min 41 s with no violation ("Model checking completed. No error has been found."); the 2026-09-03 receipt repeated the run with identical state counts (1 min 45 s) and recorded the specification digest (`b13cfc83…9728`), which makes the model-checking claim the one result in this paper in the *Reproduced* state. The model is evidence about the protocol as specified, not a proof of the Rust implementation; fidelity is carried by the all-engine transaction gate tests and the process-crash matrix of §4.6. Following industrial experience with formal methods at scale [23], we treat the specification as the authority for what "one commit" means.

4 Evaluation

4.1 Methodology and evidence classes

The project distinguishes three environment classes: class 1 (developer workstations; never quotable), class 2 (virtualized cloud hosts; quotable only as "virtualized, no latency certification") and class 3 (dedicated hardware). Comparative claims require class 3. Every receipt records host, kernel, toolchain, exact source commit, baseline versions, workload and raw harness output, and is committed to the repository under `docs/gates/evidence/`. Record state is *Measured* means one run per phase on one host of the class; *Reproduced* means the same quantity was repeated under the public protocol on another machine with the evidence retained. We label per claim, not per paper (Table 1). Three receipts on this host class exist eight days apart, each on a different physical machine; where a code path did not change between them, the three values are three independent runs and the claim is *Reproduced*. Where 3.0.0 changed the path, the final value is one run of the final code and the claim is *Measured*, with the earlier receipts as context only. Deltas under 10 % between receipts are not claimed in either state.

The dedicated host for the release measurement is an AWS `i7i.metal-24xl` (bare metal; Intel Xeon Platinum 8559C, 96 logical CPUs, 755 GiB RAM, six 3.75 TB local NVMe, ext4 `noatime`, `performance` governor, hypervisor flag absent; Ubuntu 24.04, kernel `7.0.0-1011-aws`, rustc 1.96.0; region us-east-2). Three receipts exist on this host class eight days apart: `8aeb6ea` (release 2.2.0, 2026-08-30), `2ff8a4b` (3.0.0 candidate, 2026-09-03) and `a443c52` (3.0.0 after the B$^+$tree split fix and the buffer-pool bound, 2026-09-

Table 1: Record state per claim. "Three receipts" = `8aeb6ea`, `2ff8a4b`, `a443c52`, three physical machines of the class.

claim	state
TLC model check: identical state graph in two receipts	Reproduced
Strict single-SET commit 1.81/1.82/1.82 ms (three receipts)	Reproduced
UPDATE with fsync per commit 1.75/1.85/1.82 ms	Reproduced
Reopen 7.7/7.7 s at 250k; 34.6/34.5 s at 1M	Reproduced
250k query ladder (bm25 6.2/6.2, facet 10.6/10.7, phrase 7.3/7.2, fuzzy 12.0/12.0 ms)	Reproduced
Baseline hot paths (SQLite, DuckDB, Redis, Tantivy) in all three receipts	Reproduced
Cross-engine delta transaction 1.02 ms; version-depth sweep	Measured (3.0.0 path)
Embedded GET 2.2 µs; SQL point read 20.3 µs; lexical query 111 µs	Measured (3.0.0 path)
1M query ladder 23.2–54.2 ms; buffer-pool comparison	Measured (3.0.0 path)
Scorer equivalence at 1M, bit-identical	Reproduced (two receipts, two layouts)
Crash matrices (class 2, 2026-08-02)	Measured, pre-3.0.0
Cross-host byte-identical directory (engine 1.2.2)	Reproduced (two hosts)

Table 2: Cross-engine transaction, 1 INSERT + 1 SET + 1 indexed document, memory durability, 2,000 commits, p50/p99. `a443c52 §4`, earlier receipts as context.

