

TGMS: An Agent-Native Bi-Temporal Graph Management System

Validated Temporal Operators and Trace-Grounded Answer Checking

Xiaofei Zhang
University of Memphis
xiaofei.zhang@memphis.edu

July 2026 · System Description Preprint (v3) · <https://github.com/zxf-work/tgms>*

Abstract

Temporal graph question answering requires exact composition over time and, when records are corrected, reconstruction of prior belief states. Large language model (LLM) agents are unreliable at identifier grounding, arithmetic, and interpreting partial query results. We present TGMS, an agent-native bi-temporal graph system that assigns the model two bounded roles: constructing a plan over a fixed operator interface and verbalizing the result. A static verifier checks plan structure, identifier grounding, declared output fields, temporal arguments, and estimated cost. Thirteen deterministic temporal operators execute over valid-time and transaction-time data and produce content-addressed traces with completeness metadata. A claim verifier checks the final answer against those traces.

On a frozen 94-task CollegeMsg test split run with three seeds, TGMS using Qwen2.5-14B-Instruct-AWQ obtains 0.408 normalized typed-answer accuracy, compared with 0.106 for vector-RAG, 0.064 for static-graph RAG, and 0.152 for text-to-Cypher. On correction probes, TGMS obtains 0.897; the two latest-state baselines obtain zero, while vector-RAG obtains 0.154 by answering current-belief cases. The main result transfers to a second communication corpus and a synthetic temporal-pattern suite, although complex multi-operator pattern plans remain beyond the 14B planner. Before gating, 21 of 220 emitted answers (9.5%) contain at least one unsupported gated claim; with gating, 0 of 199 emitted answers do, at a 1.0-percentage-point reduction in overall answer accuracy. Mutation experiments show that output contracts and evidence-completeness tracking catch errors that value-only checking misses. In a separate full-precision serving configuration, accuracy rises markedly with model scale (0.138/0.340/0.628 at 7B/14B/32B) while the tested baselines change little, and increasing vector-RAG’s retrieval breadth to a long-context configuration does not improve it. These results suggest that agent-facing database interfaces should make belief time, result contracts, and evidence completeness explicit.

1 Introduction

Temporal graphs arise in communication logs, transaction records, and evolving knowledge bases. They create two difficulties for LLM agents that are less visible in static text collections.

The first difficulty is temporal composition. Consider the question: “Among accounts reachable from X in February, how many cyclic triangles closed within one day?” Answering it requires time-respecting reachability [27], followed by δ -temporal motif counting [17]. A retrieval pipeline that serializes edges into text may omit the exact structure needed by the second step.

*This preprint accompanies TGMS v0.3.0: development results, the frozen-test campaign (thresholds recorded in the repository’s dated decision log before measurement), and post-campaign scale, fair-baseline, and portability studies.

The second difficulty is belief revision. A graph may change because the world changed, or because an earlier record was wrong. These cases are different. Answering “what did we believe on March 1?” requires both valid time and transaction time [24]. A latest-state snapshot cannot reconstruct this distinction after a correction has been applied.

LLMs introduce a separate set of risks. They may perform arithmetic incorrectly, invent identifiers, or report values that do not appear in the evidence.

These problems share one cause: conventional database interfaces assume a competent client, whereas an LLM is an approximate planner and reporter. TGMS therefore places the LLM outside the trusted computing boundary. The model chooses and composes operations and verbalizes their results, while the database owns temporal semantics, identifier grounding, arithmetic, bounded execution, provenance, and claim checking. We call a database interface *agent-native* when it is designed for a fallible machine planner: operations have machine-readable input and output contracts, explicit bounds and costs, deterministic result identities, and evidence metadata that supports automated checking.

This paper makes four contributions, each subordinate to that thesis:

- **Belief-state-preserving storage:** a bi-temporal property graph substrate with explicit assertion, retraction, and correction, backed by an append-only event log that replays to byte-identical state across storage backends.
- **A machine-checkable computation interface:** a fixed algebra of thirteen temporal operators with typed input *and* output contracts, deterministic results, pagination, and pre-execution cost guards.
- **Evidence-aware execution and answer verification:** a static plan verifier, a deterministic executor that produces content-addressed traces with completeness metadata, and a claim verifier that gates final answers against those traces.
- **Empirical evidence about when and why this separation helps:** frozen test suites over two real communication networks and one synthetic temporal-pattern workload, measuring answer accuracy, correction handling, plan validity, unsupported claims, verifier fault detection, model-scale behavior, baseline sensitivity to context limits, and operator latency.

Evaluation status and chronology. We first used a 22-task development split to finalize the operator interface, prompts, repair protocol, and acceptance criteria; the acceptance thresholds were recorded in the repository’s dated decision log and committed evaluation driver before measurement (an internal dated record, not an external registry). We then froze the evaluation protocol and ran the reported test campaign on 290 unique tasks, with three seeds for CollegeMsg. Post-campaign studies examine model scale, quantization, longer-context retrieval, and cross-family portability; these are reported separately from the frozen primary evaluation (Section 5.5).

