Computing&AI Connect
ISSN: 3104-4719
2026, Article ID. 2026.0042, Cite as: https://doi.org/10.69709/CAIC.2026.155251

SCIFINITI PUBLISHING

Research Article

OPEN ACCESS

IncrConsist: Efficient Incremental Consistency Maintenance for LLM-Maintained Knowledge Bases

Bailing Zhang

School of Computer Science and Data Engineering, NingboTech University, Ningbo 315336, China

Article History
Received: May 15, 2026 Accepted: September 15, 2026 Published: October 25, 2026

Abstract

Background: LLM-based agents are increasingly deployed not only to answer queries but to maintain persistent, evolving knowledge bases—continuously ingesting sources, revising assertions, and reconciling contradictions. After every edit, the knowledge base must be verified for internal consistency; however, naïve full re-verification requires $O(n^2)$ pairwise comparisons, exceeding 90 seconds per edit step at a 500-page scale, rendering it impractical for continuous maintenance. **Methods:** We present IncrConsist, an incremental consistency maintenance framework that confines verification to a compact *affected set* derived from the edit's provenance graph neighborhood. The framework formalizes five atomic edit operations (Add, Delete, Modify, Merge, Split) and three layered consistency invariants: local schema (INV1), reference chain (INV2), and mutex-based global semantic consistency (INV3). Affected-set computation integrates strength-weighted provenance BFS, keyword-Jaccard semantic expansion, and word-boundary mutex detection, closing the graph-topology gap that prevents pure traversal from reaching globally inconsistent assertion pairs. **Results:** On a synthetic 61-page AI-course knowledge base (1,220 assertions, 908 typed provenance edges), IncrConsist achieves a 1,660× speedup over full re-verification, reduces the checked assertion count by 99.3% (average 10.0 nodes per step), and attains 0.80 violation recall against 0.66 for a naive 1-hop baseline. Scalability experiments across 50–500 pages confirm sub-linear time growth (3.4× for a 10× scale increase). **Conclusions:** IncrConsist is the only method tested to detect mutex-based global semantic contradictions, a class missed entirely by graph-traversal methods. The work demonstrates that incremental affected-set computation—classical in database materialization and ontology maintenance—can be adapted to LLM-maintained text knowledge bases, with extensions for typed provenance strength and lexical mutex conflict.

Keywords:
incremental consistency maintenance; LLM knowledge base; affected-set computation; provenance graph; truth maintenance; edit taxonomy; knowledge verification

1. Introduction

1.1. Background and Motivation

Large language models [1,2] have enabled agents based on LLMs that are increasingly expected to maintain persistent knowledge bases rather than produce isolated answers. In these systems, knowledge is continuously edited: new assertions are ingested from documents, existing claims are modified or deleted, pages are merged or split, and cross-references are updated over time [3,4]. This creates a maintenance problem that is easy to overlook. After each edit, the system must determine whether the knowledge base remains internally consistent: that individual assertions are well-formed, that dependency chains are intact, and that no two assertions contradict each other. Karpathy's LLM Wiki pattern [5] is one recent example of this broader shift; it proposes having the LLM act as a disciplined curator of a Markdown knowledge base,

Corresponding Author:
Bailing Zhang, School of Computer Science and Data Engineering, NingboTech University, Ningbo 315336, China;
bailing.zhang1961@gmail.com

© 2026 Copyright by the Author.
Licensed as an open access article using a CC BY 4.0 license.

2026, Vol. 3, Article ID. 2026.0042
https://doi.org/10.69709/CAIC.2026.155251

flagging contradictions and updating cross-references as knowledge evolves.

A naïve solution is full re-verification: recheck every assertion and every potentially conflicting assertion pair after every edit. This quickly becomes impractical. Pairwise semantic consistency checking scales as $O(n^2)$—for a knowledge base with $n = 10{,}000$ assertions, a single edit can trigger fifty million comparisons. Our experiments confirm that even a capped implementation (200,000 pair limit) takes over 90 seconds per edit step at 500 pages, making it incompatible with any real-time or interactive maintenance loop.

1.2. Technical Challenges

The classical answer—not to re-verify everything, but to compute the affected substructure induced by an edit—has been studied extensively in database and knowledge-representation systems. Incremental view maintenance and Datalog materialization algorithms, such as DRed [6] and the Backward/Forward algorithm [7], restrict re-derivation to the portion of the knowledge structure that actually depends on the changed facts. Ontology change management [8] similarly derives affected axioms by propagating changes along subsumption hierarchies. Truth-maintenance systems [9,10] maintain dependency-directed justifications so that belief status can be updated locally when a supporting assumption changes. These approaches share the essential principle: the verification cost should scale with the size of the affected neighborhood, not the total knowledge base.

However, applying this principle to LLM-maintained knowledge bases requires addressing two challenges that do not arise in classical databases. First, provenance links between assertions are *typed by semantic strength*: a conclusion derived by direct entailment should propagate changes further than one inferred by soft analogy or cited for context only. Classical incremental methods assume uniform edge semantics, but in LLM-maintained text, the dependency strength is heterogeneous. Second, *global semantic contradictions* can arise between assertions with no provenance edge—one page may describe a method as *supervised* while another describes it as *unsupervised*—so pure graph traversal is structurally insufficient to detect INV3-class inconsistencies. Closing this gap requires an additional semantic expansion step that goes beyond what traditional affected-set computation provides.

1.3. Contributions and Research Questions

This paper makes the following contributions.

- A taxonomy of five atomic edit operations with formal affected-set characterizations, and a three-layer invariant specification (local, reference, global mutex-based) with checkable predicates (Section 3).
- An affected-set computation algorithm combining strength-weighted provenance BFS with keyword-Jaccard semantic expansion and word-boundary mutex detection—the latter addressing INV3 detection without introducing false positives from substring ambiguity (Section 3.4).
- An empirical evaluation on a 1220-assertion synthetic AI-course knowledge base demonstrated a 1660× speedup, a 99.3% impact-domain reduction, and a violation recall of 0.80. Ablation analysis also shows that semantic expansion is a critical enabler of INV3 detection (Section 4).

The evaluation is organized around four research questions. **RQ1:** How does the affected set induced by an edit behave across the five types of atomic operation, and does it remain compact relative to the size of the knowledge base? **RQ2:** What efficiency does affected-set restriction deliver over full re-verification, and how does it scale with corpus size? **RQ3:** What fraction of injected violations does IncrConsist detect, relative to full re-verification and to a naive 1-hop baseline, and how does detection differ across the three invariant layers? **RQ4:** Which components of the computation of the affected set are responsible for detection and which affect only cost?

The remainder of this paper is organized as follows. Section 2 reviews related work in incremental view maintenance, ontology and knowledge-graph evolution, truth maintenance, consistency checking for LLM-generated content, and persistent agent memory, and positions IncrConsist against these families in a comparative table. Section 3 presents the materials and methods: the knowledge-base model, the edit-operation taxonomy, the three-layer invariants, the IncrConsist algorithm with its complexity analysis, the experimental design and the disclosed use of generative AI tools. Section 4 reports the results for the four research questions together with a parameter-sensitivity analysis, Section 5 discusses the findings and limitations, and Section 6 concludes.

2. Related Work

2.1. Incremental View Maintenance and Datalog Materialization

The problem of updating query answers after base-relation changes has been studied since the 1980s, beginning with efficient materialized-view update algorithms [11].

2026, Vol. 3, Article ID. 2026.0042
https://doi.org/10.69709/CAIC.2026.155251

The Delete/Re-derive (DRed) algorithm [6] handles incremental Datalog materialization by over-deleting all potentially affected tuples and then re-deriving those that remain valid under alternative derivations. The Backward/Forward (B/F) algorithm [7] avoids over-deletion by immediately searching for alternative derivations, reducing superfluous re-computation. Both methods illustrate the core principle IncrConsist adopts: rather than recomputing global consistency from scratch, identify the affected substructure and restrict verification to it. Unlike Datalog systems, however, IncrConsist operates on an approximate provenance graph extracted from LLM-maintained natural-language assertions, not a complete rule program; derivation chains are inferred rather than explicitly declared, and edge semantics are typed by strength rather than being logically uniform [12].

2.2. Ontology Evolution and Dynamic Knowledge Graph Maintenance

Ontology change management [8,13] characterizes knowledge evolution as sequences of primitive operations on schema or instance axioms and propagates changes along subsumption hierarchies to find affected axioms. Large-scale knowledge graphs such as Freebase [14], YAGO [15], Wikidata [16,17], and DBpedia [18] face analogous maintenance challenges, as facts are continuously added, retracted, or revised by automated or crowdsourced pipelines. IncrConsist differs from these settings in two respects: the maintained objects are unstructured natural-language assertions rather than fully formalized triples, and dependency relationships carry heterogeneous strength labels rather than being logically uniform. The latter requires the strength-weighted BFS of Section 3.4, which allows CTX-labeled edges to act as propagation firewalls.