	`a443c52`	`2ff8a4b`	`8aeb6ea`
Materialized batch (ms)	46.6 / 95.8	65.0 / 127	55.1 / 101
Delta batch (ms)	**1.02 / 1.09**	1.10 / 1.16	1.13 / 1.26
SQL only	200 µs	218	205
SQL + structure	371 µs	386	444
SQL + structure + search	1.02 ms	1.07	1.13

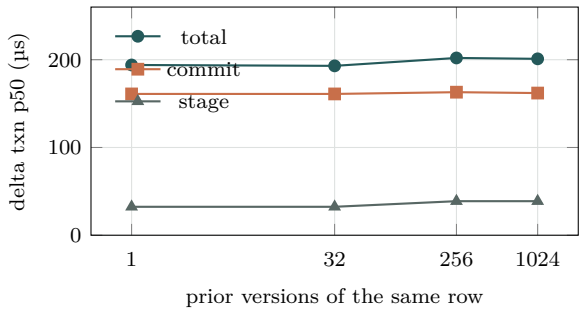

Figure 2: The cost of a cross-engine commit does not depend on how many versions of the touched row exist: 9 page reads and 3 appends at every depth. `a443c52 §5`.

03; the measured tree is commit `0295656` after the DCO rewrite). Where we quote one number it is from `a443c52` unless stated; the others appear as run-to-run context. Shipped defaults are used throughout: 8,192-frame buffer pool, one worker, no governor. Baselines are pinned: SQLite 3.50.2 (`rusqlite` 0.37 bundled, WAL, `synchronous=FULL`), DuckDB 1.5.5 (default durable WAL), Redis 7.0.15 over a Unix socket with `appendonly yes` in both `always` and `everysec` postures, and Tantivy; workloads are byte-identical deterministic corpora shared across engines, and durability postures are matched like for like where the baseline offers one.

4.2 The cross-engine transaction

Materialized versus delta. Table 2 reports the transaction shape 1 `INSERT` + 1 `SET` + 1 indexed document, memory durability, 2,000 commits, on the two paths. The delta path commits in 1.02 ms at the median (1.09 ms at p99) against 46.6 ms (95.8 ms) for the materialized path—a 46× ratio on this receipt, 49× on `8aeb6ea`. The composition rows show where the cost lives: SQL alone 200 µs, SQL plus structure 371 µs, all three engines 1.02 ms. Adding the search engine to a transaction costs about 650 µs on this host; it does not multiply the cost of the other engines.

Flat in version depth. Table 3 is the contract's invariant measured. With 1, 32, 256 or 1,024 prior versions of the same row the delta transaction costs 194–202 µs at the median, performs exactly 9 page reads and 3 WAL appends, and never loads complete state. Three orders of magnitude of history are invisible to the commit. With 0, 256 or 4,096 unrelated items per engine the cost rises from 346 µs to 893 µs: this is tree height (17→41 page reads), not a scan, and it is 15–16 % cheaper than on `2ff8a4b` because the even leaf split writes fewer pages per rewritten path (appends 15→11 and 20→14).

Group commit. With eight producers on the delta path, strict commits run at 1.54 ms p50 and 587/s; group durability at 3.14 ms p50 and 2,263/s, 3.86× the strict throughput at twice the latency (256 commits, standalone smoke).

4.3 Durability ablation and a published regression

Table 4 isolates the durability classes on identical single-`SET` commits (materialized path, 10,000 commits). Strict costs 4.39 ms at the median, of which the receipt clocks account for execution 1.58 ms, WAL append 49 µs, page sync 536 µs and WAL sync 589 µs; memory durability 3.86 ms; group with eight produc-

Table 3: Delta-transaction scaling sweeps, CPU 0, median of three runs. a443c52 §5.

prior versions	total	stage	commit	reads	appends	full loads
1	194 µs	32.4	161	9	3	0
32	193 µs	32.4	161	9	3	0
256	202 µs	38.8	163	9	3	0
1,024	201 µs	38.8	162	9	3	0
unrelated items	total	stage	commit	reads	appends	
0	346 µs	34.2	272	17	5	
256	676 µs	33.6	577	30	11	
4,096	893 µs	51.1	764	41	14	

Table 4: Durability classes, identical single-SET commits, materialized path, 10,000 commits, p50. a443c52 §4; 2ff8a4b §5.

class	a443c52	2ff8a4b	8aeb6ea
Strict	**4.39 ms**	**35.5 ms**	4.98 ms
Memory	**3.86 ms**	**34.2 ms**	3.62 ms
Group, 8 producers	**12.4 ms**	**40.4 ms**	16.1 ms

ers 12.4 ms per commit but 497 commits/s against 196/s strict.

