FlowX.AI

TECHNICAL PAPER

MNEME · ACTIVE, GOVERNED MEMORY FOR AI AGENTS

A Knowledge Graph Agents Can Write To, Safely

MNEME, the FlowX.AI Ontology Layer: a provenance-tracked structured-memory graph that agents read, validate against, and contribute to, without corrupting the institutional source of truth.

Bogdan Răduță
Head of Research, FlowX.AI
bogdan.raduta@flowx.ai

FlowX.AI · Applied Research

September 2026 · Version 1.2

ABSTRACT

The 2026 consensus in enterprise AI is that retrieval is not enough: agents need structured, relational, persistent knowledge, and the knowledge graph has become the "beyond-RAG" layer of choice[1], with graph-based retrieval producing more comprehensive answers than vector retrieval on corpus-wide questions[2]. Yet across production enterprise platforms one assumption is nearly universal: the ontology is *read-only* context the agent consults, and keeping it correct is a separate human discipline. Research systems have begun to evolve ontologies automatically, gated by a performance signal[3]; that answers a different question from the one a regulated enterprise asks, which is who authorized each change. We present **MNEME**, the FlowX.AI Ontology Layer, which makes the opposite choice: the ontology is *active, governed memory the agent helps build*. Agents read, validate against, and write back to the same project-scoped knowledge graph. Each Concept and Relation carries *provenance* (pdm_import, agent_write, or manual) and *confidence* as first-class fields; agent writes are staged below a confidence bar, screened by conflict detection of three kinds, and quarantined out of canonical retrieval until promoted by reinforcement across runs or by human review. The institutional source of truth, imported one-way from the platform data model, is authoritative and read-only to agents, so a hallucination cannot become a regulatory taxonomy entry. MNEME runs on pluggable backends behind one contract-tested interface, retrieves by a hybrid of vector similarity and graph expansion, extends the same governance to access, history, and schema, and exposes a twelve-node Agent Builder toolkit and an MCP server. We describe the architecture, the write-back loop, the toolkit, an illustrative regulated taxonomy, and a retrieval evaluation on HotpotQA.

KEYWORDS ontology · knowledge graph · active memory · provenance · GraphRAG · entity resolution · conflict detection · structured memory

1 Introduction

An agent that answers from vector search alone is working from a pile of look-alike passages with no notion of how anything relates to anything else. The enterprise has noticed, and 2026 has been the year the knowledge graph moved from a data-team artifact to the centerpiece of agent design: graph-based retrieval produces more comprehensive answers than vector retrieval on questions that span a whole corpus[2], analysts name semantic layers and GraphRAG among the year's defining trends[1], and platform vendors now ship ontology-driven agent construction as a headline feature[4]. The premise is sound. Agents need persistent, relational, traceable knowledge, and an ontology supplies exactly the structure that flat retrieval lacks[5,6].

But look at how these systems treat the ontology and a shared limitation appears: it is a *read-only* oracle. The agent queries it, grounds an answer in it, and moves on; the graph itself is maintained by a separate human governance process, on a separate cadence, with the agent strictly a consumer. That is a strange division of labor for a system whose entire promise is that it learns. The agent that just resolved an ambiguous entity, classified a novel document, or discovered a relationship the graph did not contain has produced exactly the knowledge the ontology most needs, and then throws it away. The institutional memory does not improve from use; it improves only when a human gets around to editing it.

> *Production platforms make the ontology a read-only oracle the agent consults. The agent that just learned something the graph lacks is the agent best placed to extend it, and the one whose contribution we are most afraid to trust. Make the ontology active memory, and make the write safe.*
>
> MNEME · CORE THESIS

The reason production platforms keep the ontology read-only is not oversight; it is fear, and a reasonable one. In a regulated enterprise the ontology encodes product taxonomies, regulatory classifications, and risk hierarchies that must stay consistent across thousands of agent runs. Letting a stochastic model write to that structure sounds like inviting a hallucination into the source of truth. We present **MNEME**, the FlowX.AI Ontology Layer, which resolves the dilemma rather than ducking it: agents *do* write back, but every contribution is provenance-stamped, conflict-screened, staged below a confidence bar, quarantined from canonical retrieval, and promoted only by repeated corroboration or human review. The authoritative facts, imported from the platform data model, remain read-only to agents. The result is active memory that gets richer every run, with the regulatory blast radius of a read-only system.

1.1 Contributions

- A reframing of the enterprise ontology from a read-only oracle to active, governed memory that agents read, validate against, and safely extend.
- A provenance-first domain model in which every Concept and Relation carries its trust pedigree and confidence as first-class fields, making the graph auditable by construction.

- A safe write-back loop: deduplicate, conflict-detect, stage below a confidence bar, quarantine from canonical retrieval, and promote by reinforcement or human review.
- A pluggable-backend architecture (Postgres with pgvector by default, Neo4j opt-in, and a Postgres+Qdrant registration path) behind one contract-tested interface, with hybrid vector-plus-graph retrieval in a single call.
- Governance that extends past individual facts: identity-aware access control, bi-temporal history with as-of reads, staged schema-change proposals, and governed scenario overlays.
- A twelve-node Agent Builder toolkit that makes ontology operations first-class on the visual canvas, the agent patterns that compose them, and a second access surface for external agents over MCP.

2 Read-Only Knowledge, and Its Limits

2.1 Three kinds of agent knowledge

Enterprise agents draw on three distinct stores, and conflating them is the source of much confusion. *Retrieval* (RAG over a vector store[7]) supplies unstructured content by similarity, excellent for "what does this document say" and blind to structure. *Operational data* (databases, the platform data model) supplies the authoritative current state of entities, excellent for "what is this customer's balance" and not designed for semantic reasoning. Between them sits the *ontology*: a graph of concepts and relations that says how the business's world is organized, what a Mortgage is, that it is a Secured Loan, that a Secured Loan inherits certain required attributes. Graph retrieval over that structure is what enables multi-hop reasoning and explainable paths that flat similarity cannot produce[6].

2.2 The read-only assumption, and what it costs

The market has converged on building that middle layer, but on treating it as immutable from the agent's side. The cost is twofold. First, the ontology decays relative to reality: new products, new regulations, and newly-resolved entity variants accumulate in agent runs and are never written back, so the graph drifts out of date until a human governance cycle catches up. Second, the knowledge an agent is uniquely positioned to capture, that two records are the same entity, that a document introduces a concept the taxonomy lacks, is exactly the knowledge that evaporates at the end of the run. A system sold as institutional memory does not actually remember anything its agents learn.