2.3. Truth Maintenance and Belief Revision

Truth-maintenance systems (TMS) [9] maintain justification records for each believed proposition and update belief status when supporting assumptions change, avoiding global re-derivation by tracing dependency-directed chains. The assumption-based TMS (ATMS) [10] extends this to maintain multiple simultaneously consistent assumption sets. In the belief-revision literature, the AGM postulates [19] formalize the minimum-change principle for updating a belief set upon receiving new information. IncrConsist shares the dependency-directed update philosophy of TMS and the minimality spirit of AGM, but focuses on bounding the *verification region* rather than maintaining a complete logical closure or a provably minimal change set. The distinction is practical: IncrConsist is designed for text assertions whose logical semantics are only partially formalized, making a complete closure computation infeasible.

2.4. Consistency Checking for LLM-Generated Knowledge

Hallucination and factual inconsistency in LLM outputs have received considerable attention [20]. Methods such as FActScore [21] decompose the generated text into atomic facts and evaluate each against a reference corpus, while self-consistency checking [22] uses a majority vote across multiple LLM samples to identify unreliable outputs. Post-hoc verification-and-revision pipelines such as Chain-of-Verification [23] and RARR [24] correct a generated text against retrieved evidence, and analyses of parametric versus non-parametric memory [25] characterize when model-internal knowledge should not be trusted. Recent work has systematized *knowledge conflicts*—within a context, between contexts, and between context and parametric memory [26]—and *knowledge editing* techniques for updating facts stored in model parameters [27]; both lines concern the model's internal knowledge or its immediate prompt context, whereas IncrConsist governs the consistency of an external, persistent assertion store that outlives any single context window. Integration of knowledge graphs has been proposed as a grounding mechanism [28], and probing studies have characterized the internal knowledge structures of LLMs [29]. These methods predominantly address generation-time or query-time consistency: given a newly generated text, is it factually accurate? IncrConsist targets a different problem—*maintenance-time* consistency—asking which assertions in a persistent knowledge base need re-validation after a committed edit. The distinction is analogous to the difference between validating a new database tuple at insertion time and maintaining global integrity constraints after a sequence of batch updates.

2.5. Persistent Agent Memory and LLM-Maintained Knowledge Bases

LLM-based agents increasingly require persistent long-term memory to function effectively across sessions. Generative Agents [3] demonstrate that agents equipped with a memory stream—a natural-language log of experiences with retrieval, reflection, and planning components—exhibit coherent long-term behavior. MemGPT [4] extends this with OS-inspired hierarchical memory management that pages information in and out of context. The LLM Wiki pattern [5] proposes a distinct approach:

rather than a log, the agent maintains a structured, interlinked wiki whose pages are incrementally compiled and cross-referenced. Unlike retrieval-augmented generation [30], which re-derives context from raw sources at query time, such a wiki accumulates knowledge across sessions. More recently, graph-structured retrieval systems such as GraphRAG [31] and HippoRAG [32] construct entity graphs over document corpora to support global and multi-hop queries, and production memory layers such as Mem0 [33] and A-MEM [34] equip agents with explicit add/update/delete memory operations over an evolving store. These systems continuously *structure and update* accumulated knowledge, which makes the maintenance problem studied here directly applicable to them; yet none verifies the internal consistency of the store after each update. What these systems currently lack is a principled maintenance layer that verifies consistency as the wiki evolves—the gap IncrConsist is designed to fill. A structured comparison across all five families is given in Table 1.

2.6. Positioning Relative to Prior Work

Table 1 summarizes how IncrConsist differs from the four established research families reviewed above and from recent graph-structured retrieval and agent-memory systems, along five dimensions: the object being maintained, the semantics attached to dependency edges, the scope of consistency that is checked, when verification happens, and whether verification is incremental in the size of the change rather than the size of the store. No prior line of work combines strength-typed provenance propagation with global semantic conflict detection over unstructured text assertions at maintenance time, which is precisely the combination the LLM-wiki setting requires.

Table 1: Positioning of IncrConsist relative to prior research families. "Verification timing" distinguishes checking a newly generated artifact (generation time), answering from a store (query time), and re-validating a persistent store after a committed edit (maintenance time).

Family	Maintained Object	Edge Semantics	Consistency Scope	Verification Timing	Incremental
Incremental view maintenance/Datalog [6,7,11]	Materialized relations	Logically uniform rules	Derivability of tuples	Maintenance time	Yes
Ontology/KG evolution [8,13,14,16]	Formalized axioms/triples	Subsumption, typed relations	Schema and instance axioms	Maintenance time	Partially
TMS/belief revision [9,10,19]	Believed propositions	Justification links	Logical closure/ minimal change	Maintenance time	Yes
LLM-output consistency [21–24]	Generated text	—	Factuality vs. reference corpus	Generation/query time	No
Graph retrieval & agent memory [31–34]	Entity graphs, memory items	Similarity, co-occurrence	None (retrieval quality only)	Query time	—
IncrConsist (this work)	Unstructured text assertions	Strength-typed provenance (HARD/SOFT/CTX)	Schema, references, global mutex conflicts	Maintenance time	Yes

3. Materials and Methods

This section specifies the knowledge-base model and the three-layer consistency invariants (Sections 3.1–3.3), the IncrConsist algorithm and its complexity analysis (Sections 3.4–3.6), the experimental design used to evaluate it (Section 3.7), and the disclosed use of generative AI tools (Section 3.8).

3.1. Knowledge Base Model

An LLM wiki is a pair $\mathcal{K} = (\mathcal{A}, G)$ where $\mathcal{A}$ is a finite set of *assertions* and $G = (V, E, \sigma)$ is a typed directed provenance graph. Each assertion $a \in \mathcal{A}$ carries a content string $a.content$, a confidence score $a.conf \in [0, 1]$, a source type $a.src \in \mathcal{S}$, page identifier $a.page$, and status $a.status$. Each directed edge $(u, v) \in E$ carries a strength label $\sigma(u, v) \in \{\text{HARD}, \text{SOFT}, \text{CTX}\}$: HARD edges encode logical entailment (retraction of u necessarily invalidates v); SOFT edges encode probabilistic inference (retraction of u weakens but does not invalidate v); CTX edges encode contextual reference (retraction of u does not propagate to v). Table 2 summarizes the notation used throughout Sections 3–4.

2026, Vol. 3, Article ID. 2026.0042
https://doi.org/10.69709/CAIC.2026.155251

Table 2: Notation used in this paper.

Symbol	Meaning
$\mathcal{K} = (\mathcal{A}, G)$	LLM wiki: assertion set and provenance graph
$\mathcal{A}$, n	assertion store; $n = \lvert\mathcal{A}\rvert$
$G = (V, E, \sigma)$	typed directed provenance graph
$\sigma(u, v)$	edge strength label $\in \{\text{HARD}, \text{SOFT}, \text{CTX}\}$
$a.content$	content string of assertion a
$a.conf$	confidence score $\in [0, 1]$
$a.src \in \mathcal{S}$	source type; $\mathcal{S} = \{\text{DOC_EXTRACT}, \text{DOC_ORIGIN}, \text{LLM_INFER}, \text{DERIVED}, \text{USER_EDIT}, \text{CROSS_DERIVE}\}$
$a.page$, $a.status$	page identifier; lifecycle status
$\mathcal{S}_{\text{inv}}$	invalid statuses {deleted, merged, split, retracted}
$\Phi(op)$, k	affected set of edit op; $k = \lvert\Phi\rvert$
$\text{sub}(v)$	BFS-reachable downstream subgraph of v
$\text{deps}(a)$, $\text{sem}(a)$	direct dependencies; semantic neighbors
$\mathcal{M}$, $\mu_M(a)$	mutex groups; terms of group M in $a.content$
$\text{wb}(t, \cdot)$	word-boundary match of term t
$\rho(a)$	risk score, Equation (4)
$d_{\max}$	maximum BFS depth (default 3)
k_{sem}	semantic top-k (default 10; written k in Section 4.5)
τ	INV3 node cap (150; secondary 5000-pair cap)

3.2. Edit Operation Taxonomy

Wiki maintenance is modeled as sequences of atomic edit operations. Table 3 defines the five operation types and their affected-set complexity.

Table 3: Five atomic edit operation types: formal affected-set characterization, asymptotic complexity, whether the operation triggers semantic and mutex expansion (Sem. exp.), and the measured mean affected-set size and time per step from the 200-step main run (identical to Table 4). $\text{sub}(v)$ denotes the BFS-reachable downstream subgraph from v; S denotes the number of semantic neighbor candidates (default $k = 10$); D denotes the number of direct dependency links.