1.1 System overview and trust boundary

Figure 1 shows the architecture as a trust boundary. The LLM appears exactly twice — as planner and as reporter — and both roles sit outside the boundary: nothing the model produces is executed or emitted unchecked. Inside the boundary, deterministic components own everything that must be true: the static verifier admits or rejects plans; the operator engine computes over the bi-temporal store; the executor records content-addressed evidence; and the claim verifier checks the final answer against that evidence. The remainder of the paper follows this decomposition: the

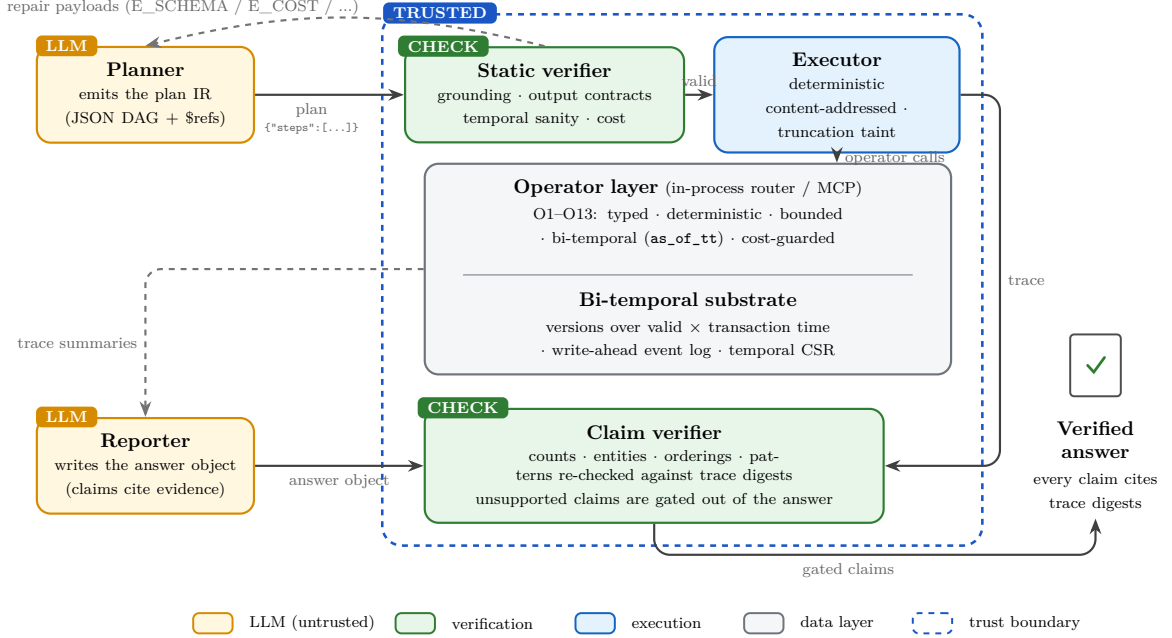


Figure 1: TGMS as a trust boundary. The planner and reporter (the LLM) are outside the trusted region; the static verifier, operator engine, bi-temporal store, execution trace, and claim verifier are inside it. Model outputs cross the boundary only through checks.

substrate (Section 2), the operator interface (Section 3), and planning, execution, and verification (Section 4).

2 Bi-temporal substrate

Each logical node and edge has a stable identity and one or more versions. A version carries a half-open valid-time interval $[vt_s, vt_e)$ and a half-open transaction-time interval $[tt_s, tt_e)$. TGMS stores time as int64 epoch microseconds and uses 2^{62} as the open-ended upper bound. A snapshot $G(t, tt)$ contains the versions whose valid-time intervals contain t and whose transaction-time intervals contain tt .

The write API provides three bi-temporal operations. **assert** records a new belief and carves its valid-time interval out of any overlapping prior belief of the *same logical identity*; independently coexisting facts use distinct identities, so within one identity the believed valid-time intervals are always pairwise disjoint. **retract** closes a fact’s valid time (the world changed; the earlier belief remains correct for its interval). **correct** closes the *transaction time* of an erroneous version and records its replacement (we were wrong; the record of the error is preserved). TGMS also provides a bulk-ingest path for instantaneous event streams.

A minimal worked example. At transaction time tt_1 we assert that edge e has weight $p=1$ over valid time $[0, 100)$: one version, $vt=[0, 100)$, $tt=[tt_1, \infty)$. At tt_2 we *correct* the weight to $p=2$ for $vt=[40, 60)$: the original version’s transaction interval closes at tt_2 , and three new versions open — $p=1$ over $[0, 40)$, $p=2$ over $[40, 60)$, and $p=1$ over $[60, 100)$, all with $tt=[tt_2, \infty)$. The current snapshot at $vt=50$ returns $p=2$; the same query pinned to $as_of_tt=tt_1$ returns $p=1$ — the belief before the correction — and continues to do so forever.

All writes pass through an append-only write-ahead event log. The log records every *attempted*

batch together with its deterministic outcome: only successfully committed batches contribute to the materialized graph state, and replay reproduces both the successful state transitions and the recorded failures, skipping the latter exactly as the original writer did. In this sense the log is the source of truth for state and audit trail for attempts. Replaying it into a backend reproduces the same store digest. We test this property across Kùzu [6] and DuckDB [18] adapters. Update semantics are implemented once, above a small adapter interface, so the backends share the same behavior. A hybrid logical clock provides strictly increasing transaction times. Failed batches remain in the log. Replay reproduces the failure and skips the batch in the same way. Version rows also reserve `source` and `provenance_ref` fields so that future agent write-back will not require a schema migration.

Property-based tests generate random sequences of assertions, retractions, and corrections. They check two main invariants. First, the versions believed for one identity have pairwise-disjoint valid-time intervals at every historical transaction time. Second, results pinned to a past `as_of_tt` remain byte-identical after later corrections. We call the second property *bi-temporal immutability*. Enforcing it required two details. Result digests must exclude metadata that describes the current belief state. Returned versions must also hide belief-closure timestamps that occur after the requested transaction time.