2.3 Why write-back is hard, and what makes it safe

The reason production platforms do not let agents write is that the obvious implementation is dangerous: if an agent can mint a concept, a hallucination becomes a taxonomy entry, and in a regulated setting that is a compliance incident waiting to propagate. Safe write-back is therefore not a feature toggle but an architecture: contributions must be attributable (provenance), screened against the existing structure (conflict detection), held apart from authoritative knowledge until they earn their place (staging and promotion), and unable to affect canonical answers while pending (quarantine). MNEME is the design of that architecture. Table 1 contrasts the read-only norm, eval-gated self-evolution (Section 2.4), and active memory.

2.4 Self-evolving ontologies

Agent memory has become an active research area in its own right[8], and one recent system attacks the read-only assumption from the research side. EvoOntology[3] maintains a versioned ontology state for data agents in three layers: a content layer of Terms, Mappings, Constraints, and Evidence; a schema layer that fixes the admissible node fields, relation types, and reference patterns; and a tool layer that exposes the ontology as an MCP server with browse and resolve tools and a compact session manifest. An evolution agent refines that state from historical agent trajectories through attribution-guided typed edits, and a candidate version is kept only when a backbone-conditional paired evaluation against its parent, on the same validation set and budget, shows an improvement above a margin. Because the gate is conditioned on the LLM backbone, a change is accepted relative to a specific model. The authors report gains on three data-agent benchmarks: DDR-Bench, InsightBench, and BIRD.

The two designs govern different things. EvoOntology governs *versions* with a *performance* signal, in training-time rounds driven by an evolution agent, and a version is accepted relative to the model that reads it. MNEME governs *facts* with *trust* signals (provenance, conflict screening, staging, quarantine, earned promotion), per agent run, with human stewardship, and a fact is accepted on authority, independent of which model reads it. We found no provenance-based permission model, authoritative read-only core, or steward review path in EvoOntology, and those are the properties a regulated deployment needs. The approaches are complementary rather than competing: an evaluation gate measures whether a change helps; it does not establish who authorized it or whether it is safe for a regulated taxonomy. Section 10.1 sketches how the two could be combined.

Table 1. Read-only knowledge graph, eval-gated self-evolving ontology, and active, governed memory.

Property	Read-only KG (the norm)	Self-evolving (EvoOntology)	MNEME (active memory)
Agent role	Consumer only	Task agent reads; a separate evolution agent writes	Consumer and contributor
Improves from use?	No (human edits only)	Yes (from historical trajectories)	Yes (staged, promoted)
Unit of change	Manual edit	Candidate ontology version	Individual fact; schema proposal
Acceptance signal	Human judgment	Task-score gain over the parent version, for a given LLM backbone	Corroboration or steward review
Provenance	Often absent or metadata	Evidence records; no provenance-based permissions described	First-class on every fact; drives permissions
Hallucination risk to canon	N/A (no writes)	Screened by data probes and the evaluation gate	Quarantined; cannot reach canon
Conflict handling	Manual	None described	Detected and logged pre-stage
Source of truth	Separate, manual	No authoritative read-only core described	PDM imported, read-only to agents
Human review	Is the process	Not described; acceptance is automatic	Review queue; schema-reviewer role

3 The MNEME Architecture

MNEME sits as a structured-memory layer between unstructured retrieval and operational data. It does not replace either: the knowledge base still holds document chunks, and the platform data model still holds authoritative entity state. The ontology is the connective structure agents reason over and contribute to. Figure 1 shows the three layers and the flows between them.

FIG. 1 · THE STRUCTURED-MEMORY LAYER BETWEEN RETRIEVAL AND OPERATIONAL DATA

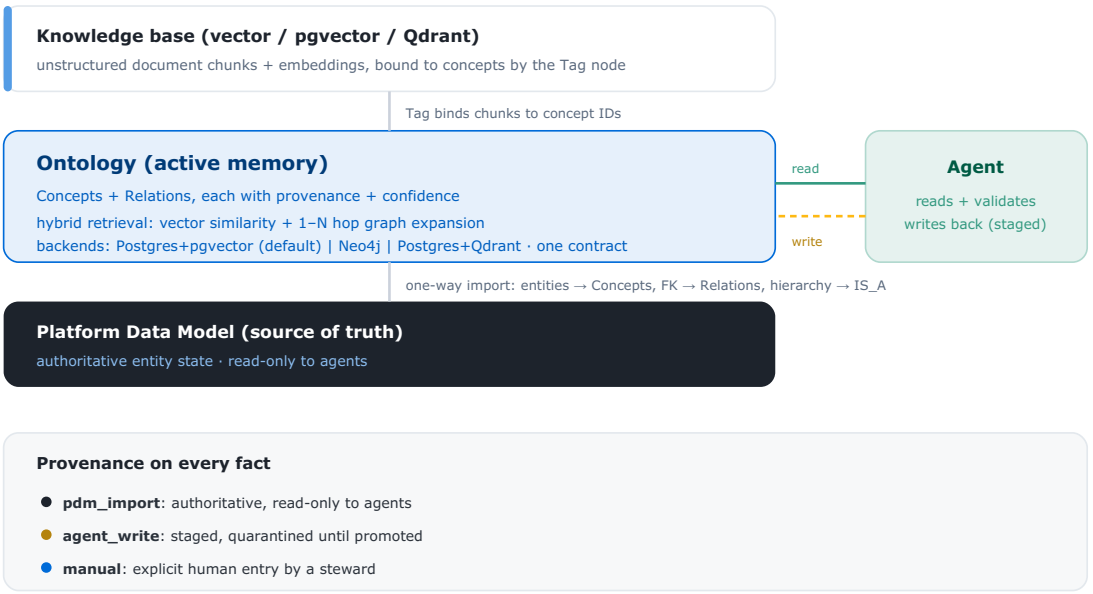


Figure 1. The ontology is the middle layer: the knowledge base supplies content (bound to concepts by the Tag node), the platform data model supplies authoritative state (imported one-way, read-only to agents), and the agent both reads from and writes back to the ontology. Every fact carries provenance, which is what makes the read and write semantics safe to differentiate.