The middle column is the regression this project published rather than hid. The `2ff8a4b` receipt showed the same commits at 35.5, 34.2 and 40.4 ms—seven times slower than the release before—while the receipt clocks were unchanged. The 31 ms lived outside every clock: a B$^+$tree batch-rewrite change had split overflowing leaves fill-first, producing trees of one-entry leaves (904 leaves for 4,000 keys where ≤ 21 are expected), and the complete-state load of the materialized path was paying for it. The fix splits evenly; the `a443c52` column is the same host class eight hours later. We include this because it is exactly what the receipt discipline is for: the number was wrong, the receipt said so, and the receipt of the fix is bound to the fixing commit.

4.4 Per-engine hot paths against specialized baselines

Table 5 places each engine next to the system built for that workload, on the same host, same run. Hyphae loses the SQL point read to SQLite by 11× (20.3 µs vs 1.8 µs), loses durable lexical ingest to Tantivy by 41× (1.12 s vs 27.0 ms per 1,000-document batch) and lexical query latency by 1.4×, and loses the embedded GET at p99 to Redis over a Unix socket (16.3 µs vs 9.5 µs) while winning it at p50 (2.2 µs vs 7.9 µs). Durable writes are the honest cost of copy-on-write publication plus two synchronizations: an `UPDATE` with fsync per commit is 1.75 ms against SQLite's 21.9 µs (SQLite's default journal posture does not synchronize the same way; the receipt records both postures).

The trend across the three receipts is what changed in 3.0.0: the prepared point read went from 34.0 µs to 20.3 µs (denser leaves, hot path resident in the larger pool), the embedded GET from 11.1 µs to 2.2 µs, and the lexical query from 4.09 ms in 2.2.0 to 111 µs—37×—with Tantivy's lead shrinking from 50× to 1.4×. None of these are claims of superiority over the baseline; they are the cost of running the workload inside one transactional process, and they are the cost the application would otherwise pay in reconciliation code.

4.5 Scale: the document ladder to one million

The search engine ships with a collection bound of 250,000 documents; the bound moves only when a receipt exists for the next rung. Table 6 shows the 250k and 1M rungs on the dedicated host, the latter with the bound lifted on that host only. After the leaf-split fix and the 8,192-frame pool, the 1M query ladder is linear in the document count—bm25 23.2 ms, filtered+facet 42.6 ms, phrase 24.2 ms, fuzzy(1) 54.2 ms, i.e. 3.3–4.5× the 250k stage for 4× the documents, where the previous receipt had shown 6.6–8.3×. At 250k nothing moved: the smaller pool already held a 250k query's segments. The manifest at 1M is 1,952 chunks behind a 39,056-byte header; a controlled comparison on the same 1M directory shows the pool bound alone accounts for 2.3× on the scorer (50.8 ms to 22.4 ms), and the CPU profile no longer shows page verification (BLAKE3 and CRC32C, 44 % of samples before) in the top 40.

Scorer equivalence. The durable posting scorer is checked against a retained-model reference scorer that recomputes the ranking from first principles. At one million documents, on a tree whose physical layout the split fix had just changed, the two agree on 1,000 ranked hits—identifiers, order and score bits—with the model taking 3.5 hours and the durable scorer

Table 5: Per-engine hot paths on the same host (p50 / p99 unless noted). Losses in colour. a443c52 §1--3.