3 A fixed operator algebra

TGMS exposes the thirteen operators in Table 1. The interface is *fixed*: plans may invoke only operators from the registered set; we do not claim algebraic completeness. The set covers four workload families — historical state (O1–O3), temporal traversal (O4–O5), temporal patterns (O6–O9), and evolution analysis (O10–O11) — plus entity grounding (O12) and deterministic post-processing (O13). It is not intended to be complete; the evaluation measures which task families are expressible and which still exceed the planner or the interface. The `compute` operator keeps arithmetic out of the LLM, and `resolve_entities` is the only operator that may introduce an identifier not already present in the task.

Five rules apply to every operator.

Typed contracts. Inputs and outputs are validated against JSON Schemas generated from one registry. The same registry generates the tool definitions shown to the agent.

Deterministic results. The same store state and arguments produce the same canonical output and the same SHA-256 `result_digest`. Floating-point values are canonicalized before both serialization and threshold comparisons. This keeps the engine and its test oracle consistent.

Bounded execution. Two distinct bounds apply. Result *cardinality* is bounded through `limit/cursor` pagination. Computational *work* is bounded separately through pre-execution cost estimation and refusal — a motif count can return one number after an expensive search, so pagination alone would not bound it. The rejection includes concrete ways to narrow the request, which the planner may use during repair.

Bi-temporal semantics. Every operator accepts `as_of_tt`. A plan can therefore query a past belief state without changing the operator interface.

Table 1: TGMS operator algebra. Each operator also accepts `as_of_tt`, `limit`, and `cursor`, and declares its output fields.

	Operator	Semantics
O1	<code>entity_history</code>	Returns the version history of a node under a selected belief state.
O2	<code>snapshot_subgraph</code>	Returns a k -hop neighborhood, with $k \leq 3$, from snapshot $G(t, tt)$.
O3	<code>diff_snapshots</code>	Returns additions, removals, and changes between $G(t_1)$ and $G(t_2)$.
O4	<code>temporal_reachability</code>	Computes earliest arrival over time-respecting paths, with exact handling of wait limits.
O5	<code>temporal_paths</code>	Returns up to $k \leq 20$ node-simple, time-respecting paths.
O6/O7	<code>count/find_temporal_motifs</code>	Counts or returns exact δ -temporal motifs [17], ordered by (t, eid) .
O8	<code>graph_metric_timeseries</code>	Returns bucketed node, edge, degree, and reciprocity statistics.
O9	<code>burst_detection</code>	Detects unusual buckets with a rolling z-score or trailing-median rule.
O10	<code>neighborhood_evolution</code>	Returns neighbors gained or lost and a degree time series.
O11	<code>co_active</code>	Applies an Allen-relation interval join to two edge selections.
O12	<code>resolve_entities</code>	Resolves names or user identifiers to graph entities.
O13	<code>compute</code>	Applies count, sum, min, max, top- k , filter, or interval-relation operations to prior outputs.

Declared outputs. Each operator lists its output fields in the registry. The static verifier rejects references to fields that do not exist. Section 6 shows why this check matters for small models.

Correctness testing is a release gate. Each operator is compared with an independent brute-force implementation on 500 randomized combinations of stores and arguments. We also test metamorphic properties, including diff composition and bi-temporal immutability. These tests exposed problems in both the code and the specification. One test found a collision in the original version identifier. Another showed that a greedy reachability rule under a maximum-wait constraint depended on processing order. TGMS now uses an exact multi-label search over (node, arrival) states.

4 Planner–Executor–Verifier

A TGMS plan is a small JSON DAG. Each step calls one operator. References between steps use a limited `$ref` language with dotted fields, `rows[i]`, and `rows[*].field` projections. The language does not allow general expressions. An `answer_spec` identifies the step and field that contain the final result.

Before execution, a static verifier checks the plan schema, acyclicity, reference scope, temporal arguments, output fields, and estimated cost. It also applies a grounding rule. A literal identifier is allowed only if it appears in the task input. Other identifiers must be produced by a prior step and passed through `$ref`. This prevents plans from introducing identifiers that were neither supplied nor resolved. Rejections return structured violations. The planner may revise a plan up to three

times. We report both first-attempt validity and success after repair.

The executor runs the DAG deterministically and records content-addressed results. Re-executing the same plan over the same store state reproduces the same result digests. The executor also propagates *truncation taint*. If a step reads a truncated result page, that step and all dependent steps are marked as using incomplete evidence.

A reporter LLM produces a typed answer object. Each claim cites one or more evidence steps. The claim verifier checks counts and values against named trace fields. It checks entity mentions against the cited content, re-evaluates Allen relations for ordering claims, and re-executes operators for temporal pattern claims. A claim that cites truncated or tainted evidence can be rated no higher than *weakly supported*. Count, value, entity, and ordering claims are currently gated before the answer is emitted. Pattern checks are reported but are not yet used to block output.

4.1 Extensions outside the evaluated pipeline

Two components ship with TGMS but are not exercised by the primary evaluation; we summarize them for completeness. First, a *threat-model* defense: stored graph content is treated as data rather than instructions — strings inserted into prompts are escaped, length-limited, and placed inside explicit data fences, and red-team fixtures for this boundary run in continuous integration. Second, an *evolution memory*: operator-computed facts over weekly windows are summarized by an LLM, stored only when every numeric statement matches the computed values, and stamped with their creation transaction time. A later correction quarantines any note whose valid-time window overlaps the corrected interval. Quarantined notes are excluded from planner and reporter context and cannot be cited; final claims must still resolve to operator traces — the memory supplies hints, never evidence.