Type	Affected Set	Complexity	Sem. Exp.	Mean $\lvert\Phi\rvert$	ms/ Step
Add	$\{a'\} \cup \text{deps}(a') \cup \text{sem}(a')$	$O(D+S)$	yes	8.60	21.1
Delete	$\{a\} \cup \text{sub}(a)$	$O(\lvert\text{sub}\rvert)$	no	2.14	0.7
Modify	$\{a\} \cup \text{sub}(a) \cup \text{sem}(a, c')$	$O(\lvert\text{sub}\rvert+S)$	yes	18.40	129.8
Merge	$\{a_i, \ldots\} \cup \bigcup_i \text{sub}(a_i)$	$O(k\lvert\text{sub}\rvert)$	no	5.96	7.5
Split	$\{a, a'_1, a'_2\} \cup \text{sub}(a)$	$O(\lvert\text{sub}\rvert)$	no	3.87	1.9

3.3. Three-Layer Consistency Invariants

INV1 (Local consistency). An assertion a satisfies INV1 if it is schema-valid and no same-page sibling triggers a configurable conflict rule:

$$\text{INV1}(a, \mathcal{A}) \equiv \text{schema}(a) \ \wedge \ \forall a' \in \text{sibs}(a) : \neg\text{rule}(a, a'). \quad (1)$$

The current rule table contains six patterns (e.g., "CNN does not use self-attention"; "unsupervised does not require labeled data"). Schema validity requires a non-empty content string of at most 10,000 characters, a numeric $a.conf \in [0, 1]$, and $a.src \in \mathcal{S}$ (the six source types listed in Table 2). The two LLM-provenance types (LLM_INFER, DERIVED) are the assertions weighted more heavily by w_{src} in the risk score of Equation (4).

INV2 (Reference consistency). All listed parent dependencies must exist and have active status:

$$\text{INV2}(a, \mathcal{A}) \equiv \forall p \in a.deps : \ p \in \mathcal{A} \ \wedge \ p.status \notin \mathcal{S}_{\text{inv}}. \quad (2)$$

Invalid statuses $\mathcal{S}_{\text{inv}} = \{\text{deleted}, \text{merged}, \text{split}, \text{retracted}\}$.

INV3 (Mutex-based global consistency). We maintain mutex groups $\mathcal{M}$, each containing terms that are mutually exclusive when applied to the same topic. For any two assertions a, b of active status within the affected set:

$$\text{INV3}(a, b) \equiv \exists M \in \mathcal{M} : \ \mu_M(a) \neq \emptyset \ \wedge \ \mu_M(b) \neq \emptyset \ \wedge \ \mu_M(a) \cap \mu_M(b) = \emptyset, \quad (3)$$

where $\mu_M(a) = \{t \in M \mid \text{wb}(t, a.content)\}$ extracts terms via word-boundary regular expressions.
Word-boundary matching is essential: naïve substring search would extract *supervised* from within *unsupervised*, producing false conflicts.

Five mutex groups are currently defined:

$\{supervised, unsupervised, self\text{-}supervised\}$
$\{online, offline\}$; $\{generative, discriminative\}$
$\{deterministic, stochastic\}$
$\{parametric, non\text{-}parametric\}$

Pairs are compared irrespective of page membership: the INV3 class of interest is precisely contradiction between assertions that live on different pages and are therefore never co-located for a human reader. In addition to

2026, Vol. 3, Article ID. 2026.0042
https://doi.org/10.69709/CAIC.2026.155251

mutex-group conflict, the INV3 checker applies six direct-negation templates that match a positive predication in one assertion against its explicit negation in another ("X uses Y" versus "X does not use Y", with analogous templates for *is*, *are*, *does*, *can*, and *requires*). Both mechanisms are purely lexical; the resulting scope limitation is discussed in Section 5.3. Only assertions whose status is active participate in the INV3 pairing, which is the mechanism underlying the state-accumulation effect analyzed in Section 5.1.

3.4. Affected-Set Computation

The affected set $\Phi(op)$ is a *compact operational approximation* to the set of assertions whose consistency status may have changed after applying the editing operation *op*. It is not provably minimal because the provenance graph is itself an approximation; its purpose is to bound the verification region while maintaining high recall.

Strength-weighted BFS. Starting from the operation's primary target, the algorithm traverses forward provenance edges with propagation rules: $(\text{HARD}, \text{HARD}) \to \text{HARD}$; $(\text{HARD}, \text{SOFT}) \to \text{SOFT}$; any edge into CTX stops propagation. A maximum depth $d_{\max} = 3$ prevents runaway traversal. This modification over a plain BFS allows CTX edges to act as propagation firewalls without requiring explicit edge deletion.

Semantic expansion for Add/Modify. BFS cannot detect INV3 conflicts between assertions with no provenance edge. For Add and Modify operations, two supplementary expansions augment Φ: (i) keyword-Jaccard top-k neighbors of the modified content (general surface similarity), and (ii) all assertions containing *conflicting* mutex terms under word-boundary matching (targeted INV3 coverage). Expansion (ii) is an $O(n)$ scan per Modify step; an inverted index would reduce this to $O(\log n)$. Crucially, semantic expansion is not an efficiency trick but a correctness mechanism: without it, INV3 conflicts between graph-disconnected assertions cannot be discovered, as the ablation in Section 4.4 confirms.

3.5. Invariant Checking and Risk Scheduling

Within the affected set, nodes are processed in decreasing order of a risk score:

$$\rho(a) = 3 \cdot \mathbf{1}[a = v^*] + 2 \cdot \text{outdeg}(a) + 1.5 \cdot w_{src}(a) + (1 - a.conf), \quad (4)$$

where v^* is the directly edited node and w_{src} weights the LLM-inferred and derived assertions more heavily (more error-prone). Ordering reduces the mean time to first violation detection (MTTFD) without altering total recall, since all affected nodes are eventually checked.

Algorithm 1 states the entire procedure. INV1 and INV2 are checked in two passes over the risk-ordered affected set; both are $O(k)$ where $k = |\Phi|$. The INV3 pairwise check is $O(k^2)$ with a safety cap $\tau = 150$ on the number of nodes compared, plus a secondary cap of 5000 pairs inside the checker; if $|\Phi| > \tau$, only the τ highest-risk nodes are compared. In the experiments the average affected-set size is $k = 10.0$ (45 pairs per step) and the largest observed is 86 nodes (3655 pairs), so neither cap was activated in any of the 200 steps and no INV3 check was truncated.

Algorithm 1 IncrConsist

Require: Edit *op*, provenance graph G, assertion store $\mathcal{A}$
Ensure: Violation report $\mathcal{V}$

Apply *op* to $\mathcal{A}$
$\Phi \leftarrow \text{BFS}(op.target, G, d_{\max})$
if $op.type \in \{\text{ADD}, \text{MODIFY}\}$ **then**
  $\Phi \leftarrow \Phi \cup \text{SemanticNbrs}(op, \mathcal{A}, k)$
  $\Phi \leftarrow \Phi \cup \text{MutexNbrs}(op.content', \mathcal{A})$
end if
Sort Φ by $\rho(\cdot)$ descending
$\mathcal{V} \leftarrow \emptyset$
for all $a \in \Phi$ **do**
  if $\neg\text{INV1}(a, \mathcal{A})$ **then** $\mathcal{V} \mathrel{+}= (a, \text{INV1})$
  end if
end for
for all $a \in \Phi$ **do**
  if $\neg\text{INV2}(a, \mathcal{A})$ **then** $\mathcal{V} \mathrel{+}= (a, \text{INV2})$
  end if
end for
$\Phi' \leftarrow$ if $|\Phi| \le \tau$ then Φ else top-τ by ρ
for all $(a, b) \in \binom{\Phi'}{2}$ **do**
  if $\neg\text{INV3}(a, b)$ **then** $\mathcal{V} \mathrel{+}= (a, b, \text{INV3})$
  end if
end for
return $\mathcal{V}$

3.6. Complexity Analysis

We consolidate the computational cost of one IncrConsist step; let $n = |\mathcal{A}|$ be the number of assertions and $k = |\Phi|$ be the size of the affected set. *Affected-set construction.* Per-operation costs follow the taxonomy of Table 3: traversal-only operations (Delete, Merge, Split) cost $O(|\text{sub}|)$, bounded by the downstream subgraph under $d_{\max}$; expansion-bearing operations (Add, Modify) add the semantic terms, of which the keyword-Jaccard top-k contributes a bounded number of candidates by construction while the mutex-term scan is $O(n)$ per step—the single asymptotically dominant term. *Invariant check-*

2026, Vol. 3, Article ID. 2026.0042
https://doi.org/10.69709/CAIC.2026.155251

ing. INV1 and INV2 are $O(k)$ passes over the risk-ordered set; INV3 is $O(k^2)$ pairwise comparison, capped at $\tau = 150$ nodes and 5000 pairs. *Amortized cost.* Under the 30/30/20/10/10 operation mix of the experiments, the expected per-step cost is $0.6\,c_{\text{exp}} + 0.4\,c_{\text{trav}}$, where $c_{\text{exp}} = O(n + k^2)$ and $c_{\text{trav}} = O(|\text{sub}| + k^2)$; with the observed mean $k = 10.0$, the k^2 term is negligible (45 pairs) and the $O(n)$ mutex scan dominates, which the measured profile confirms: Modify averages 129.8 ms against 0.7–7.5 ms for traversal-only operations (Table 4), and total step time grows $3.4\times$ over a $10\times$ corpus increase (Section 4.2). Replacing the linear scan with an incrementally maintained inverted index (mutex term $\rightarrow$ assertion-ID list) would reduce this term to $O(\log n)$ retrieval, as discussed in Section 5.2. By contrast, full re-verification is $O(n^2)$ pairwise, which the 200,000-pair cap converts into a large constant in our baseline (90+ seconds per step at 500 pages).

