

Grokers: Bottom-Up Inductive Comprehension and Write-Time Intelligence over Typed Knowledge Graphs

Gregory Magarshak
Qbix, Inc. & Intercoin, Inc.
IE University NYC
New York, NY, USA
gmagarshak@faculty.ienyc.edu

Abstract—We present GROKERS, an architecture for building persistent, structured comprehension of typed knowledge graphs through bottom-up inductive traversal of dependency subgraphs. Unlike retrieval-augmented generation (RAG), which pays full comprehension cost at every query, GROKERS pushes intelligence to write time: autonomous Groker agents analyze nodes in a typed stream graph, extract structured attributes via governed language model (LM) calls, and inductively compose that understanding upward through dependency relations—writing enriched typed attributes that serve all future queries at zero additional LM cost. We prove three formal properties: (1) the *Byte-Identity Theorem*, establishing that context blocks assembled from a transactionally-maintained denormalization index are byte-identical across LM turns between semantic changes, enabling KV-cache hit rates approaching 100%; (2) the *Accumulation Monotonicity Theorem*, establishing that the fraction of interactions resolved without LM calls is non-decreasing in the number of completed interactions under a governed wisdom library growth protocol; and (3) the *Dual-Traversal Ordering Theorem*, establishing that top-down generation and bottom-up comprehension are the unique correct traversal orderings for their respective tasks over a dependency DAG, and that their composition closes into a complete generation-comprehension cycle. We further present a deterministic alternative to embedding-based semantic search, with a synonym caching protocol whose LM fallback rate converges to zero for finite-vocabulary domains. A reference implementation is provided in the open-source Qbix / Safebox / Safebots stack, building on the Magarshak Machine SPACER substrate [1].

Index Terms—knowledge graphs, bottom-up comprehension, LLM agents, write-time intelligence, KV-cache optimization, dependency analysis, semantic search, wisdom library, Magarshak Machine

I. INTRODUCTION

The dominant paradigm for grounding language model (LM) responses in structured knowledge is *retrieval at query time*: embed the query, find nearest-neighbor document chunks by vector similarity, inject them into the prompt context [2]. This approach has a structural deficiency: it pays the full cost of comprehension on every query, regardless of whether that query is novel or structurally equivalent to thousands of prior queries. For knowledge domains with recurring interaction patterns—enterprise software, document management, structured task execution—this is architecturally wasteful.

We present GROKERS, an architecture that inverts this design choice. Comprehension work is performed *once*, at write time, by autonomous agents that traverse the dependency graph of a typed knowledge substrate bottom-up—from leaf nodes toward roots—extracting and storing structured typed attributes via governed LM calls. By query time, the relevant structure is already present as typed attributes on graph nodes.

The core substrate is the *typed stream graph*: a directed graph of typed nodes with structured fields, typed attributes, and weighted typed edges, realizing the Streams (S) and Relations (R) components of the Magarshak Machine SPACER framework [1]. A transactionally-maintained denormalization table (STREAMS_CATEGORY) stores, for each node, the complete pre-ranked relation neighborhood updated atomically on every relation modification, providing complete context in ≈ 1 ms.

Contributions

- 1) The GROKERS write-time comprehension architecture (§V).
- 2) The *Byte-Identity Theorem* with KV-cache cost analysis (§IV).
- 3) The *Wisdom Library* with the *Accumulation Monotonicity Theorem* (§VI).
- 4) The *Dual-Traversal Ordering Theorem* with incremental expansion corollary (§VII).
- 5) A deterministic semantic search system with *Synonym Cache Convergence* (§IX).

Positioning

Individual components—knowledge graphs [3], LM-based enrichment [4], KV caching [5], program libraries [6]—are each established. Our contribution is the formal analysis of their composition and the three proved properties above, none of which appear in prior work.

II. RELATED WORK

Retrieval-augmented generation. Lewis et al. [2] established RAG as the standard for grounding LM responses.

GraphRAG [3] adds community summarization over a knowledge graph but retains embedding similarity as the retrieval mechanism; neither GraphRAG nor any RAG variant achieves byte-identity of assembled context or monotone LM-call elimination.

Code comprehension agents. Bottom-up comprehension is established in software engineering [7]. Recent LM tools (Copilot [8], Devin [9], RepoUnderstander [4]) operate reactively without persistent enriched structure or formal traversal-ordering guarantees.

Memory systems. MemGPT [10] and Mem0 [11] retrieve facts by embedding similarity at query time. GROKERS differs: enrichment is write-time, retrieval is a typed graph read, and the accumulation property is proved.