5 Evaluation

5.1 Questions and experimental design

The evaluation tests four hypotheses: **H1** transaction-time storage enables correction queries that are unavailable from latest-state inputs; **H2** operator output contracts improve plan validity and execution success; **H3** completeness-aware verification reduces unsupported output; **H4** the interface remains usable across data scales and model configurations. We first used a 22-task development split to finalize the interface, prompts, repair protocol, and acceptance thresholds; the protocol was then frozen (task generator, splits, and split SHAs recorded in the repository’s dated decision log) before any test-split measurement, and task generation was not altered afterwards. All systems use temperature 0, the same model checkpoint, the same seeds, the same repair budget, and the same typed answer format.

5.2 Data, frozen suites, systems, and metrics

Task construction. Tasks are program-generated with engine-computed gold: oracle plans are executed by TGMS itself, so no LLM-generated label is ground truth. The generator defines 17 semantic templates (10 single-operator temporal questions, T1; 3 evolution questions, T3; 4 multi-step analytical questions, T4), each with three hand-written paraphrases; every instantiated task samples template parameters (entities, windows, thresholds) from the store and uses one paraphrase. Instances whose oracle execution fails or degenerates are discarded. Correction probes are constructed by injecting a correction and generating a *pair* of separately scored questions

Table 2: Frozen suites. CollegeMsg test answer types: 42 count, 34 value, 14 interval, 4 entity-set. The CollegeMsg test split contains 13 probe *questions*: 7 before-correction halves and 6 current-belief halves.

suite	dev	test	T1	T2	T3	T4 / probes
CollegeMsg	22	94	48	—	20	13 / 13
email-EU	22	94	48	—	20	13 / 13
synthetic	24	102	48	8	20	13 / 13

Table 3: What each system can see. “valid time” for vector-RAG means timestamps serialized into chunk text; no baseline retains transaction-time history.

system	input representation	valid t.	trans. hist.	exact ops	bounded
TGMS	bi-temporal versions	yes	yes	yes	yes
Vector-RAG	retrieved chunks of serialized event sentences	in text	no	no	yes
Static-graph RAG	2-hop edge lists, latest snapshot	current only	no	no	yes
Text-to-Cypher	latest-state property graph, timestamp property	as property	no	query engine	yes

over the same logical fact — one asking for the belief before the correction, one for the current belief — keeping only pairs whose two gold answers differ. The synthetic suite adds planted-pattern mining tasks (T2) whose gold comes from the generator’s manifest. Tasks are split 20/80 (development/test), stratified by family; because the split cuts across probe pairs, a split may contain one or both halves of a pair.

Because the operator set and the workload were co-designed, the study does not establish coverage of independently collected temporal-graph questions; the frozen protocol limits tuning to the test sets but not this closed-world alignment (Section 8).

Systems and the information each receives. Table 3 makes the input representations explicit, because the correction probes are a *capability* comparison — they demonstrate the value of retaining transaction-time history — while the non-probe tasks compare the reliability of different planning and data-access interfaces over comparable information.

We also run a no-verifier TGMS ablation whose raw reporter answers are scored by the claim verifier acting as a measurement instrument without gating.

Metric: normalized typed-answer accuracy. Scoring compares the typed final answer, not prose. Count and value answers must agree within 10^{-9} after float canonicalization; entity-set answers require set equality of identifiers; interval answers count as correct when interval intersection-over-union is at least 0.5; remaining kinds use canonical-JSON equality. Because of the interval rule we call the pooled measure *normalized typed-answer accuracy* rather than exact match; counts and values are exact. One known laxity: entity identifiers are collected from `uid/src/dst` fields at any nesting depth, which can accept an answer with the right identifiers in the wrong structure. Gold answers are computed by the engine under the same normalization.

Table 4: Frozen-test campaign. Model: Qwen2.5-14B-Instruct-AWQ (AWQ 4-bit), vLLM 0.11, 24 GB Turing GPU; CollegeMsg pools three seeds (per-seed spread one task), other suites one seed. Paired bootstrap over CollegeMsg tasks (10k resamples): TGMS – vector-RAG +0.30 [+0.21, +0.40]; – static-graph RAG +0.34 [+0.25, +0.44]; – text-to-Cypher +0.26 [+0.15, +0.36].

	TGMS	Vector-RAG	Static-graph	Text-to-Cypher
CollegeMsg (94×3)	0.408	0.106	0.064	0.152
email-EU (94)	0.309	—	0.053	0.106
synthetic (102)	0.314	—	0.029	0.157
probes, CollegeMsg (13×3)	0.897	0.154	0.000	0.000
probes, email-EU (13)	0.846	—	0.000	0.000

Unsupported-claim rate. An answer is *unsupported* if at least one emitted gated claim (count, value, entity, or ordering; pattern claims are checked but not yet gated) is rejected by the claim verifier. We report the answer-level rate with explicit denominators, together with *coverage* (fraction of tasks for which an answer with at least one gated claim is emitted) and *conditional accuracy* (accuracy among emitted answers), because a zero unsupported rate can arise either from correct supported answers or from abstention.

5.3 Primary frozen-test results

Table 4 reports the campaign: frozen splits, three seeds for CollegeMsg, Qwen2.5-14B-Instruct-AWQ served by vLLM 0.11 on a single 24 GB Turing GPU.