workload	HYPHAE	baseline	
SQL point workload, 1M rows			
point SELECT, prepared	20.3 / 36.3 µs	SQLite 1.8 / 2.1 µs DuckDB 191 / 235 µs	**11× slower**
UPDATE, fsync/commit	1.75 / 2.43 ms	SQLite 21.9 / 402 µs DuckDB 915 µs / 1.38 ms	**slower**
UPDATE, batched ×100	30.6 / 31.4 ms	SQLite 724 µs / 4.34 ms	**slower**
load, 1,000-row batches	4.0 /s	SQLite 963 /s; DuckDB 13.1 /s	**slower**
Keyspace, 1M keys			
GET	2.2 / 16.3 µs	Redis UDS 7.9 / 10.8 µs	p50 3.6× faster, **p99 slower**
SET strict vs Redis `always`	1.81 / 2.53 ms	509 / 910 µs	**slower**
SET no-fsync-ack vs `everysec`	371 / 404 µs	9.6 / 11.5 µs	**slower**
Lexical, 100k documents			
query top-10	111 / 181 µs	Tantivy 78.3 / 82.9 µs	**1.4× slower**
ingest, 1,000-doc durable batch	1.12 / 1.73 s	Tantivy 27.0 / 45.7 ms	**41× slower**

Table 6: Document-cap ladder, fresh NVMe loads, ladders on reopened directories, p50 of 16 samples, limit 10. a443c52 §6.

rung	ingest/s	vacuum	reopen	bm25	filt.+facet	phrase	fuzzy(1)
250k	4,265	13.8 s	7.7 s	6.2 ms	10.6 ms	7.3 ms	12.0 ms
1M	3,638	64.3 s	34.6 s	23.2 ms	42.6 ms	24.2 ms	54.2 ms
ratio	0.85×		4.5×	3.7×	4.0×	3.3×	4.5×

39.5 ms cold and 22 ms warm. This is the fourth rung (100k, 250k, 1M twice) with bit-identical agreement. The first bm25 sample after a fresh load remains the cold-cache outlier (p95 17 s at 1M) and is reported, not excluded.

4.6 Crash, isolation and determinism

Process crash. The commit path has seven boundaries (blob staged, promoted, page appended, page synchronized, WAL appended, WAL synchronized, root published). A harness runs one child process per boundary and kills it with `SIGKILL` exactly there, then reopens the directory. The recovered CSN sequence is $[\bot, \bot, \bot, \bot, 1, 1, 1]$: through page synchronization the prior state reopens; from WAL append onward the complete new state reopens; no boundary exposes a mixed engine state. Four further boundaries cover checkpoint publication (eleven scenarios in total, all passed).

Block-layer power loss. The same eleven scenarios were replayed under `dm-log-writes` [17]: a fresh ext4 file system over the logging target, the process killed at each boundary, and the block log replayed only to the last *stable* write before mounting, running recovery and a read-only `e2fsck`. The recovered sequence differs from the process-kill case at exactly one boundary—$[\bot, \bot, \bot, \bot, \bot, 1, 1]$—because after "WAL appended" the page cache held a complete transaction that stable storage did not; the engine correctly reopens the prior state. We report this as the reason the two matrices both exist. Their status is `block-replay-not-physical-device-cut`: they are not evidence of a physical power cut, a device-cache loss or a tear inside a completed block write.

Both matrices are class 2 (`m6i.2xlarge`, EBS) and were recorded on 2026-08-02 against commits `91af0b7` and `0a167bc`, a month and the B$^+$tree layout change before the 3.0.0 source; no crash receipt is bound to the measured 3.0.0 commits. Process-kill lanes for group commit, WAL retention, vacuum, blob collection and migration remain open. This is the largest gap between what the model says and what the shipped code has been shown to do.

Isolation and metamorphic checks. Six isolation properties are proven by gate tests—repeatable reads from an immutable snapshot, private writes visible in-transaction, first-committer-wins on overlapping row writes, a rejected transaction cannot replace a committed value, disjoint stale write sets rebase and both commit, committed state survives reopen—and 768 metamorphic comparisons (256 deterministic cases × 3 rewrite families over a 64-row typed dataset, seed 20260804) return byte-equivalent rows for AND/OR commutativity, double negation, De Morgan, primary-key range ordering and CTE identity. Bounded TPC-C-shaped transactions (eight properties over nine ta-

ble families) and one admitted TPC-H shape (Q3; the other 21 are listed with their unsupported feature) round out the relational evidence. All of these are labelled "bounded implementation evidence, not promoted into the G2 closure": they do not claim phantom freedom, lost-update freedom, write-skew freedom or any official benchmark result.