KV-cache optimization. Anthropic prompt caching [5], SGLang [12], and LMCache [13] cache whatever prefix the caller provides. GROKERS is, to our knowledge, the first to *architect* context assembly so that stable prefixes are byte-identical from first principles.

Magarshak Machine. Magarshak [1] introduces the SPACER substrate: append-only streams, policy governance, five-phase action execution (COMPUTE \rightarrow REQUIRE \rightarrow EXECUTE), and the bidirectional relation index. GROKERS is an application layer on this substrate; its formal properties derive in part from SPACER’s substrate guarantees (Minimal Cache Invalidation, Embarrassing Parallelism, ε -deterministic view capabilities).

Hierarchical generation. Hierarchical generation [14], [15] and planning-then-writing [16] do not formalize traversal ordering or relate it to KV-cache stability. The Dual-Traversal Ordering Theorem (§VII) provides this connection.

III. THE TYPED STREAM GRAPH

Definition 1 (Typed Stream Graph). A typed stream graph $G = (V, E, \tau, \alpha, w)$ consists of: node set V with type function $\tau : V \rightarrow \mathcal{T}$; directed typed edge set $E \subseteq V \times \mathcal{R} \times V$; typed attribute function $\alpha : V \rightarrow (K \rightarrow A)$; and weight function $w : E \rightarrow \mathbb{R}_{\geq 0}$ driven by vote aggregation $w(v, r, u) = \sum_i \text{vote}_i(v, r, u)$, updated in the same database transaction as vote events (realizing the *R/Relations* component of [1]).

Definition 2 (Dependency Subgraph). The dependency subgraph $D(v)$ is the induced subgraph on nodes reachable from v by `depends_on` edges. A node $u \in D(v)$ is a leaf if it has no outgoing `depends_on` edges within $D(v)$. We assume $D(v)$ is a DAG for all $v \in V$.

Definition 3 (Enriched / Stale Nodes). Node v is enriched if $\alpha(v)$ contains non-null values for the core schema fields {digest, keywords, qualityScore}. (Implementations may include additional fields such as `grokVersion` for cache invalidation; these are outside the formal model.) Node v is stale if $\alpha(v)[\text{stale}] = \text{true}$.

IV. DENORMALIZATION INDEX AND THE BYTE-IDENTITY THEOREM

A. Index Structure

The `STREAMS_CATEGORY` table maintains for each $v \in V$:

$$N(v) = \{r \mapsto \{w_j \mapsto [p_j, s_j, t_j, i_j]\}_j \mid (v, r, u_j) \in E\}$$

where p_j, s_j are the `publisherId/streamName` of the j -th neighbor at weight w_j under relation type r , and t_j, i_j are its denormalized title and icon. The table is updated in the same database transaction as every `Streams :: relate()` / `Streams :: unrelate()` call—no background jobs, no eventual consistency.

Definition 4 (Context Block). The context block $C(v)$ is the output of the deterministic function `buildCachedContext(v)`: reads $\alpha(v)$ and $N(v)$; renders them as structured text with relation types ordered by signal strength (unique types first, non-unique by maximum weight descending).

B. The Byte-Identity Theorem

Theorem 1 (Byte-Identity). Let $t_1 < t_2$ be timestamps such that no write to $\alpha(v)$ or any edge incident to v occurs in (t_1, t_2) . Then $C_{t_1}(v) = C_{t_2}(v)$ as byte strings.

Proof. $C(v)$ is a deterministic function of $(\alpha(v), N(v))$. $N(v)$ is modified only within transactions that modify edges incident to v ; none occur in (t_1, t_2) by hypothesis. $\alpha(v)$ is modified only by governed writes (via `Action.propose` in SPACER’s EXECUTE phase [1]); none occur by hypothesis. Both inputs are identical at t_1 and t_2 ; determinism gives $C_{t_1}(v) = C_{t_2}(v)$. \square

Corollary 1 (KV Cache Hit Rate). Assume semantic changes to v follow a Poisson process with mean inter-arrival time $T_c(v)$, and turns arrive with mean inter-turn interval T_t , with $T_t \ll T_c(v)$. The expected fraction of turns for which $C(v)$ is a cache hit is $1 - T_t/T_c(v)$, which approaches 1 as $T_c(v)/T_t \rightarrow \infty$.