Correction probes (H1). TGMS answers 0.897 of probe questions on CollegeMsg and 0.846 on email-EU. The two latest-state baselines score exactly zero; vector-RAG’s 0.154 comes entirely from current-belief halves, whose answers a latest-state input can contain. The before-correction halves require the earlier belief state, which no baseline input preserves — a representational distinction, not a planning-quality one (Table 3).

Answer support, coverage, and the cost of gating (H3). An answer is *emitted* when it carries at least one claim. On CollegeMsg, the ungated system (the no-verifier ablation) emitted 220 answers of 282 task runs, of which 21 (9.5%) contained at least one unsupported gated claim. With gating, 0 of 199 emitted answers did. (Counting instead over the 270 task runs that produced any measurable report, the ungated rate is 7.8%.) Gating cost 1.0 percentage point of overall accuracy (0.418 ungated vs. 0.408 gated). The gated system’s coverage was 199/282 (0.706) with conditional accuracy 0.548 among emitted answers — so part of the zero unsupported rate is bought by abstention, and we report both sides of that trade. These guarantees extend only to gated claim types, not to pattern claims, approximate operators, write-back, or adversarial tool behavior.

An honest zero. On the eight synthetic planted-pattern tasks (T2) the 14B planner never produced a valid multi-operator mining plan: coverage on that family was zero, so accuracy was zero while no unsupported claim was emitted. Abstention is the designed failure mode, but it is a coverage failure, not a success.

Replication of the development result. The frozen CollegeMsg result, 0.408, closely matches the 0.409 development-split result obtained during protocol design (22 tasks, same configuration),

Table 5: Verifier mutation classes. “Detected” means the mutated claim no longer verifies as fully supported. The final column disables the mechanism under test where one exists.

mutation class	cases	detected	w/o mechanism
count ± 1	250	250	—
entity substitution (fabricated)	250	250	—
real-but-uncited entity added	100	100	—
ordering operand swap	100	100	—
unit confusion (μ s as ms)	100	100	—
wrong belief state	60	60	—
truncated-page count	15	15	0
wrong-step citation	100	36	—
entity member dropped	100	0	—

indicating that the development figure was not an artifact of split selection.

5.4 Mechanism ablations and verifier validation

Output contracts (H2). With output-contract checking disabled on the development split, first-emission “validity” *rises* — the validator simply checks less — while execution success and accuracy fall: at 7B, validity 0.23→0.41 but execution success 0.55→0.23 and accuracy 0.136→0.091; at 14B, validity 0.50→0.55 but accuracy 0.409→0.318. Both runs were repeated at an equal generation budget with identical results. The check does not create or remove failures; it moves them from silent runtime deaths to statically repairable rejections.

Completeness tracking (H3). Disabling truncation-taint propagation on the development split lets 3 answers present a fully supported claim over truncated evidence (versus 0 with tainting); most development tasks fit one result page, which is why the controlled generator below isolates the mechanism more sharply.

Verifier fault injection. Table 5 summarizes all mutation classes. The classic classes perturb mechanically derived, fully grounded answers from oracle plans (87-answer pool; zero false positives). The database-semantics classes run on a dedicated non-frozen suite with an enriched 202-answer pool (zero false positives); a receipted rerun on the frozen-campaign store reproduces every rate. The wrong-belief-state class takes its incorrect value from a control execution of the same plan with `as_of_tt` removed; the truncated-count class shrinks one page limit so a correct count is computed over a partial page.

Two entries are deliberate negatives. Wrong-step citations are caught only when the surrogate step cannot ground the claim (a provenance pointer naming an uncited step is now rejected outright; the remaining 64 cases were genuinely grounded by the surrogate evidence). Dropped entity-set members are invisible to trace grounding by construction: the verifier checks that claimed identifiers appear in cited evidence (precision), while set completeness is checkable only against the database — gold scoring catches it, the trace verifier does not. The reasoning chain for the truncation row: value-only verification accepts a count that is arithmetically correct over the returned page; completeness metadata records that the page is partial; propagating that metadata through dependent steps prevents the count from being treated as a complete answer.

Table 6: Model-scale study. All rows: iTiger cluster, vLLM 0.11/cu126, temperature 0, one seed, frozen CollegeMsg test split; fp16 except the 72B row (AWQ 4-bit). Emitted unsupported-claim rate is 0.000 in every row.

model	TGMS		Static-graph	Text-to-Cypher
	acc.	probes	acc.	acc.
Qwen2.5-7B fp16	0.138	0.38	0.096	0.096
Qwen2.5-14B fp16	0.340	0.77	0.032	0.191
Qwen2.5-32B fp16	0.628	1.000	0.074	0.277
Qwen2.5-72B AWQ	0.511	0.31	0.096	0.138

5.5 Model and serving sensitivity

These post-campaign studies were run under a different serving configuration — a multi-GPU cluster (RTX 5000/6000 Ada, H100; the University of Memphis iTiger infrastructure [23]), full-precision weights except where noted, vLLM 0.11/cu126, one seed — against the same frozen splits and a byte-identically replayed canonical store. Absolute values should not be compared with the AWQ primary evaluation; the object is within-configuration trends.