3.7. Experimental Design

Dataset. We use a synthetic AI-course wiki to obtain controlled edit operations and ground-truth violation labels at scale. Real LLM-maintained wikis rarely provide complete records of edit-induced violations with known labels, making controlled injection necessary to measure recall. The synthetic generator is parameterized to match the page granularity, edge density, and source-type distribution observed in our real WikiProvenance-Lite corpus. Specifically, a 61-page wiki is generated with 20 assertions per page, covering 61 topics (supervised learning, transformers, offline reinforcement learning, etc.), yielding 1220 initial assertions. The provenance edges are generated by an attachment procedure in which each assertion (other than the first) samples up to three previous assertions and links to each with probability $p_{\text{dep}} = 0.25$, which restricts the graph to a DAG and yields an expected ≈914 edges; the realized graph has 908 edges, of which 62.0% are HARD, 28.7% SOFT, and 9.3% CTX. Because 523 of the 1220 assertions carry at least one outgoing edge, the mean out-degree is $908/523 = 1.74$ over those source nodes (maximum 8) and $908/1220 = 0.74$ when averaged over all assertions; the conventional directed-graph density is $908/(1220 \times 1219) \approx 6.1 \times 10^{-4}$. The sparse, locally clustered topology this produces is what makes affected-set restriction effective, and it also motivates the semantic expansion step, since most assertion pairs share no provenance path at all.

Edit sequence. We simulate 200 consecutive edits (seed = 42) with type distribution: Add 30%, Modify 30%, Delete 20%, Merge 10%, Split 10%. Fifty steps carry injected violations (25% injection rate): 24 INV1 violations (empty content or *conf* > 1.0), 10 INV2 violations (deleting a parent assertion that has active children), and 16 INV3 violations (modifying content to a known mutex-contradicting template). Each violation is applied via the corresponding operation payload and verified by full re-verification as a gold standard.

Baselines. (B1) *Full Check*: scan all assertions after every edit, with a 200,000 pairwise-comparison cap for INV3 to make it computationally feasible. (B2) *Naive 1-hop*: check the edited node and its immediate graph neighbors only. To the best of our knowledge, no previously published method targets maintenance-time consistency verification for LLM-maintained text knowledge bases, so no directly comparable system exists for this task; the two baselines instead instantiate the two paradigm families applicable to it—exhaustive re-verification, which serves as the recall upper bound, and classical dependency propagation as used in ontology evolution and truth maintenance (Table 1). The complete data set, the edit sequence, the gold violation labels, and the evaluation scripts are publicly released so that the benchmark can serve as a common basis for future methods (see the Availability of Data and Materials section).

Parameter settings. Three parameters govern the computation of the affected set. The BFS depth cap $d_{\text{max}} = 3$ limits the worst-case traversal; because HARD-chain derivations in the corpus are shallow, deeper traversal adds no reachable nodes in practice. The semantic top-k $= 10$ balances INV3 coverage against the per-step cost of the Jaccard expansion. The INV3 safety cap $\tau = 150$ (with a secondary 5000-pair cap) protects against pathological affected sets; neither cap was activated in any of the 200 main-run steps (Section 3.5). The risk-score weights of Equation (4) were specified a priori to prioritize the directly edited node, high-fanout nodes, and LLM-derived assertions, and affect only detection latency, not recall (Section 4.4). A sensitivity analysis of $d_{\text{max}} \in \{1, 2, 3, 4\}$ and $k \in \{5, 10, 20\}$ is reported in Section 4.5.

Metrics. Violation recall = detected/gold; miss rate = 1 − recall; average time per step (ms); speedup = Full Check time/IncrConsist time; impact-domain reduction = $1 - |\Phi|/|\mathcal{A}|$.

Implementation and environment. The framework is implemented in pure Python 3.10 using only the standard library: the provenance graph is stored as forward/backward adjacency dictionaries traversed by a purpose-built strength-weighted BFS, and invariant checking uses the standard library regular expression engine; the released pipeline has no third-party runtime dependencies, which simplifies exact reproduction. All experiments were executed single-threaded on one machine (In-

tel Core i7, 16 GB RAM, Windows 11). Every detection count and affected-set statistic reported below is exactly reproducible from the released code with `RANDOM_SEED` = 42; wall-clock timings are hardware-dependent and were measured on the single machine described above.

3.8. Use of Generative AI Tools

In accordance with the COPE guidance on the use of AI tools in scholarly publishing, the author discloses the following. Generative AI assistance was used in two distinct phases, with different models.

Phase 1: original submission (March–May 2026). The generative AI assistant Claude (Anthropic; model `claude-sonnet-4-6`), accessed through the Anthropic web interface, was used in three delimited roles. (i) *Code drafting*: initial Python scaffolding for the experimental pipeline (`step01`–`step05`, `core/`, `engine/`) was drafted from the author's written specifications and pseudocode, then reviewed, corrected, and checked line by line by the author; all algorithmic design decisions—the strength-propagation table, the word-boundary mutex matcher, the risk-score weights of Equation (4), and the τ safety cap—were specified by the author, not by the model. (ii) *Literature search assistance*: the model suggested candidate related work; the author independently retrieved and read every cited item and verified all bibliographic details and DOIs against the publisher of record. (iii) *Language editing*: author-written prose was edited for grammar and concision.

Phase 2: revision (August–September 2026). In preparing the revised manuscript in response to editorial and reviewer comments, the author used Claude (Anthropic; model `claude-opus-5`) in the following roles. (i) *Supplementary experiment code*: the two scripts added during revision, `step06_sensitivity.py` (parameter sensitivity, Section 4.5) and `step07_baselines.py` (additional baselines, Section 4.3), were drafted with AI assistance rather than written unaided; the model also proposed the specific configurations explored (the $d_{\max} \times k$ grid and the 2-hop and Random-K baselines). Both scripts reuse the replay logic and gold labels of the existing pipeline, were reviewed by the author, and were executed by the author on the author's own machine; the `step06` cross-check reproducing the main-run values (recall = 0.800, mean $|\Phi| = 10.0$ at $d_{\max} = 3$, $k = 10$) was used to verify that they replay the same experiment. (ii) *Figure regeneration*: the plotting scripts in `plots/` that regenerate every figure from the released result files, and the graphical abstract were drafted with AI assistance; the underlying values are read directly from the experimental outputs and are unchanged. (iii) *Revision drafting and language editing*: AI assistance was used in drafting revised and newly added expository passages, including the comparative positioning table (Table 1), the notation table (Table 2), the complexity analysis (Section 3.6), and the consolidated limitations (Section 5.4). All such text was reviewed, corrected, and approved by the author, who verified every factual and numerical claim against the experimental record.

In both phases, the tool was *not* used for the experimental design of the main study, the generation of a synthetic corpus, the execution of any experiment, or the computation or aggregation of any reported number: every figure in this manuscript reports values measured on the machine of the author, and all scientific interpretation and conclusions are the author's own. No LLM inference occurs anywhere inside the evaluated pipeline: all three invariant checks are deterministic regular-expression and graph operations, which is why the reported detection counts are exactly reproducible under a fixed seed. The author reviewed and validated all AI-assisted content and takes full responsibility for the entire manuscript.

4. Results

4.1. RQ1: Affected-Set Structure Across Edit Types

Table 4 and Figure 1 decompose the 200-step main run by operation type.

Table 4: Affected-set size and cost per edit operation type over the 200-step main run ($|\mathcal{A}| = 1335$ active assertions at mid-experiment).