3.1 A provenance-first domain model

The unit of knowledge is a Concept (with a label, synonyms, an IS_A position, and an attribute schema) or a Relation between concepts. What makes the model unusual is that *provenance* and *confidence* are not metadata bolted on the side but first-class fields every fact must carry. A fact is one of three provenances, in the spirit of the W3C provenance model[9]: *pdm_import* (drawn from the authoritative data model, and read-only to agents), *agent_write* (proposed by an agent, and quarantined until promoted), or *manual* (entered explicitly by a human steward, or loaded by a governed import from an authoritative external source such as a regulator corpus or a SKOS/OWL taxonomy). A promoted agent contribution keeps its *agent_write* provenance, so "proposed by an agent, later validated" stays distinguishable from human or imported entry. Write permissions follow from provenance, so the system can let agents enrich the graph while guaranteeing they cannot alter the authoritative core: an edit to a pdm_import concept is refused. Read access is governed separately, by the access-control layer of Section 3.4. An auditor can reconstruct, for any concept, who asserted it,

when, and with what authority, and with the bi-temporal model of Section 3.5, what the graph said about it at any earlier point.

The authoritative core can be seeded along four import paths, each of which writes through the same provenance model rather than around it: the one-way platform data model importer, a regulatory-corpus importer that maps a Framework, Regulation, Chapter, Article, and Provision hierarchy onto IS_A chains, a SKOS/OWL importer (shipped with a small FIBO-based seed), and a structural importer for Apache Ossie/OSI semantic-model metadata. Appendix C summarizes what each maps and what it deliberately leaves out.

3.2 Pluggable backends, one contract

Ontologies differ in shape. A product catalog is semantic-heavy and well served by vector similarity over Postgres with pgvector, the default; a regulatory hierarchy is graph-heavy and rewards the native multi-hop traversal of Neo4j 5, which is opt-in. A third registration path pairs Postgres with a Qdrant vector index and inherits the Postgres implementation for everything else. All of them sit behind a single *OntologyStore* interface, 51 abstract methods in the current contract, whose behavioral parity is enforced by one contract-test suite run against both real Postgres and real Neo4j, not only against an in-memory stand-in, so a team chooses the backend that fits the workload rather than the one the platform happened to pick. Backend choice is fixed per ontology at creation; there is no migration tooling between backends. Embeddings are computed locally with fastembed and the all-MiniLM-L6-v2 sentence-transformer model (384 dimensions)[10,11], pluggable per ontology, so the layer runs in air-gapped deployments with no outbound calls.

3.3 Hybrid retrieval

A query against the ontology is not a pure nearest-neighbor lookup. MNEME combines vector similarity with graph expansion of one to N hops in a single call, and ranks results by a weighted blend of similarity and popularity (by default 0.7 × similarity + 0.3 × normalized popularity, configurable per deployment and per call, since a regulator corpus wants similarity to dominate while a product catalog may want popularity to count more), so an agent grounds its answer in context that is both semantically and relationally relevant. This is graph-augmented retrieval in the general sense, a similarity seed expanded along explicit relations, rather than the community-summary method evaluated for GraphRAG[2]: the agent does not merely find the closest passage, it finds the connected neighborhood of concepts that bear on the question, and can cite the path. Retrieve responses also carry the induced relations among the returned concepts, so the path is in the response rather than a second query away.

3.4 Access control

Provenance decides who may *change* a fact; a regulated deployment also has to decide who may *see* one. MNEME supports optional OIDC/JWT authentication that validates issuer, audience, expiry, signature, subject, and project assignment, with the key set loadable from a local file for air-gapped installations. Reserved roles govern ontology administration, schema review, action invocation, and MCP writes. On top of roles sit persisted allow/deny rules, each binding a role to an ontology category or an individual concept. A matching deny always wins, whatever the rule order; a concept that allow rules target is visible only to principals holding one of those roles, and a concept no rule targets stays visible. A hidden concept answers 404 rather than 403 so its existence is not disclosed,

and a relation is visible only when both of its endpoints are. Authentication is off by default for compatibility, and a production deployment enables it explicitly.

3.5 Bi-temporal facts

Every Concept and Relation carries two independent timelines, following the bi-temporal pattern used in recent agent-memory systems[12]: a world timeline (when the fact was true) and a system timeline (when the store recorded it and when it retired it). Supersession is handled by invalidation, never deletion: the prior version is archived and marked retired, so an *as-of* query returns exactly what an agent would have seen at that time. Retrieval, concept and relation listing, and graph expansion all accept an as-of time, and a query without one behaves exactly as before. Two write paths use this directly. A promotion that contradicts an existing fact invalidates the old one and stamps the new one as valid from the promotion, and a regulatory re-import that changes a provision's text versions it rather than overwriting it. This is what turns the audit claim of Section 3.1 from "who asserted it" into "what did the graph say, and when." The two-backend contract paid for itself here: an early implementation deleted the superseded record on invalidation, which is harmless on Postgres but, on Neo4j, silently deletes every relationship attached to the node. Running the same contract suite against both real backends caught the divergence before it shipped.

4 Active Memory: the Safe Write-Back Loop

The defining capability is governed, per-fact agent write-back: an agent can contribute to the graph without endangering it, because every contribution carries provenance that determines what it may touch, is screened for conflicts, and is quarantined until it earns promotion. We are not aware of a production enterprise ontology platform that offers write-back in this form; research systems that evolve an ontology automatically take a different route, discussed in Section 2.4. Write-back is a pipeline, not a single operation, and every stage exists to make a contribution safe. Figure 2 shows it for facts; Section 4.4 applies the same discipline to the schema itself, a second write path that governs structure rather than facts.