Proof. Under the Poisson assumption, the probability that at least one semantic change occurs in any given inter-turn interval $[t, t+T_t]$ is $1 - e^{-T_t/T_c(v)} \approx T_t/T_c(v)$ for $T_t \ll T_c(v)$. By Theorem 1, a cache hit occurs in every turn where no change has occurred since the previous turn. The expected hit rate is therefore $1 - T_t/T_c(v) \rightarrow 1$ as $T_c(v)/T_t \rightarrow \infty$. \square

For synchronous or near-synchronous interaction (e.g., chat), T_c is hours-to-days and T_t is seconds, so the hit rate approaches 100%. For asynchronous workflows where $T_t \sim T_c$, the hit rate decreases accordingly; the benefit is greatest in systems with rapid turn cadence relative to semantic change rate. No RAG-assembled context achieves this: retrieved chunks vary with the query, making context non-deterministic across turns even when the underlying knowledge has not changed.

Remark 1 (Relation to $\mathcal{M}\mathcal{M}$ Minimal Cache Invalidation). Let S be the set of streams requiring re-materialisation identified by $\mathcal{M}\mathcal{M}$ ’s Minimal Cache Invalidation Theorem [1] after

a dependency change. For all $v \notin S$, no write to $\alpha(v)$ or edges incident to v has occurred in the relevant interval; by Theorem 1, $C(v)$ is byte-identical and the KV cache entry for v is valid without re-computation. The two theorems partition the stream set: S identifies what must be re-computed; Theorem 1 certifies that the complement need not be.

C. Cost Analysis

Let k_s denote the stable prefix token count (a value determined by the goal system prompt and the denormalization-assembled neighborhood block; in our reference implementation, k_s is on the order of 2,000–3,000 tokens). At 10% cache-read pricing (Anthropic prompt caching [5] charges 10% of normal input cost for cache hits and 125% for cache writes, with the write cost amortized over the cache lifetime; we account for read cost only in the steady-state analysis):

$$\begin{aligned} \text{GROKERS} &: 0.1k_s + k_d \text{ per turn} \\ \text{RAG} &: k_r + k_d \text{ per turn} \quad (k_r \approx k_s - 3k_s) \end{aligned}$$

where k_d is dynamic context tokens. Stable prefix cost reduction: up to $10\times$ in the limit $k_d \ll k_s$ with $k_r = k_s$; the reduction approaches $(k_r + k_d)/(0.1k_s + k_d)$, which increases as k_r grows relative to k_d .

V. GROKER AGENTS: BOTTOM-UP COMPREHENSION

Definition 5 (Valid Processing Order). *A sequence v_1, \dots, v_n is a valid Groker processing order for DAG H if for every v_j and every u with $(v_j, \text{depends_on}, u) \in E(H)$, u appears before v_j (i.e., a topological sort of H with leaves first).*

Theorem 2 (Composability of Groker Enrichment). *Call a Groker invocation on node v individually correct if, given that all dependencies of v are correctly enriched, it produces enrichment for v that passes schema validation and maintains fitness $\geq f_{\min}$ (see §XI). If each Groker invocation is individually correct, then processing nodes in any valid order (Definition 5) produces correct enrichment for all nodes.*

Proof. Induction on topological order. *Base:* leaf nodes have no dependencies; correctness holds vacuously. *Step:* all prior nodes are correctly enriched by hypothesis; all dependencies of v_i appear before it in the order, so they are correctly enriched; individual correctness gives correct enrichment for v_i . \square

Each Groker requires only its node’s content and pre-enriched dependency attributes—not global context. Independent Groker passes on nodes with no shared dependency path write to disjoint attribute namespaces ($\alpha(v)$ for each distinct v), satisfying the disjoint write-set condition of \mathcal{MM} ’s Embarrassing Parallelism Theorem [1] for attribute enrichment, enabling coordination-free parallel enrichment scaling linearly with publisher count. Index updates to the inverted keyword index Idx involve shared mutable state and are serialized by the substrate’s transactional index maintenance, separate from and non-blocking to attribute enrichment parallelism.

A. Staleness Propagation

Proposition 1 (Staleness Completeness). *Let v be modified. Define the reverse dependency closure $S(v) = \{u \mid \exists \text{ path } u \rightsquigarrow v \text{ via } \text{depends_on}\}$. After marking all $u \in S(v)$ stale and reprocessing them in valid topological order, no node is stale.*

Proof. By Theorem 2: reprocessing $S(v)$ in valid order with v already updated restores correctness to all affected nodes. Nodes not in $S(v)$ have no dependency path to v and are unaffected. \square

Propagation cost is $O(|S(v)|)$ —the build-system model applied to semantic comprehension. This mirrors \mathcal{MM} ’s Minimal Cache Invalidation Theorem [1]: only streams reachable from the modified node in the dependency graph require reprocessing.