Operation	Steps	Mean $\lvert\Phi\rvert$	Median	Min–Max	ms/Step
Add	62	8.60	11	1–12	21.1
Modify	65	18.40	12	11–86	129.8
Delete	35	2.14	2	1–7	0.7
Merge	23	5.96	4	3–22	7.5
Split	15	3.87	3	3–7	1.9
All	200	**9.99**	11	1–86	**49.8**

Three observations follow. First, the affected set is compact for every type of operation: the mean of all 200 steps is 9.99 nodes versus 1335 active assertions, that is, 0.75% of the knowledge base, and the largest affected set observed is 86 nodes (6.4%). Verification cost is therefore governed by the local neighborhood of the edit rather than by corpus size, which is the property that makes continuous maintenance feasible at all.

2026, Vol. 3, Article ID. 2026.0042
https://doi.org/10.69709/CAIC.2026.155251

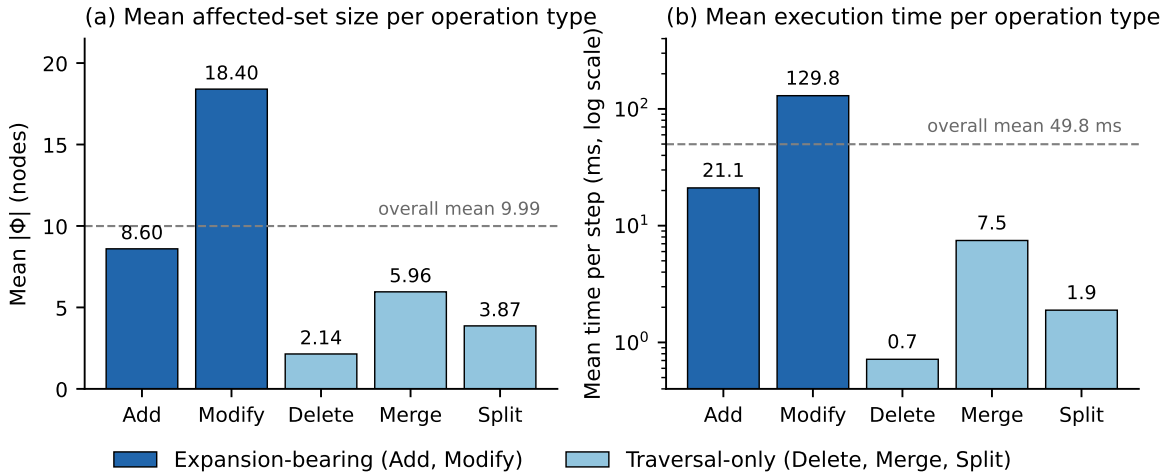

Figure 1: Affected-set structure and cost by edit operation type over the 200-step main run. (**a**) Mean affected-set size $|\Phi|$ per operation type; (**b**) mean execution time per step per operation type (log scale). Dashed lines mark the overall means (9.99 nodes; 49.8 ms). All values are measured and identical to Table 4. Modify dominates both panels because it is the only operation that combines downstream traversal with full semantic and mutex expansion.

Second, the types of operation fall into two regimes that follow directly from the taxonomy in Table 3. Delete, Merge, and Split are *traversal-only* operations: their affected sets are bounded by the downstream subgraph (means of 2.14, 5.96, and 3.87 nodes) and they cost well under 10 ms per step. Add and Modify additionally trigger semantic and mutex expansion, giving means of 8.60 and 18.40 nodes at 21.1 and 129.8 ms, respectively. Modify is the most expensive operation because it is the only one that combines both mechanisms—strength-weighted BFS over the existing dependency subgraph *and* expansion over the rewritten content—and it also produces the heaviest tail (maximum 86 nodes), which occurs when the new content introduces a mutex term that is widely distributed across the corpus.

Third, this asymmetry explains the aggregate cost profile: under the 30% Add/30% Modify mix used here, expansion-bearing operations account for 63% of steps but for more than 90% of total runtime. The optimization effort should therefore target the mutex scan on the Modify path—the inverted-index change discussed in Section 5.2—rather than the traversal component, whose cost is already negligible.

4.2. RQ2: Efficiency and Scalability

Figure 2 reports the average execution time per edit step and the reduction in the impact domain as the corpus grows from 50 to 500 pages (1019 to 10,018 assertions).

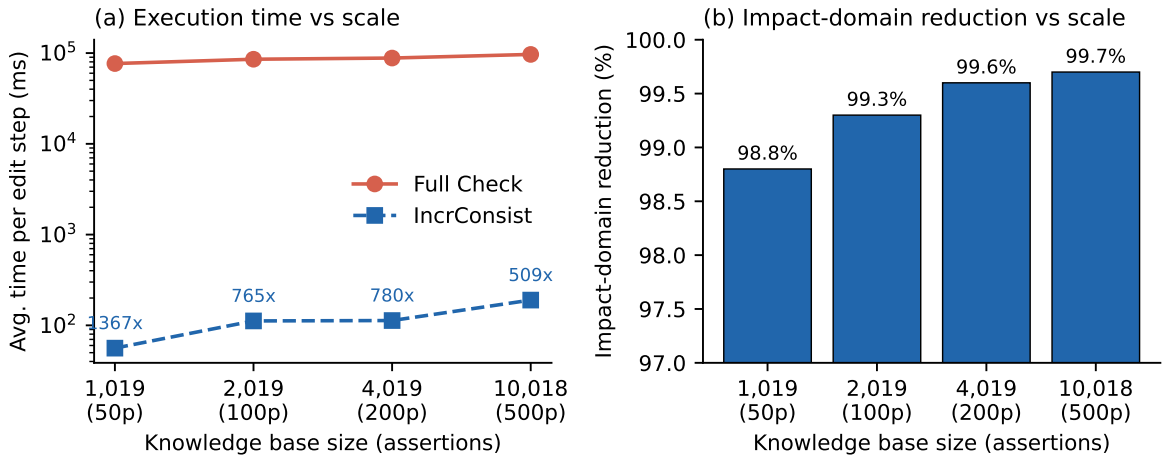

Figure 2: Scalability of IncrConsist across four corpus sizes. (**a**) Average execution time per edit step (log scale); speedup annotations show the Full-Check-to-IncrConsist ratio. (**b**) Impact-domain reduction (percentage of assertions not checked by IncrConsist).

IncrConsist ranges from 56 to 190 ms per step across scales—a 3.4× increase for a 10× increase in assertions—while Full Check ranges from 76,666 to 96,651 ms. The Full Check plateau (1.3× growth) reflects the 200,000-pair INV3 cap absorbing the true $O(n^2)$ growth; even so, 90+ seconds per step at 500 pages is operationally infeasible. IncrConsist speedup ranges from 509× (500 pages) to 1367× (50 pages); the decrease with scale arises because the $O(n)$ mutex scan in the semantic expansion step grows linearly with the size of the knowledge base. The reduction in the impact domain increases from 98.8% at 50 pages to 99.7% at 500 pages. The average affected-set size is not constant but grows sublinearly with the corpus—12.4, 15.1, 15.4, and 31.4 nodes at 50, 100, 200, and 500 pages, respectively, a 2.5× increase for a 10× increase in assertions—and this is what drives the observed 3.4× growth in time per step. The growth originates in the semantic-expansion step: keyword-Jaccard top-k contributes a bounded number of nodes by construction, but the mutex-term scan returns every active assertion carrying a conflicting term, and that population grows with the corpus. Because the affected set nevertheless grows far more slowly than $|\mathcal{A}|$, the reduction in the impact domain continues to improve with scale.

Violation recall at each scale point (0.80, 0.70, 0.90, 0.90) is reported in the released result files for completeness, but these values are not comparable across scales or with the main experiment: each scale point uses only 50 edit steps with 10 injected violations, so a single detection shifts recall by 10 points. The 200-step run in Section 4.3 is the primary accuracy result.

4.3. RQ3: Violation Detection Accuracy

Table 5 summarizes the main 200-step experiment. IncrConsist achieves 0.80 recall at 1660× speedup, improving 14 percentage points over the Naive baseline. In the 50-violation gold set, the 95% Wilson confidence intervals are $[0.67, 0.89]$ for IncrConsist and $[0.52, 0.78]$ for Naive 1-hop; because the entire pipeline is deterministic under the fixed seed, there is no run-to-run variance, and these intervals quantify the uncertainty arising from the finite gold set.

Figure 3 disaggregates the recall by invariant layer.

Table 5: Main experiment results (200-step sequence, 50 gold violations, 1335 active assertions at mid-experiment). B3 is the 2-hop traversal and B4 the Random-10 control of Section 4.3. All timing values are averages over 200 steps; speedups are computed from unrounded timings.