FIG. 2 · HOW AN AGENT CONTRIBUTION BECOMES CANONICAL KNOWLEDGE

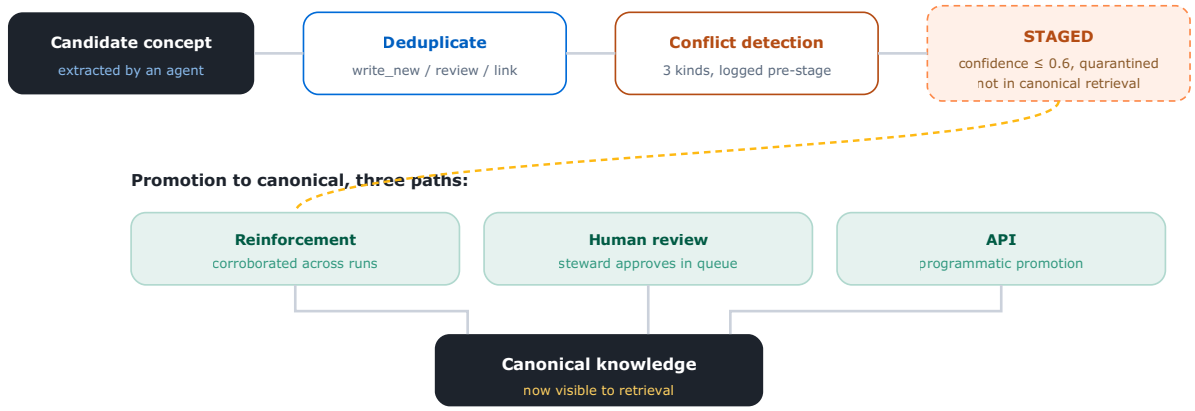


Figure 2. A candidate concept is first deduplicated (so an agent rediscovering an existing concept under a new phrasing links rather than duplicates), then screened for three kinds of conflict, then staged below a confidence bar where it is durable but invisible to canonical retrieval. It becomes canonical only by reinforcement across runs, explicit human review in the pending queue, or a programmatic promotion. A hallucination that is never corroborated and never approved stays in staging, where it cannot affect a canonical answer, until a reviewer rejects it.

4.1 Deduplicate before writing

The most common failure of agent write-back is not a hallucination but a near-duplicate: an LLM rediscovers *Personal Loan* as *Retail Personal Loan* and proposes it as new. The Deduplicate step runs a pre-flight similarity check and returns a recommended action, write a new concept, route for review, or link to the existing one, so a write happens only when the candidate genuinely adds structure. This keeps the graph from bloating with synonyms, which is the quiet way an agent-writable ontology would otherwise rot.

4.2 Conflict detection and staging

A surviving candidate is screened for three kinds of conflict against the existing structure, a label clash, a category mismatch, and a relation that contradicts an existing one, each detected and written to a queryable log before anything is staged. A write that raises a conflict is automatically flagged for human review, and a flagged write cannot take the reinforcement fast path to canonical status; only an explicit promotion clears it. The contribution is then staged at low confidence: durable, at-

tributable, and visible in a pending-writes queue, but excluded from canonical retrieval, so it cannot influence a production answer while it waits. This is the same quarantine logic that hallucination containment applies to outputs, here applied to memory: the unverified thing is kept where it can do no harm until it is verified.

4.3 Promotion earns canonical status

A staged contribution becomes canonical only by earning it, along one of three paths. *Reinforcement* promotes a concept that is independently rediscovered across enough runs, on the principle that corroboration is evidence. *Human review* lets a domain steward approve or reject from the pending queue, the path a regulated taxonomy change will usually take. *Programmatic promotion* via the API supports controlled bulk workflows. A contribution that is never corroborated and never approved never becomes canonical; it stays in staging until a reviewer rejects it. Validation with IS_A inheritance runs throughout, so a promoted concept must satisfy the required attributes of its parents, halting any agent output that violates the schema before it reaches a downstream system.

Validation is itself pluggable. Each check is a *ValidationRule* in a registry, with four built-ins (required attributes, attribute enums, required relations, and concept-reference existence), so a regulator pack can register its own rules without touching the core service. Rules can be bound to a category and are inherited by its descendants along IS_A, and binding or unbinding a rule is a schema change (Section 4.4) rather than a code change. Relation types can declare inverse, symmetric, and transitive semantics: inverse edges are projected at read time and never stored, and transitive reads are bounded by the configured hop limit.

4.4 Schema changes are staged too

Staging facts is not enough if the schema those facts obey can change the moment someone calls an API. A new required attribute, a new enum constraint, or a new relation type can invalidate every existing write against a category at once, so schema changes take the same governed route as facts. A *schema-change proposal* is a named batch of operations (add or rename a category, add a relation type, set required attributes, set an attribute enum, bind or unbind a validation rule) that moves through *draft, in review, approved, merged*, or is rejected. Approval requires the reserved schema-reviewer role of Section 3.4. Every ontology carries a schema version; a merge applies only if the proposal's base version still matches, so a stale proposal is refused with the current version rather than silently overwriting a concurrent change. Before anything is written, the whole batch passes a read-only preview, so a malformed operation fails the batch with nothing applied. That preview is not a transaction: an unexpected failure partway through applying a valid batch is not rolled back across operations (Section 10).

4.5 Scenario overlays

Governed *scenario overlays* let stewards and agents ask "what if" before proposing anything. An overlay is a persisted set of changes read overlay-first, then base, not a database fork. Retrieval, validation, listing, expansion, and relation reads all accept a scenario identifier, and access-control policies are evaluated both before and after the overlay is composed. A scenario can be created, atomically replaced, compared with the base, discarded, or promoted. Promotion never writes canonical facts directly: schema changes in the overlay become draft schema-change proposals (Section 4.4), and fact changes become staged memory writes subject to the usual conflict detection and review (Sections 4.2 and 4.3), so an overlay cannot bypass the governance it is promoted through.

5 The Twelve-Node Toolkit

On the Agent Builder canvas, MNEME appears as a dedicated Ontology palette of twelve nodes (Table 2), each mapping to one operation an agent performs against structured memory. The original nine cover grounding, traversal, validation, and staged memory; Act, Extract, and Intent add governed action preparation, document extraction, and grounded intent. They are drop-in: each writes well-known keys to the workflow state, ready for prompt injection or downstream branching, with no glue code.

Table 2. The Ontology node palette on the Agent Builder canvas.