Cross-host determinism. Two hosts with different distributions, kernels (7.0.12 vs 5.15.0), glibc (2.43 vs 2.35), CPUs (Xeon Platinum 8358 vs Xeon 6767P) and Python lines ingested the BEIR FiQA-2018 corpus (57,638 documents, 648 queries) through 7,208 windowed ingest/checkpoint/vacuum cycles. The committed directories are byte-identical: 55,562,927,163 bytes after ingest and 564,961,635 bytes after maintenance, with identical nDCG@10, recall@10 and MRR@10 to the digit. This receipt is at engine 1.2.2 and covers x86-64 only.

Scorer equivalence. See §4.5: four rungs (100k, 250k, 1M twice) on which the durable and reference scorers agree bit for bit.

5 What Hyphae is not

The engine's own claims page is the wording authority, and its non-claims are part of the contract. HYPHAE is not a universal SQL engine: no subqueries, no `UNION`, no outer joins, no expression arithmetic; unsupported shapes fail closed. It is not Redis-compatible: structures are Valkey-shaped in semantics but there is no RESP surface. It is not distributed: one process owns one directory, and multi-process access goes through the local daemon. It embeds no LLM, no embedding provider and no cloud service. The collection bound is 250,000 documents. The performance numbers are receipts for one host class and one run per phase; they are not guarantees, and deltas under 10 % between receipts are noise until a second host says otherwise.

6 Limitations and future work

Table 1 separates what has been reproduced from what has been measured once. The 3.0.0-specific paths—the delta transaction, the resident hot paths, the one-million rung—have one run of the final code; rerunning the same runbook on the published commit on a second machine of the class, and on a second CPU vendor, is the next receipt to produce. aarch64 is unmeasured. The vector branch is under-evaluated relative to the lexical branch: the roadmap's remaining conditions for the 1M bound are ANN consolidation cost and resident memory at $10^6 \times 768$ dimensions, which the delta path does not yet cover. The synthetic ladder corpus has a pathological one-term-per-document dictionary and supports cost claims only, never relevance claims. The crash matrices predate the 3.0.0 code and are class 2; binding them to the release commit on dedicated hardware is the next receipt to produce. The scorer's fail-open probe constructs errors on the hot path (5.5 % of samples); it is the cheapest next fix. A known 3.0.0 defect mislabels an all-rejected conditional structure batch as a corruption error (issue #268); it does not affect durability or the results here.

7 Related work

Multi-model and polystore systems. Multi-model databases store several models in one system [18]; polystores federate several systems behind one query interface [8]. Both address querying across models. HYPHAE is narrower and lower: it does not federate, and its contribution is that the *commit* is shared—one WAL transaction, one CSN—so the application never sees two engines disagree.

Sagas and coordinated writes. Sagas [10] and the dual-write and change-data-capture patterns [14] are what applications use when their engines cannot commit together. They trade atomicity for availability across processes. In the single-process local-first setting there is no availability to buy; HYPHAE removes the trade.

Transaction processing. WAL, ARIES-style recovery [21], MVCC [22, 25] and group commit [11, 5] are classical. Deterministic systems such as Calvin [29] obtain replayability by fixing the order of execution; HYPHAE fixes the order of publication (the CSN) and obtains cross-host bit-identical directories from a deterministic commit encoding (§4.6).

Embedded engines. SQLite [9] and DuckDB [26] are the embedded baselines we measure against, and each is faster at its own hot path. Redis/Valkey [1, 3] and Tantivy [2] are the keyspace and lexical baselines. Purpose-built vector systems [30] optimize a workload HYPHAE runs as one branch of a hybrid query.