Method	Recall	Miss	ms/Step	Speedup
Full Check (B1)	1.000	0.000	82,716	1×
Naive 1-hop (B2)	0.660	0.340	0.3	275,720×
2-hop (B3)	0.680	0.320	0.4	185,461×
Random-10 (B4)	0.500	0.500	21.3	3892×
IncrConsist	**0.800**	**0.200**	**49.8**	**1660**×

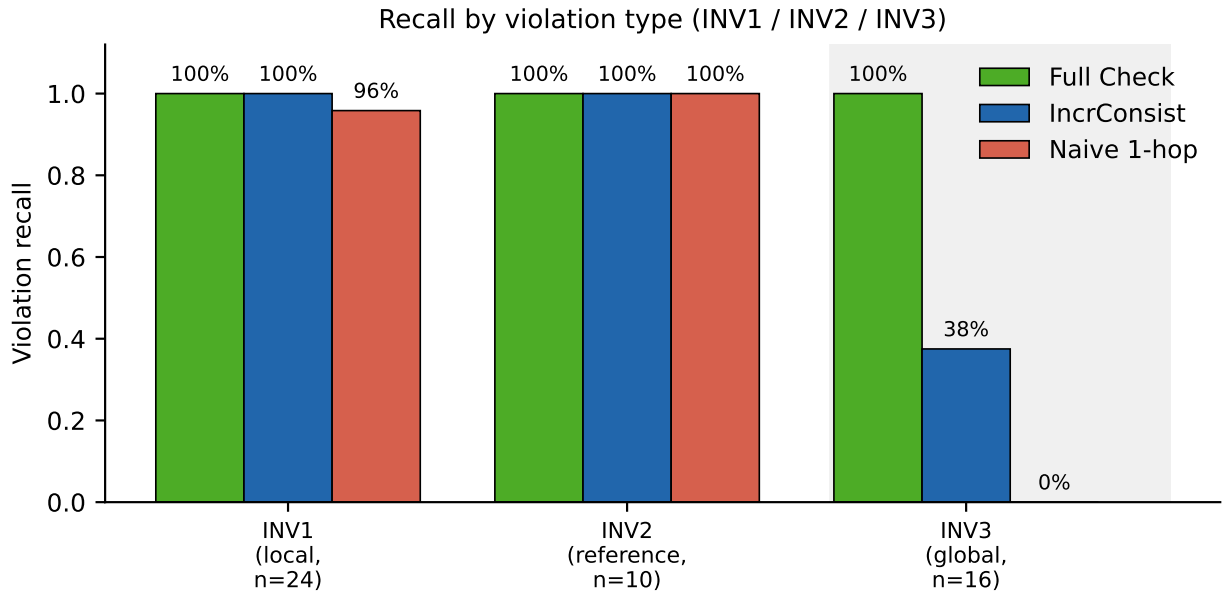

Figure 3: Violation recall by invariant layer (INV1/INV2/INV3) for each method. Numbers above bars indicate recall percentage. The INV3 region (shaded) shows that Naive 1-hop detects no mutex-based global contradictions, while IncrConsist detects 37.5%.

2026, Vol. 3, Article ID. 2026.0042
https://doi.org/10.69709/CAIC.2026.155251

For INV1 (schema violations, $n = 24$), IncrConsist and Full Check both reach 100%, while Naive 1-hop reaches 95.8% (23 of 24); the single miss is a violation injected on an assertion that is neither the edit target nor a 1-hop graph neighbor but is reached by IncrConsist through same-page semantic expansion. Since schema violations are by construction local to the edited assertion, near-saturation for all methods is expected, and this layer does not discriminate between them. For INV2 (broken references, $n = 10$), all three methods achieve 100%: the deleted parent is directly targeted, and strength-weighted BFS always reaches its dependent children. The discriminating layer is INV3 (mutex-based global contradictions, $n = 16$), where IncrConsist detects 6 of 16 (37.5%) and Naive 1-hop detects none: without semantic expansion, no method based solely on graph traversal can reach mutex-conflicting assertions that share no provenance edge. The entire 14-point aggregate recall gap between IncrConsist and Naive 1-hop is therefore attributable to INV3 detection. The partial (rather than complete) INV3 recall is analyzed in Section 5.1.

Two additional baselines sharpen this picture (Table 5; reproducible from `step07_baselines.py`). *2-hop traversal* (B3) deepens the 1-hop scheme by one level and gains only a single detection over it (recall 0.680, CI $[0.54, 0.79]$): the one INV3 pair it recovers happens to be connected by a two-edge provenance path, while the remaining conflicts are graph-disconnected, confirming that the INV3 gap is topological rather than a depth limitation of traversal. *Random-10* (B4) is an untargeted control: ten uniformly sampled active assertions per step, checked with the full three-layer machinery, matching IncrConsist's mean affected-set size. It reaches only 0.500 recall (CI $[0.37, 0.63]$), collapsing precisely in the edit-local layers (INV1 13/24, INV2 5/10) that the targeted methods saturate. Its 7/16 INV3 count is an artifact of the step-level detection criterion combined with state accumulation—injected conflicts remain active until repaired, so an untargeted sampler accrues chances over successive steps (Section 5.1)—and comes at 43% of IncrConsist's cost for a 30-point aggregate recall deficit. Together, B3 and B4 show that neither deeper traversal nor undirected sampling substitutes for the targeted affected-set computation with semantic expansion.

4.4. RQ4: Ablation Study

Figure 4 compares four ablation variants.

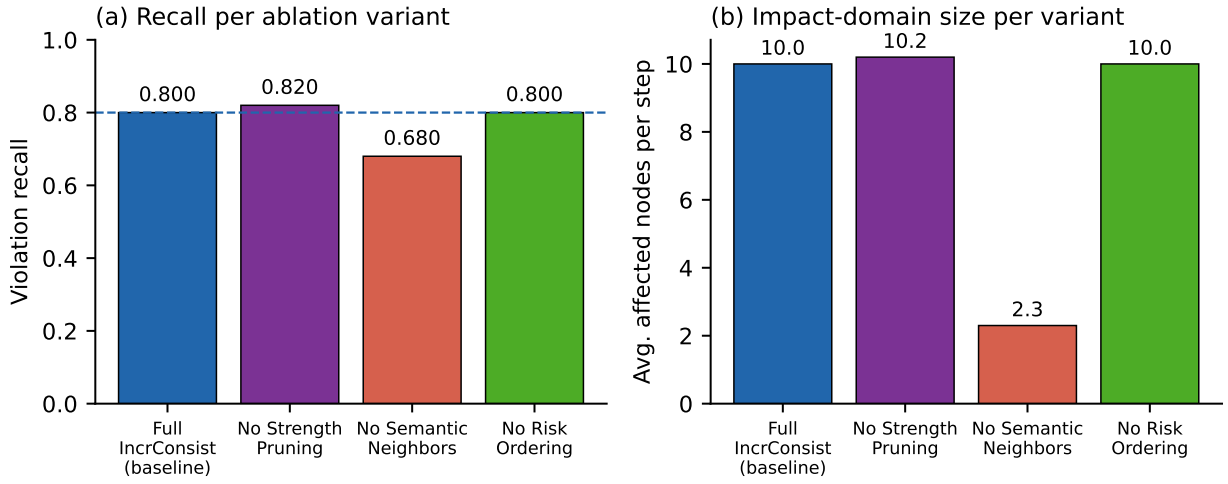

Figure 4: Ablation study: recall and average affected-set size for four IncrConsist variants. (**a**) Recall; blue dashed line marks baseline. (**b**) Affected-set size; note that No-Semantic reduces the affected-set size by 7.7 nodes but reduces recall by 12 points.

No-StrengthPrune (removing CTX-edge halting): the affected set grows by 0.2 nodes, and the recall marginally increases by 0.02 because additional CTX-reached nodes occasionally carry INV1 violations. We nevertheless retain CTX pruning: the 0.02 gain corresponds to a single violation and is within the resolution of a 50-violation gold set, whereas the pruning benefit scales with contextual-edge density, which is only 9% in this corpus and would be substantially higher in wikis with denser cross-referencing.

No-SemanticNeighbors (removing keyword-Jaccard and mutex expansion): the affected set shrinks from 10.0 to 2.3 nodes (−77%), but the recall drops from 0.80 to 0.68 (−12 points)—a loss attributable entirely to

the disappearance of INV3 detection. This confirms that semantic expansion is a *correctness mechanism*, not an efficiency component. Removing it also reduces the cost by a factor of 33, from 50.8 to 1.55 ms per step. Both figures come from the ablation run of `step04_ablation.py`, whose Full IncrConsist baseline is the same configuration and the same 200-step sequence as the main run of Table 5 but a separate process invocation; its baseline measures 50.8 ms per step against 49.8 ms in the main run, a 2% wall-clock difference attributable to timing jitter, since all detection counts and affected-set sizes are deterministic and identical across the two runs. Comparing within the ablation run, INV3 coverage is therefore priced at $50.8 - 1.55 \approx 49$ ms per edit for 12 recall points, still three orders of magnitude below full re-verification.