Node	What it does	Common use
Retrieve	Hybrid semantic + graph retrieval; top-K concepts with similarity, popularity, optional 1–N hop expansion	Grounding a prompt with relevant ontology context
Classify	Single-label classification: text to top concept + confidence	Flow routing and branching on a matched concept
Describe	Lookup by ID with full grounding context (label, synonyms, IS_A chain, attribute schema)	Deterministic grounding when the concept is known
Expand	Walk the relation graph from a concept (for example all IS_A descendants to N hops)	"Find all loan products that inherit from Secured Loan"
Tag	Tag a longer text with concept IDs by chunking and per-chunk semantic resolution	Document ingestion: concept tags as KB metadata
Deduplicate	Pre-flight similarity check before a write; returns write_new / review / link_to_existing	Prevent memory-write spam from rephrased rediscoveries
Validate	Validate a structured payload against ontology rules with IS_A inheritance	Halt or warn on agent output that violates schema
Memory Write	Stage a candidate concept or relation; conflicts detected and logged	Agents propose new structure; humans review
Memory (all-in-one)	Multi-mode node combining retrieve / validate / memory-write in one config	Compact flows where one node runs the whole loop
Act	Prepare a signed, time-limited action descriptor for a concept's bound process after pre-validation; never invokes the target itself	Governed hand-off: "start this claims workflow"
Extract	Run a document-extraction job over chunks and wait for its report; results land in staging, never in canon	Building graph structure from ingested documents
Intent	Turn free text into grounded intent: mentions resolved through retrieval, timeframe, category scope, optional action hint	Structured, auditable request parsing

5.1 The patterns that compose them

A handful of compositions cover most real flows. **Retrieval-grounded answer:** Retrieve, then answer with the returned context injected into the prompt. **Classify then describe:** Classify the input to a concept and, on a match, Describe it for clean grounding, or ask a clarifying question when nothing crosses the similarity threshold. **Multi-hop reasoning:** Classify to a starting concept, Expand its IS_A family in one call, and let the model synthesize over the family, the Neo4j differentiator made tangible. **Validate before commit:** have the model emit structured output, then Validate it against the schema so an invalid payload halts before any downstream effect. **Contribute with rails:** extract a

candidate, Deduplicate it, and only Memory-Write the genuinely new, staged for review. **Ground, then act:** Intent parses a free-text request, Validate checks the resulting payload, and Act requests a descriptor only once both pass. The same twelve nodes serve a read-only grounding flow and a full read-reason-validate-contribute-act loop; the difference is which nodes the author places.

5.2 Grounding an action, not just an answer

The read-write loop otherwise stops at information: an agent grounds an answer or stages a fact, and nothing downstream happens. An *action binding* lets a concept or category declare a link to a governed FlowX action: a BPMN business process, an Agent Builder workflow, or an integration workflow. Concepts that are IS_A descendants inherit the binding. The Act node asks the ontology to prepare the action. The service confirms the binding applies to the concept, runs the configured pre-validation, maps concept attributes to process inputs, and returns a descriptor signed with HMAC-SHA256 that expires after a set time. If validation fails strictly, no descriptor is issued, and with no signing key configured the endpoint refuses rather than falling back to an unsigned descriptor. The ontology service never executes anything itself; the downstream engine verifies the signature and runs the process. A binding to an integration workflow is tied to a specific build, so a binding that has gone stale after a republish is reported as such and refused before any descriptor is signed. The Intent node is the front half of the same path. A provider-neutral chat-completions endpoint proposes mentions, a timeframe, a category scope, and whether an action is wanted; every mention is then grounded through ontology retrieval. Low-confidence mentions stay explicitly unresolved rather than being coerced into a concept, and an action hint is drawn only from bindings that apply to the grounded concepts and that the caller's roles permit.

5.3 A second access surface: MCP

The canvas is not the only way in. MNEME also exposes the ontology as an MCP server for external agents, with eight read tools that mirror the grounding nodes (list, retrieve, validate, classify, describe, expand, tag, deduplicate) and the same semantics as the REST and node paths. Two write tools, staging a memory write and listing pending writes, are off by default. When enabled they additionally require a writer or administrator role, and a tool the caller may not use is absent from the tool listing rather than failing on call. Even enabled, MCP can only *stage*: promotion, review flags, and conflict resolution remain on the governed REST and Designer paths, so an external agent is held to the same write-back discipline as one on the canvas.

6 The Document-to-Ontology Bridge

A recurring question is how unstructured documents map onto the ontology. The answer is two layers, two indexes, one binding, and it is deliberately not "put the documents in the graph." The knowledge base holds the document chunks and their embeddings; the ontology holds the concepts; and the Tag node is the binding, resolving a document's text against the ontology and attaching the resulting concept IDs as metadata on the stored chunks. A later query can then filter the knowledge base by ontology concept, "find chunks tagged under this regulatory category," rather than by raw similarity alone, so the document store becomes navigable by meaning. Where a document genuinely introduces a new concept, a gated Memory Write stages it for review. The ontology is never the document store; it is the structural index that makes the document store comprehensible, and the Tag node keeps the two in registration without an agent having to manage the link by hand.

By default the Tag node is pure-semantic, chunk then resolve then threshold, with no model call, which makes it fast, free, and deterministic. Where higher-quality entity extraction is needed, an LLM extraction step can precede it, emitting a list of mentions that Tag resolves one by one; the node accepts that list directly, so the upgrade is a one-node composition rather than a rebuild.

For building structure from documents at scale, the Extract node runs a full extraction job instead. The job consumes the document-parsing pipeline's page-level output (MNEME does no OCR or parsing of its own) and proceeds in five steps. It grounds each chunk against the ontology, extracts candidate concepts and relations with a pluggable extractor (the reference extractor calls any OpenAI-compatible endpoint, including a locally hosted one), grounds the candidates, deduplicates them, and stages them. Every result goes through the write-back loop of Section 4; nothing extracted is written to canon directly. A candidate close to an existing concept becomes an enrichment of that concept rather than a new one. Relation endpoints that cannot be resolved are skipped rather than guessed. The same fact found in a second document reinforces the first staged write and adds its page-level source pointer, so a promoted fact arrives with its citations attached. Jobs are persisted with their state, counters, candidate hashes, principal, and report. A restart marks unfinished jobs as interrupted without losing progress, and an explicit rerun reuses candidate hashes so a retry does not stage duplicates. Per-job and daily caps on staged writes end a job with a truncated but accurate report instead of flooding the review queue, and a dry run reports what would be staged without staging it.

7 Illustrative Application

Consider the loan-product taxonomy of an illustrative regulated bank, Meridian (a fictional example used for continuity across this series). The ontology encodes the hierarchy, Mortgage is a Secured Loan is a Loan, with required attributes inherited along the IS_A chain, and is backed by Neo4j because the reasoning is graph-heavy.