VI. THE WISDOM LIBRARY AND ACCUMULATION MONOTONICITY

A. Structure

The *wisdom library* $\mathcal{W} = \{p_i\}$ is a finite set of *sandboxed imperative programs*, each with an execution phase $\phi(p_i) \in \Phi$, input/output schema, and *fitness* $f(p_i) = |\{x \in \mathcal{X}_{\text{eval}} : p_i \text{ routes and handles } x \text{ correctly}\}| / |\mathcal{X}_{\text{eval}}| \in [0, 1]$ measured over a rolling evaluation window $\mathcal{X}_{\text{eval}}$. Execution contract (realizing SPACER’s COMPUTE/REQUIRE/EXECUTE separation): reads from a pre-loaded immutable input object (no live DB queries); writes only via proposal accumulation (no direct writes, corresponding to REQUIRE); time bound ≤ 50 ms; memory ≤ 64 MB; no network access. Programs are ε -deterministic view capabilities with $\varepsilon = 0$ (imperative code, not stochastic models), enabling \mathcal{MM} ’s Probabilistic Consensus mechanism [1] to achieve agreement probability exactly 1 across Safebox nodes executing the same program on the same input.

Definition 6 (LM-Call Elimination Rate). $E(\mathcal{W}) = \frac{|\{x \in \mathcal{X} : \text{routing}(x, \mathcal{W}) \neq \text{LM}\}|}{|\mathcal{X}|}$ where \mathcal{X} is the interaction distribution for goal type g .

B. Three Growth Mechanisms

Initial authoring: at goal type creation, an LM generates $\mathcal{W}^{(0)} = \{p_1^{(0)}, \dots, p_k^{(0)}\}$ covering all phases, reviewed before activation.

Pattern promotion: after completed interaction x_t , a Groker analysis pass identifies uncovered patterns and proposes A_t : $\mathcal{W}^{(t+1)} = \mathcal{W}^{(t)} \cup A_t$.

Evolutionary selection: programs with $f(p_i) < f_{\min}$ are replaced by higher-fitness fork variants evaluated in parallel—the program-level instantiation of Grokers expert specialization.

Theorem 3 (Accumulation Monotonicity). $E(\mathcal{W}^{(t+1)}) \geq E(\mathcal{W}^{(t)})$ for all $t \geq 0$.

Proof. *Pattern promotion:* $\mathcal{W}^{(t+1)} = \mathcal{W}^{(t)} \cup A_t$. Here “resolved” means *correctly* resolved: the program routes x and its

output passes schema validation. The pattern promotion protocol requires all programs in A_t to pass review (schema validation and sandbox execution against held-out examples) before joining \mathcal{W} ; this gates out programs that would route incorrectly. Under this guard, for any x : if $\text{routing}(x, \mathcal{W}^{(t)}) \neq \text{LM}$, existing programs still correctly route x under $\mathcal{W}^{(t+1)}$ (new programs cannot remove correct coverage from previously resolved inputs). If x was unresolved but correctly covered by $p \in A_t$, it is now correctly resolved. Therefore the correctly-resolved set is non-decreasing: $E(\mathcal{W}^{(t+1)}) \geq E(\mathcal{W}^{(t)})$.

Evolutionary selection: We assume the *fitness faithfulness* condition: $f(p') > f(p)$ implies $|\{x \in \mathcal{X} : p'(x) \neq \text{LM}\}| \geq |\{x \in \mathcal{X} : p(x) \neq \text{LM}\}|$, i.e., empirical fitness is a reliable proxy for coverage on the full distribution. Under this assumption, p' covers at least the interactions well-handled by p . Fitness-based replacement therefore cannot decrease coverage, giving $E(\mathcal{W}^{(t+1)}) \geq E(\mathcal{W}^{(t)})$.

Since both mechanisms are non-decreasing and the theorem holds under the stated assumption, the result follows. \square

Corollary 2 (Decreasing Marginal LM Cost). $C_{\text{LM}}^{(t)} = (1 - E(\mathcal{W}^{(t)})) \cdot c_{\text{LM}}$ is non-increasing in t . RAG, ReAct [17], and LangChain [18] all have constant marginal LM cost per interaction—Theorem 3 is the unique mechanism among those considered that breaks this constant-cost property.