No-RiskOrdering (shuffling node processing order): recall is unchanged at 0.80 because all affected nodes are eventually checked. Risk ordering reduces MTTFD—useful in interactive settings—but does not affect aggregate detection rate.

4.5. Parameter Sensitivity

To assess robustness to the two parameters that shape the affected set, we replayed the identical 200-step edit sequence and gold labels under every combination of $d_{\max} \in \{1, 2, 3, 4\}$ and $k \in \{5, 10, 20\}$, with all other components fixed. Table 6 reports recall and mean affected-set size; both quantities are deterministic under the fixed seed and exactly reproducible from `step06_sensitivity.py`.

Table 6: Parameter sensitivity: violation recall (mean $|\Phi|$ in parentheses) over the $d_{\max} \times k$ grid on the 200-step main sequence. Bold marks the configuration used in the main experiment. The 95% Wilson half-width is ≈0.11–0.13 throughout ($n = 50$ gold violations).

	$k = 5$	$k = 10$	$k = 20$
$d_{\max} = 1$	0.78 (6.9)	0.80 (9.6)	0.92 (15.1)
$d_{\max} = 2$	0.78 (7.1)	0.80 (9.9)	0.92 (15.4)
$d_{\max} = 3$	0.78 (7.2)	**0.80 (10.0)**	0.92 (15.5)
$d_{\max} = 4$	0.78 (7.3)	0.80 (10.0)	0.92 (15.6)

Two patterns emerge. First, recall is completely insensitive to $d_{\max}$: every row is identical, confirming that INV2 violations are reached at depth 1 (the deleted parent's direct children) and that deeper HARD-chain traversal contributes to bounding, not detection. Second, recall is monotone in k, and per-layer inspection shows that the entire movement is confined to INV3: INV1 (24/24) and INV2 (10/10) are saturated at every grid point, while INV3 detections grow from 5/16 ($k = 5$) through 6/16 ($k = 10$) to 12/16 ($k = 20$). The mechanism is that injected direct-negation conflicts, which the mutex-term scan cannot see, are reachable only through the Jaccard neighborhood, so enlarging k recovers them. Doubling k from 10 to 20 therefore increases aggregate recall by 12 percentage points at approximately $1.9\times$ the per-step verification cost (and a 55% larger affected set), a favorable trade-off for deployments where INV3 coverage is the priority; we retain $k = 10$ as the main configuration for continuity and to keep the affected set below 1% of the knowledge base. This finding also refines the miss analysis of Section 5.1.

5. Discussion

5.1. Partial INV3 Recall and Knowledge-Base State Accumulation

The 37.5% INV3 recall reflects two compounding factors. First, as Delete operations accumulate over 200 steps, some "contrasting" assertions (those containing the complementary mutex term) are removed from the active pool. When a subsequent Modify injects "unsupervised" content, the mutex scan finds no active assertion containing "supervised" to pair against. This is not a defect of IncrConsist but a structural challenge for any maintenance-time checker: contradictions can be detected only if both sides of the conflict are simultaneously active. A practical mitigation is a lightweight semantic memory—a summary index of retracted assertions—so that new additions can be compared against recently deleted content.

Second, one of the five contradiction templates used for injection (self-attention vs. convolution) does not contain terms covered by the current mutex vocabulary, making it undetectable by lexical INV3 checks. This is a correctable configuration gap, not an algorithmic limitation.

The sensitivity analysis of Section 4.5 decomposes the ten INV3 misses quantitatively: six are recovered simply by enlarging the Jaccard neighborhood to $k = 20$ (a top-k truncation effect, since direct-negation conflicts are reachable only through semantic similarity, not the mutex scan), while the remaining four persist at every grid point and are attributable to the two structural factors above—deleted counterparts and the vocabulary gap.

5.2. Scalability of Semantic Expansion

The $O(n)$ mutex-aware scan is the primary source of IncrConsist's $3.4\times$ time growth from 50 to 500 pages. Replacing the linear scan with a pre-built inverted index (mutex term → assertion-ID list), maintained incrementally at $O(1)$ cost per Add/Delete operation, would reduce this

step to $O(\log n)$ retrieval. This is a straightforward engineering improvement that is left for future work.

5.3. Scope of INV3 Detection

The current INV3 checker is a lexical mechanism: it detects conflicts expressed through pre-defined mutex term groups or through the six direct-negation templates. It does not detect numerical inconsistencies ("accuracy 92%" vs. "accuracy 87%"), temporal contradictions, or paraphrase-level contradictions not captured by the mutex vocabulary. This scope limitation should not be overstated—the mutex approach handles the most common category of categorical label conflicts in AI-course wikis—but it should be acknowledged when generalizing these findings to other domain wikis. A natural direction for extending INV3 beyond the lexical mechanism is to compile numerical and temporal assertions into constraints that can be discharged by an SMT solver such as Z3 [35]; we leave this to future work.

5.4. Limitations

We consolidate the limitations of this study with their quantitative footprint. *(i) Partial INV3 recall.* IncrConsist detects 6 of 16 injected mutex-based global contradictions (37.5%, Wilson 95% CI $[0.18, 0.61]$). The sensitivity analysis (Section 4.5) shows six of the ten misses are a top-k truncation effect recoverable at $k = 20$ for $\approx$1.9$\times$ per-step cost, while four are structural: retracted counterparts (Section 5.1) and the vocabulary gap below. *(ii) Vocabulary-bounded INV3 scope.* The lexical mechanism covers five mutex groups and six negation templates; numerical, temporal, and paraphrase-level contradictions are out of scope (Section 5.3). *(iii) $O(n)$ mutex-scan bottleneck.* The scan drives the 3.4$\times$ per-step time growth over a 10$\times$ corpus increase and is addressable by an inverted index (Section 5.2). *(iv) Synthetic evaluation.* The corpus is generated to match the page granularity, edge density, and source-type distribution of a real WikiProvenance-Lite corpus, and controlled injection is necessary to measure recall against known labels; nevertheless, validation on organically evolved wikis with human-annotated violations remains future work. *(v) Single-machine timings.* Wall-clock figures are hardware-dependent; all detection counts and affected-set statistics, by contrast, are exactly reproducible from the released code under the fixed seed.

6. Conclusions

IncrConsist is an incremental consistency maintenance framework for LLM-maintained knowledge bases. By bounding verification to a compact affected set derived through strength-weighted provenance BFS and semantic neighborhood expansion, IncrConsist achieves a 1660$\times$ speedup over full re-verification and a 99.3% reduction in checked assertions, while maintaining 0.80 violation recall—14 percentage points above a naive 1-hop baseline. The entire gap over that baseline is attributable to the global semantic layer: the semantic expansion component is the critical enabler of INV3 detection, and without it, mutex-based contradictions between graph-disconnected assertions are completely missed. Scalability experiments confirm sub-linear time growth as the corpus grows tenfold, while full verification exceeds 90 seconds per step at 500 pages, making it impractical for continuous-update workflows. Limitations include partial recall of INV3 under accumulated deletions, a vocabulary-bounded scope of INV3, and an $O(n)$ mutex-scan bottleneck addressable by an inverted index. The broader implication is that the classical affected-set principles of Datalog materialization and truth maintenance can be transplanted to LLM-maintained text knowledge bases with two key extensions: strength-typed provenance propagation and lexical semantic conflict detection.

List of Abbreviations

LLM	Large Language Model
TMS	Truth Maintenance System
ATMS	Assumption-Based Truth Maintenance System
BFS	Breadth-First Search
INV1	Local Consistency Invariant (schema)
INV2	Reference Consistency Invariant (dependency chain)
INV3	Global Semantic Consistency Invariant (mutex-based)
DRed	Delete/Re-derive Algorithm
B/F	Backward/Forward Algorithm
RAG	Retrieval-Augmented Generation
KG	Knowledge Graph
AGM	Alchourrón–Gärdenfors–Makinson belief revision framework

Author Contributions

The author is solely responsible for the conceptualization, methodology, software, investigation, and writing—original draft. The author has read and agreed to the published version of the manuscript.

Availability of Data and Materials

All code, experimental scripts, and the synthetic-corpus generator are publicly available at https://github.com/aussie-bzhang/IncrConsist. The synthetic data set is fully

reproducible from `step01_make_synthetic.py` with `RANDOM_SEED=42`; no proprietary or third-party data sets are used. The repository also contains the plotting scripts that generate every figure in this manuscript directly from the released result files. All figures in this manuscript are original and were produced by the author with Matplotlib from the experimental results of this study. No figure, table, or graphical element is reproduced or adapted from any previously published work, and no third-party copyright permissions are required.

Consent for Publication

Not applicable. This study involves no human participants, and the manuscript contains no details, images, or videos relating to any individual.

Conflicts of Interest

The author declares no conflicts of interest.