Verifiable data. Authenticated data structures [20, 16] and verifiable SQL over outsourced databases [32] let a client check results from an untrusted server; ledger databases [4, 6] expose cryptographic history. HYPHAE's proofs apply the same asymmetry locally, with the explicit rule that the anchor must come from outside the directory.

Formal methods and testing. TLA$^+$ and TLC are used industrially to check distributed protocols [23]; we use them on a single-process multi-engine commit. Crash-consistency testing [24, 17] and isolation checking [7, 13] inform the crash matrices and litmus receipts in §4.6.

8 Conclusion

A local-first application does not need four systems to have four capabilities. It needs one commit that all four engines observe. Hyphae shows that this can be done with a cost that follows the transaction rather than the directory, a commit protocol whose meaning is fixed by a model-checked specification, and results that a third party can verify without trusting the machine. It also shows what that costs at the per-engine hot path, and prints the losses next to the wins. The receipts, harnesses, model and source are public at commit `24bce1a` [12].

References

[1] Redis. `https://redis.io`. Accessed September 2026.

[2] Tantivy: a full-text search engine library written in rust. `https://github.com/quickwit-oss/tantivy`. Accessed September 2026.

[3] Valkey: an open source, in-memory data store. `https://valkey.io`. Accessed September 2026.

[4] Amazon Web Services. Amazon quantum ledger database (QLDB) developer guide. `https://docs.aws.amazon.com/qldb/`. Accessed September 2026.

[5] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[6] Codenotary. immudb: immutable database. `https://github.com/codenotary/immudb`. Accessed September 2026.

[7] Natacha Crooks, Youer Pu, Lorenzo Alvisi, and Allen Clement. Seeing is believing: A client-centric specification of database isolation. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 73–82, 2017.

[8] Jennie Duggan, Aaron J. Elmore, Michael Stonebraker, Magdalena Balazinska, Bill Howe, Jeremy Kepner, Sam Madden, David Maier, Tim Mattson, and Stan Zdonik. The BigDAWG polystore system. *ACM SIGMOD Record*, 44(2):11–16, 2015.

[9] Kevin P. Gaffney, Martin Prammer, Larry Brasfield, D. Richard Hipp, Dan Kennedy, and Jignesh M. Patel. SQLite: Past, present, and future. *Proceedings of the VLDB Endowment*, 15(12):3535–3547, 2022.

[10] Hector Garcia-Molina and Kenneth Salem. Sagas. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, pages 249–259, 1987.

[11] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[12] Hyphae Research Foundation. Hyphae 3.0.0 source, contracts, and evidence receipts. `https://github.com/Hyphae-Research-Foundation/hyphae`, 2026. Release tag `release-v3.0.0-crates`, source commit `24bce1accdff8d14127797afe6f237a57c1cd4f3`.

[13] Kyle Kingsbury and Peter Alvaro. Elle: Inferring isolation anomalies from experimental observations. *Proceedings of the VLDB Endowment*, 14(3):268–280, 2020.

[14] Martin Kleppmann. *Designing Data-Intensive Applications*. O'Reilly Media, 2017.

[15] Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.

[16] Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. Dynamic authenticated index structures for outsourced databases. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pages 121–132, 2006.

[17] Linux kernel documentation. dm-log-writes: Linux device-mapper target for logging writes. `https://www.kernel.org/doc/html/latest/admin-guide/device-mapper/log-writes.html`. Accessed September 2026.

[18] Jiaheng Lu and Irena Holubová. Multi-model databases: A new journey to handle the variety of data. *ACM Computing Surveys*, 52(3):55:1–55:38, 2019.

[19] 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.

[20] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology — CRYPTO '87*, volume 293 of *LNCS*, pages 369–378, 1987.

[21] C. Mohan, Don Haderle, Bruce Lindsay, Hamid Pirahesh, and Peter Schwarz. ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM Transactions on Database Systems*, 17(1):94–162, 1992.