Scale. Under the evaluated prompts, models, and representations, the operator-backed interface benefits more strongly from increasing full-precision model size than the two tested baselines (0.138 to 0.628 from 7B to 32B, with probe questions reaching 1.000 at 32B, while neither baseline exceeds 0.277). This suggests that the constrained plan representation makes additional planning capability more usable; it does not establish that serialized-context interfaces categorically cannot benefit from stronger models. The 14B full-precision row (0.340) differs from the primary 14B-AWQ result (0.408); the two were served on different hardware, engines, and precisions and are not directly comparable.

Quantization. The 72B-AWQ configuration underperforms the 32B full-precision configuration on both accuracy and probes. This result is consistent with quantization-related degradation in structured planning, but the present experiment does not separate quantization from model size and serving effects; a matched pair (32B fp16 vs. 32B AWQ, or 72B fp16 vs. 72B AWQ) is required to isolate the cause.

Long-context retrieval control. Two vector-RAG configurations must be distinguished. The *original chunking target* ($k=20$ chunks of 256 events, roughly 2×10^5 prompt tokens on this corpus) is not runnable on any serving window we tested; the primary campaign therefore used $k=1$ of 256 events, and a measured $k=2$ request of 36,956 tokens was rejected by the serving window. The *long-context control* ($k=20$ chunks of 24 events, $\sim 27k$ tokens, run on an H100 within the model’s native 32k window, 14B fp16) scored 0.021, consuming 32,475 tokens per task, against TGMS’s 0.362 at 6,521 tokens in the same run. Increasing retrieval breadth from the locally feasible configuration to this long-context configuration did not improve accuracy; the main result is therefore not explained solely by the small k of the original experiment. Chunking granularity, ranking, serialization, and prompt design were not exhaustively explored.

Cross-family transfer. With the same untuned operator manual, Llama-3.1-8B reaches 0.043 (verified insensitive to generation budget) and Phi-4-mini (3.8B) 0.015. The two smaller untuned

models produce few successful plans, indicating that portability to weaker model families is not automatic; determining whether a family-independent capability threshold exists requires matched-size experiments. Across the tested configurations — four scales, three families, two quantizations, two serving clusters — gated claim support remained stable (no measured unsupported gated claims) while answer accuracy and coverage varied substantially.

Constrained decoding. Grammar-constrained JSON decoding of the plan IR, evaluated as the escalation for syntactically invalid plans, degraded the 14B development result on every axis (accuracy 0.409→0.227, first-emission validity 0.50→0.32, 65% more wall time): syntactic validity was not the bottleneck, and post-generation checking with a repair loop was both safer and cheaper than constraining generation.

Same-information baseline and further studies. Three later pre-registered studies (repository decisions D-025 to D-027) sharpen the interface claim. First, a same-information baseline lets the model write SQL directly against the identical bi-temporal version store, with the schema and temporal semantics documented in the prompt and the same repair budget and answer contract. It matches TGMS on correction probes (0.846–0.923; history, not the interface, decides probe answerability) and separates by workload on overall accuracy: TGMS leads by +0.124 on CollegeMsg (95% CI [+0.071, +0.181]) while the two are statistically indistinguishable on a flatter financial-trust workload (Bitcoin-OTC, a fourth frozen suite of 94 tasks: 0.309 vs. 0.340, CI includes zero). Second, 110 independently written questions (authors saw only a data description, never the operator list) put the algebra’s coverage at 10/110, with grouped/distinct aggregation the dominant missing capability (30 questions blocked by it alone). Third, a dev-split diagnosis extended runtime repair to all structured executor refusals and made refusal payloads name the computed legal alternative, raising dev execution success from 7/22 to 12/22 without changing accuracy; frozen test numbers were not rerun. Full receipts accompany the artifact [?].

5.6 Operator performance

Operators are benchmarked in isolation on the host CPUs of the development server (two Xeon Silver 4210, 40 hardware threads, 94 GB RAM, DuckDB backend) over synthetic stores of 10^5 , 10^6 , and 10^7 edge versions, with one untimed warm-up and seven timed repetitions; values are operator-only p50 wall times. Two-hop snapshots run in 13/99/272 ms across the three scales, global diffs in 23/163/485 ms, and metric time series in 144/157/1127 ms; the 10^7 store occupies 2.6 GB. Over-budget traversals refuse with a narrowing suggestion rather than degrade. Prompt size stays ~6–10k tokens per task at every data scale because structure never enters the context window.

6 Design lessons and their experimental support

Each lesson follows the same chain: observed failure → missing system property → mechanism added → controlled test → consequence.

Lesson 1: models invent *output* fields. Our first live run scored 0 of 22 development tasks on execution — with every plan passing input validation. Plans referenced `s2.count` where the operator emits `rows_total`: argument schemas alone are insufficient for a model-generated dataflow plan, because nothing constrained what the model read *back*. TGMS added declared output fields

to the operator registry and static checking of every result reference, with the legal field list in the repair payload. In the controlled re-run, execution success on the three probe tasks rose from 0/3 to 3/3 and overall from 0/22 to 12/22; the effect replicates at 14B (Section 5.4). Result schemas deserve the same rigor as argument schemas.

Lesson 2: verification must check evidence completeness. A live 14B run called reachability, received the default page of 100 rows (of 343), and computed a flawlessly correct count — of the page. Value-only verification accepts that claim; it is false as a statement about the database. TGMS records completeness metadata on every result page and propagates truncation taint through dependent steps; claims resting on tainted evidence are capped below full support. The controlled generator in Section 5.4 shows the mechanism is load-bearing: 15/15 truncated-page counts refused full support with tainting, 0/15 without. The same treatment should extend to sampling, approximation, timeouts, and stale replicas — database answers already know when they are partial.