In day-to-day grounding, an underwriting agent Classifies an incoming application to a product concept, Expands the concept's governing rules in a single multi-hop call, and Validates that the proposed terms inherit the required attributes of the parent type; a violation halts the flow before terms reach a downstream system. None of this writes to the graph, it is read-only grounding, and it already exceeds what flat retrieval could do, because the agent reasons over the product family, not over look-alike paragraphs.

The active-memory difference shows when a new regulation is published. An agent extracts the new classification it introduces, runs Deduplicate (which confirms it is genuinely new, not a rephrasing of an existing rule), and issues a Memory Write. Conflict detection checks the candidate against the existing hierarchy and finds no contradiction, so it is staged at low confidence, attributed to *agent_write*, and held in the pending-writes queue, invisible to production grounding. A compliance steward reviews it there, confirms the placement in the taxonomy, and promotes it; only now does it become canonical and start grounding answers, carrying its provenance for any future audit. The institutional taxonomy improved from an agent run, on a regulated artifact, without a single moment at which an unreviewed model output could have driven a lending decision. That is the whole point of making memory active and governed at the same time.

8 Empirical Evaluation

Section 3.3's hybrid-retrieval claim describes a general property attributed to graph-augmented retrieval, that following explicit relations recovers evidence a similarity search misses, not a number MNEME itself has published. To check whether that property holds for the specific retrieval mechanism this paper describes (a similarity-seeded entry point plus a one-hop expansion over entity-mention links, Section 3.3), we built a standalone reference implementation and evaluated it on a public multi-hop QA benchmark. The implementation is a minimal reproduction of the mechanism, not the FlowX.AI Platform's production Postgres/Neo4j retrieval path (which uses learned embeddings and a persisted, corpus-wide graph rather than a per-query mention graph); it exists to test the claim, and the code and test suite are released as supplementary material.

8.1 Dataset and setup

We use the HotpotQA distractor validation set[13]: 7,405 questions, each with ten candidate Wikipedia paragraphs (two containing the supporting facts, eight textually plausible distractors) and human-annotated supporting-fact titles. We compare two retrieval modes with an identical entry point. *Lexical-only (BM25)* ranks the ten candidates by BM25 similarity to the question and takes the top-k. BM25 stands in for the similarity entry point: the production path uses dense embeddings, but both are single-hop scorers that see a paragraph only if it is textually close to the question, which is the blind spot the graph hop targets. *Hybrid* takes the top-1 BM25 candidate as an entry point, builds a per-question graph with an edge between any two candidates where one paragraph's text mentions the other's title, and returns the entry point plus its graph neighbors (falling back to the next-best BM25 candidates if the budget k is not filled by neighbors alone). A retrieval is scored as a hit only if both gold supporting titles appear in the returned set (exact-recall@k), evaluated separately for HotpotQA's two reasoning types: *bridge* questions, which require traversing from one entity to a second, textually unrelated one, and *comparison* questions, which typically name both entities directly.

8.2 Results

Table 3 reports exact-recall@k over the full validation set. On bridge questions, the reasoning type the graph hop specifically targets, hybrid retrieval roughly doubles lexical-only recall at @2 (55.5% vs. 27.4%) and gives a smaller but still substantial lift at @4 (74.5% vs. 51.1%), directionally consistent with the claim that graph traversal recovers multi-hop evidence a similarity search alone misses. On comparison questions, where both entities are typically already named in the question and similarity search alone can find both paragraphs directly, the graph hop does not help and is slightly negative (@2: 15.5% vs. 20.4%): the entry-point-plus-neighbor heuristic can spend its budget on a graph neighbor of the top BM25 hit instead of the second named entity, when the two compared entities do not mention each other. Both results are the honest shape of the mechanism: it wins specifically where similarity-only retrieval falls short, and does not universally dominate.

Table 3. Exact-recall@k (both gold supporting titles retrieved), HotpotQA distractor validation, n=7,405.

Question type	n	BM25-only@2	Hybrid@2	BM25-only@4	Hybrid@4
Bridge (multi-hop)	5,918	27.4%	**55.5%**	51.1%	**74.5%**
Comparison	1,487	**20.4%**	15.5%	46.3%	46.1%
Overall	7,405	26.0%	**47.5%**	50.1%	**68.8%**

8.3 What this evaluation does and does not show

This experiment isolates the retrieval mechanism of Section 3.3 in a standalone harness built for this test; it does not exercise the write-back pipeline of Section 4, the provenance model, or the real Postgres/Neo4j backends, none of which HotpotQA is designed to evaluate. The result supports a narrower claim than "hybrid retrieval is better": it shows that similarity-seeded graph expansion helps precisely on the multi-hop reasoning type it targets, and can slightly hurt on a reasoning type it does not, which is the kind of result an honest evaluation should produce rather than a uniform win. Benchmarking the write-back loop (promotion accuracy, false-promotion rate under the confidence-staging policy of Section 4) against a labeled corroboration dataset remains future work. The reference implementation, the HotpotQA evaluation script, and a pytest suite that reproduces Table 3 are released as supplementary material.

9 Discussion

The market is right that agents need a knowledge graph; the question MNEME answers differently is whether the agent is allowed to help build it. Read-only graphs, including the ontology-driven agent offerings shipping in 2026[4], treat maintenance as a human discipline orthogonal to agent operation, which leaves the graph perpetually behind the agents that use it. Eval-gated evolution[3] is a different answer to the same problem: an automated process improves the ontology between deployments, and a change is accepted because it measurably helps the task, not because anyone authorized it. Active memory closes the gap per fact and per run, and the reason it can be done safely in a regulated enterprise is the same reason the rest of this platform is safe: the unverified thing is quarantined and must earn promotion. Provenance is the mechanism, and it is the same artifact that AI governance frameworks ask for[14,15,16], a record of who asserted what, with what authority, when.

MNEME also composes with the rest of the series rather than standing alone. Its nodes are RAILS nodes[17], so an ontology operation is a typed, deterministic step in a compiled graph. Its quarantine of low-confidence writes is the memory-side analogue of hallucination containment. Its provenance and pending-review queue are governable by the same policy engine that evidences every other agent action. Structured memory is not a separate product bolted to the platform; it is the layer that gives the agents a shared, durable, auditable model of the enterprise's world, and lets them improve it without being trusted blindly.