[22] Thomas Neumann, Tobias Mühlbauer, and Alfons Kemper. Fast serializable multi-version concurrency control for main-memory database systems. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 677–689, 2015.

[23] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. How Amazon Web Services uses formal methods. *Communications of the ACM*, 58(4):66–73, 2015.

[24] Thanumalayan Sankaranarayana Pillai, Vijay Chidambaram, Ramnatthan Alagappan, Samer Al-Kiswany, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. All file systems are not created equal: On the complexity of crafting crash-consistent applications. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 433–448, 2014.

[25] Dan R. K. Ports and Kevin Grittner. Serializable snapshot isolation in PostgreSQL. *Proceedings of the VLDB Endowment*, 5(12):1850–1861, 2012.

[26] Mark Raasveldt and Hannes Mühleisen. DuckDB: An embeddable analytical database. In *Proceedings of the 2019 International Conference on Management of Data (SIGMOD)*, pages 1981–1984, 2019.

[27] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.

[28] Michael Stonebraker and Uğur Çetintemel. "one size fits all": An idea whose time has come and gone. In *Proceedings of the 21st International Conference on Data Engineering (ICDE)*, pages 2–11, 2005.

[29] Alexander Thomson, Thaddeus Diamond, Shu-Chun Weng, Kun Ren, Philip Shao, and Daniel J. Abadi. Calvin: Fast distributed transactions for partitioned database systems. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 1–12, 2012.

[30] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, Ruiyi Jiang, Yi Wei, and Charles Xie. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD)*, pages 2614–2627, 2021.

[31] Yuan Yu, Panagiotis Manolios, and Leslie Lamport. Model checking TLA+ specifications. In *Correct Hardware Design and Verification Methods (CHARME)*, volume 1703 of *LNCS*, pages 54–66, 1999.

[32] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. IntegriDB: Verifiable SQL for outsourced databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1480–1491, 2015.

A Evidence ledger

Every quantity in this manuscript maps to a receipt file in the Hyphae repository (`docs/gates/evidence/`) at source commit `24bce1accdff8d14127797afe6f237a57c1cd4f3`, tag `release-v3.0.0-crates`. The complete ledger—each value with its file, section, measured commit, host class and the receipt's own caveats—is `evidence/ledger.md` in the manuscript repository. Table 7 lists the receipts by section.

Table 7: Receipts behind each section (repository paths under `docs/gates/evidence/` unless noted).

section	receipt(s), measured commit, class
§4.2, §4, §4.4, §4.5	`hyphae-3.0-metal-a443c52-2026-09-03.md` (`a443c52` = `0295656`, class 3); `baseline-i7i-metal-2026-09-03.md` (`2ff8a4b`, class 3); `baseline-i7i-metal-2026-08-30.md` (`8aeb6ea`, class 3)
delta contract	`docs/native/delta-all-engine-transaction-v1.md`; `native-delta-all-engine-transaction-linux-2026-08-03.md`
§3	`docs/formal/README.md`; TLC runs in the 2026-08-30 and 2026-09-03 metal receipts §1
§4.6 crash	`native-process-crash-matrix-linux-2026-08-02.md` (`91af0b7`, class 2); `native-checkpoint-process-crash-linux-2026-08-02.md`; `native-block-power-loss-replay-linux-2026-08-02.md` (`0a167bc`, class 2)
§4.6 isolation	`native-g2-isolation-litmus.md`, `native-g2-metamorphic.md`, `native-g2-sqllogictest.md`, `native-g2-tpcc.md`, `native-g2-tpch.md`
§4.6 determinism	`rag-cross-host-determinism-2026-08-22.md` (engine 1.2.2)
manifest, ladder	`collection-cap-250k-2026-09-02.md` (c-16, class 2); `collection-manifest-chunked-1m-ladder-2026-09-03.md`
proofs	`docs/native/native-proof-v1.md`, `docs/native/native-witness-v1.md`
release identity	`docs/release/receipts/3.0.0.md`
wording, classes, non-claims	`docs/product/claims.md`