Lesson 3: serving limits shape baseline design. Numeric-dense event text tokenizes at roughly 0.65 tokens per character, so the original vector-RAG chunking target could not fit any window we could serve locally, and the primary campaign ran it at $k=1$. When later hardware allowed a runnable $k=20$ long-context control, added breadth did not improve the baseline (Section 5.5). Baseline configurations should be reported together with the serving limits under which they ran, and retrieval-breadth conclusions should be tested rather than assumed.

7 Related work

Temporal and bi-temporal graph databases. The distinction between valid time and transaction time is classical [24]. Recent systems add temporal support to property graphs. AeonG [11] uses a current and historical storage split. Gradoop’s TPGM supports distributed temporal graph analysis [20]. T-GQL proposes a temporal graph query language [3]. These systems are designed mainly for human-written queries in a general language. TGMS instead exposes a small operator algebra with explicit types, bounds, costs, and output fields. These contracts allow an LLM to construct plans and allow the system to verify their execution. We are not aware of prior work that combines agent-facing temporal operators, transaction-time reasoning, and trace-based answer checking in one graph management system.

Graph-augmented retrieval and agent memory. GraphRAG summarizes corpus-derived graphs for query-focused retrieval [5]. HippoRAG uses graph structure for long-term memory consolidation [9]. Zep/Graphiti uses a bi-temporal knowledge graph for agent memory [19], and TOKI formalizes bitemporal write-time operators for contradiction resolution in relational agent memory [26]. These systems manage what the agent remembers or retrieve graph-derived content into the model context. TGMS takes a different approach. The model does not receive the full graph structure. Operators compute over the graph, and the model composes the calls and reports the result. TGMS also quarantines summaries when later corrections overlap their source windows. TGMS shares the bi-temporal foundation of Zep and TOKI but targets temporal graph analytics: the checked artifacts are operator plans and final answer claims rather than memory writes.

Tool use, planning, and program-aided reasoning. Toolformer [22], ReAct [28], and LLM-Compiler [13] show that language models can call and compose external tools. Program-aided

methods delegate arithmetic and symbolic computation to an interpreter [8, 2]. ToolGate attaches Hoare-style pre- and postconditions to individual tool invocations [14]. TGMS applies these ideas to temporal graphs. Its plan is a constrained DAG over a fixed set of independently tested operators, checked statically before execution rather than call by call. The executable surface also enforces identifier grounding and output-field validity. Our output-contract result suggests that result schemas deserve the same care as argument schemas when tools are designed for small models.

Natural-language interfaces to databases. Text-to-SQL and related natural-language database interfaces map a question directly to a query [29]. Our text-to-Cypher baseline follows this pattern and uses the same models, repair budget, and answer format as TGMS. TGMS adds a finer-grained audit trail. Each claim points to named operator results and their digests, rather than relying only on re-execution of the generated query. Temporal KGQA benchmarks such as CronQuestions [21] evaluate time-aware questions over a fixed knowledge graph. Our correction probes instead test whether a system can distinguish a past belief state from the current corrected state.

Faithfulness and claim verification. RARR [7], FActScore [15], and Chain-of-Verification [4] compare generated text with retrieved or model-produced evidence; a recent survey frames such mechanisms as evidence tracing over execution provenance [25]. TGMS uses narrower evidence with stronger structure. Claims cite deterministic operator results, so supported values can be checked exactly. The executor also records whether evidence was truncated, which prevents a correct value computed over an incomplete page from being treated as fully supported.

Temporal graph analysis. TGMS follows standard temporal-network definitions for time-respecting paths [27, 10] and δ -temporal motifs [17]. The evaluation uses data from the SNAP and temporal graph benchmark tradition [16, 12]. TGMS places these algorithms behind bounded operator contracts and returns an explicit refusal when a request exceeds the cost limit. For reachability with a maximum-wait constraint, the system uses exact multi-label search instead of the processing-order-dependent greedy rule found during specification testing.

Positioning. TGMS combines three elements. It provides database-level temporal semantics with a transaction-time axis. It exposes an agent-facing computation surface with machine-checkable contracts. It verifies final claims against deterministic execution evidence. The correction probes test the first element, the output-contract experiment motivates the second, and fault injection evaluates the third. The tool server uses MCP [1], so it can be connected to agent frameworks that support the protocol.

8 Limitations

Workload alignment. The tasks and the operator set were co-designed; the study does not yet establish coverage of independently collected temporal-graph questions. **Domain scope.** The real datasets are communication networks; cross-domain generalization is untested. **Capability versus fairness.** Correction probes demonstrate the value of transaction-time information but compare systems with different representational capabilities (Table 3). **Coverage trade-off.** Zero unsupported gated claims is partly achieved through abstention; coverage and conditional accuracy are reported alongside it. **Verification scope.** Pattern claims are checked but not gated; approximate operators and probabilistic entity resolution are unsupported; the guarantees do not

extend to adversarial tool behavior or write-back. **Causal uncertainty.** The scale study does not isolate quantization from size and serving effects, and two small cross-family models cannot identify a general capability threshold. **System maturity.** Concurrency, isolation, incremental index maintenance, cost-based plan rewriting, and agent write-back remain open; write-back is schema-ready but disabled pending provenance and authorization policies. **Baseline scope.** The study evaluates particular vector-RAG, static-graph RAG, and text-to-Cypher implementations, not every retrieval or query-generation design.