10 Limitations

- **The empirical evaluation covers retrieval, not write-back.** Section 8 measures the similarity-seeded graph-hop mechanism alone, with a lexical (BM25) entry point, on a public multi-hop benchmark with a per-question mention graph. It does not measure promotion accuracy, false-promotion rate, or behavior against the real Postgres/Neo4j/Qdrant backends, and the graph-hop heuristic itself is a net negative on comparison-type questions. The additions of Sections 3.4 to 6 are described from the shipped system, and none of them is exercised by Section 8. Bi-temporal history, schema proposals, scenario overlays, action binding, extraction, and bound validation rules have contract tests that run against the in-memory store and, when enabled, against real Postgres, Neo4j, and Postgres+Qdrant; access control has a policy contract test plus authenticated end-to-end API tests; the MCP surface is covered by in-process client tests, including parity with REST.
- **Promotion policy is a judgment.** Reinforcement thresholds and confidence bars trade richness against caution; set them loose and the graph fills with weakly-corroborated concepts, set them tight and useful additions languish in staging.
- **The review queue has no retention policy.** A staged write that is never corroborated stays in staging until a reviewer rejects it; nothing expires it, so without steward capacity the pending queue grows without bound. The per-job and daily extraction caps limit how fast it grows, but they are not a retention policy.
- **Deduplication is similarity-bounded.** The pre-flight check catches near-duplicates by embedding similarity; a genuinely novel phrasing of an existing concept that is semantically distant can still slip through as a new entry until review catches it.
- **One-way PDM import.** The data model is the source of truth and the import is one-directional by design; the ontology does not write back to the PDM, so corrections discovered in the graph must flow to operational systems by a separate path.
- **Backend parity is behavioral, not identical.** Postgres and Neo4j are contract-tested for equivalent behavior, but deep multi-hop traversal is capped more tightly on Postgres (three hops, with truncation reported); very graph-heavy ontologies belong on Neo4j. The Postgres+Qdrant path inherits the Postgres implementation for everything except the vector index. Backend choice is fixed per ontology, with no migration tooling between backends.
- **Tag quality depends on chunking.** The default semantic Tag works best on short, entity-like text; long or discursive passages benefit from an LLM extraction step in front, which reintroduces a model call the pure-semantic path avoids. Chunks beyond the configured cap are dropped without error.
- **Schema batches are validated, not transactional.** A schema-change proposal is fully previewed before any operation is applied, so a malformed batch writes nothing, but an unexpected failure partway through applying a valid batch is not rolled back across operations.
- **Process renames are not reconciled.** An action binding to a business process is invoked by process name, with the stable definition identifier kept for audit; if the process is renamed, the binding must be updated by hand.

- **History is API-only.** Bi-temporal facts and as-of reads are available through the API and the nodes; the Designer has no history view.
- **Authentication is opt-in.** OIDC enforcement and ACLs are off by default for compatibility; a deployment that does not enable them has no access boundary beyond the network.
- **Declared non-goals.** MNEME does not federate queries to external semantic systems at query time; external structure enters through governed import paths with dry-run and stable diffs. It does not execute metrics or KPIs; imported metrics remain descriptive concepts that reference their external definition.
- **Not built.** Computed attributes and functions, typed SDK generation, graph analytics, and time-series or geospatial property types are not part of the current system and are gated on demonstrated demand.

10.1 Future work: combining trust and performance gates

The two governance signals of Section 2.4 could be combined, and the pieces MNEME already has suggest how. This is a direction, not a shipped capability; no part of the pipeline below exists today. A scenario overlay (Section 4.5) could play the role of a candidate version, the base graph the role of its parent, and promotion through proposals and staging the role of publication. Proposals could be mined from agent trajectories and staged into an overlay rather than written by hand. A paired evaluation could then attach a performance score to each proposal before a steward reviews it, so the steward decides with both signals in view: whether the change helps, and whether it is authorized and safe. The same loop would produce the write-back benchmark that Section 8.3 lists as missing, measuring promotion accuracy and false-promotion rate on labeled proposals.

11 Conclusion

Enterprises have accepted that agents need a knowledge graph, and production enterprise offerings have almost universally built it read-only, leaving the institutional memory unable to learn from the agents that depend on it. MNEME makes the ontology active memory instead: agents read it, validate against it, and write back to it, while provenance, deduplication, conflict detection, low-confidence staging, quarantine from canonical retrieval, and earned promotion ensure that a contribution improves the graph only when it has proven itself. The authoritative core, imported from the platform data model, stays read-only to agents throughout. The same discipline now reaches the schema, the graph's history, what-if scenarios, and the actions a concept can trigger. The effect is a structured memory that gets richer every run without the regulatory blast radius that has kept production platforms read-only, a knowledge graph agents can write to, safely, which is the capability the next generation of enterprise agents will be built on.

References

[1] Gartner. *Top Trends in Data and Analytics for 2026.* Pidsley, D., Thanaraj, R., Long, C., Ramakrishnan, R., & Sallam, R. Gartner Research, 17 February 2026.

[2] Edge, D., Trinh, H., Cheng, N., et al. *From Local to Global: A Graph RAG Approach to Query-Focused Summarization.* arXiv:2404.16130, 2024 (v2, 2025). (LLM-judged comprehensiveness and diversity win rates for graph-based global retrieval over vector RAG.)

[3] Chong, M., Zhang, S., Fan, J., & Du, X. *EvoOntology: A Self-Evolving Ontology Layer for Data Agents.* arXiv:2609.15779, 2026.

[4] Blumenfeld, Z. *Graph-driven AI for all: Neo4j Aura Agent enters general availability.* Neo4j blog, 3 February 2026.

[5] Gruber, T. R. *A Translation Approach to Portable Ontology Specifications.* Knowledge Acquisition, 5(2):199–220, 1993.

[6] Hogan, A., Blomqvist, E., Cochez, M., et al. *Knowledge Graphs.* ACM Computing Surveys, 54(4):1–37, 2021.

[7] Lewis, P., Perez, E., Piktus, A., et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.* Advances in Neural Information Processing Systems (NeurIPS) 33, 2020.

[8] Hu, Y., Liu, S., Yue, Y., Zhang, G., et al. *Memory in the Age of AI Agents: A Survey.* arXiv:2512.13564, 2025.