Funding

The study did not receive any external funding.

Acknowledgments

The author thanks the editors and the anonymous reviewers of *Computing&AI Connect* for their careful reading, which materially improved this manuscript.

AI Declaration

The generative AI assistant Claude (Anthropic) was used in two phases: `claude-sonnet-4-6` for the original submission and `claude-opus-5` during revision, covering code scaffolding, the two supplementary experiment scripts, the figure-plotting scripts, literature search, and language and revision editing. Section 3.8 states the full scope of each phase. AI tools were not used for the experimental design of the main study, data generation, execution of any experiment, or computation of any reported number, and no LLM inference occurs within the evaluated pipeline. All experimental designs, results, interpretations and scientific conclusions are the sole responsibility of the author, who reviewed and validated all AI-assisted content.

References

[1] T. B. Brown et al., "Language models are few-shot learners," in *Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[2] W. X. Zhao et al., "A survey of large language models," *arXiv*, 2023, arXiv:2303.18223. [CrossRef]

[3] J. S. Park, J. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, "Generative agents: Interactive simulacra of human behavior," in *Proc. 36th Annu. ACM Symp. User Interface Softw. Technol. (UIST '23)*, San Francisco, CA, USA, Oct. 2023, pp. 1 – 22. [CrossRef]

[4] C. Packer et al., "MemGPT: Towards LLMs as operating systems," *arXiv*, 2023, arXiv:2310.08560. [CrossRef]

[5] A. Karpathy, "LLM Wiki: A Pattern for Building Personal Knowledge Bases Using LLMs," *GitHub Gist*, Apr. 2026. [Online]. Available: https://gist.github.com/karpathy/442a6bf555914893e9891c11519de94f

[6] A. Gupta, I. S. Mumick, and V. S. Subrahmanian, "Maintaining views incrementally," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, New York, NY, USA, 1993, pp. 157–166. [CrossRef]

[7] B. Motik, Y. Nenov, R. E. F. Piro, I. Horrocks, and D. Olteanu, "Parallel materialisation of Datalog programs in centralised, main-memory RDF systems," in *Proc. 28th AAAI Conf. Artif. Intell. (AAAI-14)*, Québec City, QC, Canada, Jul. 2014, pp. 129–137. [CrossRef]

[8] P. Plessers, O. De Troyer, and S. Casteleyn, "Understanding ontology evolution: A change detection approach," *J. Web Semantics*, vol. 5, no. 1, pp. 39–49, Mar. 2007. [View Online]

[9] J. Doyle, "A truth maintenance system," *Artif. Intell.*, vol. 12, no. 3, pp. 231–272, Nov. 1979. [CrossRef]

[10] J. de Kleer, "An assumption-based TMS," *Artif. Intell.*, vol. 28, no. 2, pp. 127–162, Feb. 1986. [CrossRef]

[11] J. A. Blakeley, P.-Å. Larson, and F. W. Tompa, "Efficiently updating materialized views," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Washington, DC, USA, 1986, pp. 61–71. [CrossRef]

[12] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*. Reading, MA, USA: Addison-Wesley, 1995. [Online]. Available: https://books.google.com.bh/books/about/Foundations_of_Databases.html?id=HN9QAAAAMAAJ&redir_esc=y

[13] G. Flouris, D. Manakanatas, H. Kondylakis, D. Plexousakis, and G. Antoniou, "Ontology change: Classification and survey," *Knowl. Eng. Rev.*, vol. 23, no. 2, pp. 117–152, Jun. 2008. [CrossRef]

[14] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: A collaboratively created graph database for structuring human knowledge," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Vancouver, Canada, 2008, pp. 1247–1250. [CrossRef]

[15] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: A core of semantic knowledge," in *Proc. 16th Int. Conf. World Wide Web (WWW)*, Banff, Canada, 2007, pp. 697–706. [CrossRef]

[16] F. Erxleben, M. Günther, M. Krötzsch, J. Mendez, and D. Vrandečić, "Introducing Wikidata to the linked data web," in *Proc. 13th Int. Semantic Web*

Conf. (ISWC), Riva del Garda, Italy, 2014, pp. 50–65. [CrossRef]

[17] D. Vrandečić and M. Krötzsch, "Wikidata: A free collaborative knowledgebase," *Commun. ACM*, vol. 57, no. 10, pp. 78–85, Oct. 2014. [CrossRef]

[18] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "DBpedia: A nucleus for a web of open data," in *Proc. 6th Int. Semantic Web Conf. (ISWC)*, Busan, Korea, Nov. 2007, pp. 722–735. [CrossRef]

[19] C. E. Alchourrón, P. Gärdenfors, and D. Makinson, "On the logic of theory change: Partial meet contraction and revision functions," *J. Symbolic Logic*, vol. 50, no. 2, pp. 510–530, Jun. 1985. [CrossRef]

[20] Z. Ji et al., "Survey of hallucination in natural language generation," *ACM Comput. Surv.*, vol. 55, no. 12, pp. 1–38, Mar. 2023. [CrossRef]

[21] S. Min et al., "FActScore: Fine-grained atomic evaluation of factual precision in long form text generation," in *Proc. 2023 Conf. Empirical Methods Natural Language Process. (EMNLP)*, Singapore, Dec. 2023, pp. 12076–12100. [CrossRef]

[22] X. Wang et al., "Self-consistency improves chain of thought reasoning in language models," in *Proc. 11th Int. Conf. Learn. Represent. (ICLR)*, Kigali, Rwanda, 202. [Online]. Available: https://dblp.dagstuhl.de/rec/conf/iclr/0002WSLCNCZ23.html

[23] S. Dhuliawala et al., "Chain-of-verification reduces hallucination in large language models," *arXiv*, 2023, arXiv:2309.11495. [CrossRef]

[24] L. Gao et al., "RARR: Researching and revising what language models say, using language models," in *Proc. 61st Annu. Meet. Assoc. Comput. Linguist. (ACL)*, Toronto, Canada, Jul. 2023, pp. 16477–16508. [CrossRef]

[25] A. Mallen, A. Asai, V. Zhong, R. Das, D. Khashabi, and H. Hajishirzi, "When not to trust language models: Investigating effectiveness of parametric and non-parametric memories," in *Proc. 61st Annu. Meet. Assoc. Comput. Linguist. (ACL)*, Toronto, Canada, Jul. 2023, pp. 9802–9822. [CrossRef]

[26] R. Xu et al., "Knowledge conflicts for LLMs: A survey," in *Proc. 2024 Conf. Empirical Methods Natural Language Process. (EMNLP)*, Miami, FL, USA, Nov. 2024, pp. 8541–8565. [CrossRef]

[27] N. Zhang et al., "A comprehensive study of knowledge editing for large language models," *arXiv*, 2024, arXiv:2401.01286. [CrossRef]

[28] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, and X. Wu, "Unifying large language models and knowledge graphs: A roadmap," *IEEE Trans. Knowl. Data Eng.*, vol. 36, no. 7, pp. 3580–3599, Jul. 2024. [CrossRef]

[29] R. Cohen, M. Geva, J. Berant, and A. Globerson, "Crawling the internal knowledge-base of language models," in *Findings Assoc. Comput. Linguist.: EACL 2023*, Dubrovnik, Croatia, 2023, pp. 1856–1869. [Online]. Available: https://aclanthology.org/2023.findings-eacl.139/

[30] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 33, 2020, pp. 9459–9474. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html

[31] D. Edge et al., "From local to global: A Graph RAG approach to query-focused summarization," *arXiv*, 2024, arXiv:2404.16130. [CrossRef]

[32] B. Gutiérrez, Y. Shu, Y. Gu, M. Yasunaga, and Y. Su, "HippoRAG: Neurobiologically inspired long-term memory for large language models," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 37, 2024, pp. 59532–59569. [Online]. Available: https://www.proceedings.com/079017-1902.html

[33] P. Chhikara, D. Khant, S. Aryan, T. Singh, and D. Yadav, "Mem0: Building production-ready AI agents with scalable long-term memory," *arXiv*, 2025, arXiv:2504.19413. [CrossRef]

[34] W. Xu, Z. Liang, K. Mei, H. Gao, J. Tan, and Y. Zhang, "A-MEM: Agentic memory for LLM agents," *arXiv*, 2025, arXiv:2502.12110. [CrossRef]

[35] L. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Proc. 14th Int. Conf. Tools Algorithms Constr. Anal. Syst. (TACAS)*, ser. Lecture Notes Comput. Sci., vol. 4963. Berlin, Germany: Springer, 2008, pp. 337–340. [CrossRef]

Disclaimer/Publisher's Note: The views expressed in this article are those of the author(s) and do not necessarily reflect the views of the publisher or editors. The publisher and editors assume no responsibility for any injury or damage resulting from the use of information contained herein.