9 Conclusion

TGMS explores a database architecture in which an LLM plans and reports, while the data system owns temporal semantics, deterministic computation, bounded execution, and evidence checking. The frozen evaluation shows that this separation is particularly valuable for questions that require transaction-time belief states and for answers whose support depends on complete intermediate results. The experiments also expose two general interface requirements: machine-generated plans need explicit output contracts, and claim verification must represent evidence completeness rather than checking values alone. TGMS does not yet solve general temporal graph question answering: its workload remains domain-specific, complex multi-operator plans frequently lead to abstention, and several claim types and approximate operators remain outside the gating boundary. These limitations motivate cost-based plan rewriting, broader workload coverage, concurrency control, and verification with approximate evidence.

Artifacts. Code, benchmark generator, task suites, guided demo, and trace viewer: <https://github.com/zxf-work/tgms> (Apache-2.0).

References

- [1] Anthropic. Model context protocol. <https://modelcontextprotocol.io>, 2024.
- [2] Wenhua Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*, 2023.
- [3] Ariel Debrouvier, Eliseo Parodi, Matías Perazzo, Valeria Soliani, and Alejandro Vaisman. A model and query language for temporal graph databases. *The VLDB Journal*, 30:825–858, 2021.
- [4] Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. Chain-of-verification reduces hallucination in large language models, 2023. arXiv:2309.11495.
- [5] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. From local to global: A graph RAG approach to query-focused summarization, 2024. arXiv:2404.16130.
- [6] Xiyang Feng, Guodong Jin, Ziyi Chen, Chang Liu, and Semih Salihoğlu. Kùzu graph database management system. In *Conference on Innovative Data Systems Research (CIDR)*, 2023.

- [7] Luyu Gao, Zhuyun Dai, Panupong Pasupat, Anthony Chen, Arun Tejasvi Chaganty, Yicheng Fan, Vincent Y. Zhao, Ni Lao, Hongrae Lee, Da-Cheng Juan, and Kelvin Guu. RARR: Researching and revising what language models say, using language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023.
- [8] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *International Conference on Machine Learning (ICML)*, 2023.
- [9] Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. HippoRAG: Neurobiologically inspired long-term memory for large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.
- [10] Petter Holme and Jari Saramäki. Temporal networks. *Physics Reports*, 519(3):97–125, 2012.
- [11] Jiamin Hou, Zhanhao Zhang, Zhouyu Wang, Yongjun Zhang, Wei Lu, Anqun Pan, and Xiaoyong Du. Aeong: An efficient built-in temporal support in graph databases. *Proceedings of the VLDB Endowment*, 17(6):1515–1527, 2024.
- [12] Shenyang Huang, Farimah Poursafaei, Jacob Danovitch, Matthias Fey, Weihua Hu, Emanuele Rossi, Jure Leskovec, Michael Bronstein, Guillaume Rabusseau, and Reihaneh Rabbany. Temporal graph benchmark for machine learning on temporal graphs. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023.
- [13] Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. An LLM compiler for parallel function calling. In *International Conference on Machine Learning (ICML)*, 2024.
- [14] Yanming Liu, Xinyue Peng, Jiannan Cao, Xinyi Wang, Songhang Deng, Jintao Chen, Jianwei Yin, and Xuhong Zhang. ToolGate: Contract-grounded and verified tool execution for LLMs. *arXiv preprint arXiv:2601.04688*, 2026.
- [15] Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen-tau Yih, Pang Wei Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. FActScore: Fine-grained atomic evaluation of factual precision in long form text generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.
- [16] Pietro Panzarasa, Tore Opsahl, and Kathleen M. Carley. Patterns and dynamics of users’ behavior and interaction: Network analysis of an online community. *Journal of the American Society for Information Science and Technology*, 60(5):911–932, 2009.
- [17] Ashwin Paranjape, Austin R. Benson, and Jure Leskovec. Motifs in temporal networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining (WSDM)*, pages 601–610, 2017.
- [18] 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.
- [19] Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A temporal knowledge graph architecture for agent memory, 2025. arXiv:2501.13956.

- [20] Christopher Rost, Kevin Gomez, Matthias Täschner, Philip Fritzsche, Lucas Schons, Lukas Christ, Timo Adameit, Martin Junghanns, and Erhard Rahm. Distributed temporal graph analytics with GRADOOP. *The VLDB Journal*, 31:375–401, 2022.
- [21] Apoorv Saxena, Soumen Chakrabarti, and Partha Talukdar. Question answering over temporal knowledge graphs. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2021.
- [22] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [23] Mayira Sharif, Guangzeng Han, Weisi Liu, and Xiaolei Huang. Cultivating multidisciplinary research and education on gpu infrastructure for mid-south institutions at the university of memphis: Practice and challenge, 2025.
- [24] Richard T. Snodgrass. *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann, 1999.
- [25] Yiqi Wang, Jiaqi Zhang, Taotao Cai, Zirui Liu, Qingqiang Sun, Zequn Sun, Zhangkai Wu, Manqing Dong, Mingkai Zheng, Xuefei Yin, and Yanming Zhu. From agent traces to trust: A survey of evidence tracing and execution provenance in LLM agents. *arXiv preprint arXiv:2606.04990*, 2026.
- [26] Ziming Wang. TOKI: A bitemporal operator algebra for contradiction resolution in LLM-agent persistent memory. *arXiv preprint arXiv:2606.06240*, 2026.
- [27] Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. Path problems in temporal graphs. *Proceedings of the VLDB Endowment*, 7(9):721–732, 2014.
- [28] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.
- [29] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.