[9] World Wide Web Consortium. *PROV-O: The PROV Ontology.* W3C Recommendation, 2013.

[10] Reimers, N., & Gurevych, I. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks.* Proc. EMNLP 2019. (MiniLM sentence-transformer family.)

[11] Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., & Zhou, M. *MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers.* NeurIPS 2020. arXiv:2002.10957.

[12] Rasmussen, P., Paliychuk, P., Beauvais, T., Ryan, J., & Chalef, D. *Zep: A Temporal Knowledge Graph Architecture for Agent Memory.* arXiv:2501.13956, 2025.

[13] Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., & Manning, C. D. *HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering.* Proc. EMNLP 2018. arXiv:1809.09600.

[14] European Parliament and Council. *Regulation (EU) 2024/1689 (Artificial Intelligence Act).* Official Journal of the European Union, 2024.

[15] National Institute of Standards and Technology. *Artificial Intelligence Risk Management Framework (AI RMF 1.0).* NIST AI 100-1, 2023.

[16] International Organization for Standardization. *ISO/IEC 42001:2023, Information technology, Artificial intelligence, Management system.* ISO, 2023.

[17] Răduță, B. *RAILS: The Graph Is Deterministic, Even When the Nodes Are Not.* FlowX.AI Technical Paper, v1.1, August 2026.

Appendix A Node Output Keys

Each toolkit node writes well-known keys to the workflow state, suitable for prompt injection or downstream branching.

Table 4. Workflow-state keys written by each node.

Node	Output keys
Retrieve	backend, items[], ontology_context (flat text for prompt injection), relations[] (induced among returned concepts)
Classify	concept_id, label, category, confidence, matched, backend
Describe	concept, ancestors[], grounding_text (formatted)
Expand	related[], ids[] (flat list)
Tag	tags[], tag_ids[] (deduplicated), backend
Deduplicate	recommended_action, is_duplicate, most_similar, neighbors[], backend
Validate	valid, violations[], backend (sets a halt signal on strict failure)
Memory Write	staged, conflict
Memory (all-in-one)	Union of the above for the modes it ran
Act	halted, descriptor (binding_id, concept_id, process_ref, inputs, signature, issued_at, expires_at) or violations[] (sets the same halt signal as Validate)
Extract	the job report (candidates_seen, staged, reinforced, conflicted, skipped, per_document_counts, truncated) or the job error
Intent	the intent result: entities[] (mention, concept_id, confidence), timeframe, scope (categories), action_hint, unresolved[]

Appendix B The Provenance and Conflict Model

The trust model that makes write-back safe, in one place.

Table 5. The provenance, conflict, and promotion model.

Element	Values	Effect
Provenance	pdm_import / agent_write / manual	pdm_import is authoritative and read-only to agents; agent_write is staged and quarantined, and keeps that provenance after promotion; manual is explicit human entry or a governed external import (regulatory corpus, SKOS/OWL, OSI)
Confidence	0–1; agent writes staged at ≤ 0.6	A staged write is durable but excluded from canonical listing, search, and traversal; promotion sets confidence to 1.0
Conflict: label clash	same label, different concept	Logged before stage; routed for review
Conflict: category mismatch	proposed category contradicts structure	Logged before stage; routed for review
Conflict: relation contradicts	new relation contradicts an existing one	Logged before stage; never silently overwrites
Review state	none / needs_review	Set automatically when a conflict is detected at staging; blocks reinforcement auto-promotion until explicitly promoted
Promotion	reinforcement (default 3 runs) / human review / API	Moves a staged fact to canonical; validation against IS_A inheritance must pass
Supersession	valid_from / valid_to; recorded_at / invalidated_at	A promotion that contradicts an existing fact invalidates it (archived, never deleted); as-of reads still see the old fact
Validation strictness	strict / warn / advisory	Strict halts the flow on a schema violation

Appendix C PDM-to-Ontology Mapping and Import Sources

The one-way import that bootstraps an ontology from an existing platform data model without duplicating it. Re-import preserves agent extensions and updates only PDM-sourced fields.

Table 6. Platform data model to ontology mapping.

Platform Data Model	Becomes in the ontology
Entity	Concept (provenance = pdm_import, read-only to agents)
Foreign-key relationship	Relation between concepts
Type / subtype hierarchy	IS_A relation (drives inherited-attribute validation)
Attribute / column	Concept attribute in the schema
(agent-added structure)	Preserved across re-import; only PDM-sourced fields refreshed

The PDM importer is one of four import paths (Table 7). The PDM, SKOS/OWL, and OSI importers offer a dry-run preview, and none of the four bypasses the provenance model.

Table 7. Import sources.

Importer	Maps	Provenance and scope
Platform data model	Entities to Concepts, foreign keys to Relations, type hierarchy to IS_A	pdm_import; one-way, read-only to agents
Regulatory corpus	Framework, Regulation, Chapter, Article, Provision as an IS_A chain, with citation metadata	manual; re-import versions changed provisions bi-temporally
SKOS / OWL	SKOS concepts, broader and related links, alternate labels; a named-class OWL subset (classes, subclass, object properties with named domain and range)	manual; unmappable axioms reported as skipped, never silently dropped; ships a small FIBO-based seed
OSI / Ossie	Models, datasets, fields, relationships, and metrics as concepts and relations, keyed by stable source paths	manual; structural only, metric expressions stay opaque external references

Revision history

v1.0, June 2026: initial release. v1.1, August 2026: added the empirical evaluation (Section 8). v1.2, September 2026: system description brought up to date with the shipped Ontology Layer (twelve nodes, access control, bi-temporal facts, schema proposals, scenario overlays, extraction, action binding, importers, MCP); related work on self-evolving ontologies; accuracy fixes (an unsourced retrieval figure removed, the BM25 baseline correctly labeled, tables and references renumbered).

FlowX.AI

FlowX.AI builds and orchestrates enterprise-grade AI agents for regulated industries, with a focus on banking, financial services, and insurance. The FlowX.AI Ontology Layer gives agents a shared, provenance-tracked structured memory they can read, validate against, and safely contribute to, so institutional knowledge stays consistent and grows with use rather than decaying between manual governance cycles.

FlowX.AI Technical Paper · MNEME v1.2 · Bogdan Răduță, Head of Research · bogdan.raduta@flowx.ai