The elimination rate $E(\mathcal{W}^{(t)})$ is bounded above by the structural predictability P_g of goal type g . For task-oriented goal types (capability building, document review, support resolution), P_g is expected to be substantial—reflecting the large fraction of interactions that follow predictable structural patterns—though its precise value is domain-dependent and subject to empirical measurement (see Discussion, §XI). The remaining fraction requires genuine synthesis and LM calls indefinitely.

VII. DUAL-TRAVERSAL ORDERING

Definition 7 (Coherent Generative Task). A coherent generative task over DAG H assigns artifact content to each node v using the content of its dependencies as shared context, and is coherent if all consumers of a shared dependency use the same version of that dependency.

Theorem 4 (Dual-Traversal Ordering). Let H be a dependency DAG. (a)

- 1) The valid orderings for correct Groker comprehension are precisely the topological sorts of H with leaves first (bottom-up); any ordering not in this class may produce incorrect enrichment for some node.
- 2) The valid orderings for coherent artifact generation over H are precisely the topological sorts of H with roots first (top-down); any ordering not in this class may produce an incoherent artifact for some node.
- 3) These two classes of orderings are reverses of each other on any DAG: reversing any member of the root-first class yields a member of the leaf-first class, and vice versa.

The enriched attributes $\{\alpha(v)\}_{v \in V}$ produced by bottom-up comprehension of a generated artifact constitute, via `buildCachedContext`, a byte-identical (Theorem 1) KV-cached context for subsequent top-down generation of artifacts over the same schema.

Proof. (a) Let $(v, \text{depends_on}, u) \in E$. Groker comprehension of v requires enriched $\alpha(u)$. If v is processed before u , $\alpha(u)$ is unenriched and correctness fails. Therefore u must precede v —a topological sort with leaves first. Sufficiency follows from Theorem 2.

(b) Generating v requires u 's content as shared context (definition of coherent generative task). If v is generated before u , shared context is unavailable and coherence fails. Therefore u must precede v —a topological sort with roots first.

(c) Root-first and leaf-first topological sorts are reverses of each other on any DAG. The generation phase (order σ) produces content for all nodes, each artifact appended to the graph (Append-Only Safety of [1] guarantees no loss). The comprehension phase (order σ^{-1}) enriches all nodes. The resulting $\{\alpha(v)\}$ is assembled by the deterministic `buildCachedContext` into a context block that, by Theorem 1, is byte-identical across turns until the artifact changes. \square

Corollary 3 (Incremental Shared Dependency Expansion). During top-down generation, when a leaf encounters a new shared dependency d not in the current shared dependency stream: (i) Adding d does not invalidate leaves not depending on d ; (ii) leaves already generated that depend on d are stale (Proposition 1) and require regeneration; (iii) all subsequent leaf generations have d available as shared context.

This corollary formalizes the website generation pattern: a new CSS variable discovered during page generation is added to the design system stream, requires regenerating only affected pages, and is available to all subsequent pages.

A. Context Stability Hierarchy

The dual-traversal structure maps directly onto the KV-cache stability hierarchy:

- **PERMANENT:** root-level architecture, stable for the goal type's lifetime
- **SESSION:** section/module context, byte-identical between semantic changes
- **COLD:** multi-level summary tree, changes every $\approx 10^2$ – 10^3 messages
- **DYNAMIC:** wisdom-program-selected context, varying per turn

Root-level architecture \rightarrow PERMANENT; section context \rightarrow SESSION; leaf requirements \rightarrow DYNAMIC. Shared context at each hierarchy level is computed once and reused across all descendant leaf generations.

VIII. OBSERVATIONS SCHEMA AND WRITE-TIME EXTRACTION

An *observations schema* maps stream types T to extraction specifications $\sigma(T, C)$: *instruction clauses* $[\ell_1, \dots, \ell_k]$

(natural-language directives constraining LM extraction output, constituting the stable portion of the system prompt for type T), output schema Σ , attribute namespace π , and staleness condition δ .

Extraction uses a two-part call: the *stable system prompt* (constant across all extractions of type T and category C , KV-cached by Theorem 1) and the *dynamic user content* (the specific stream’s content and existing attributes, varying per stream).

Proposition 2 (Extraction Amortization). *With k_s stable system prompt tokens and \bar{k}_u mean dynamic content tokens, the mean per-extraction prefill cost at 10% cache pricing is $\bar{C} = 0.1k_s + \bar{k}_u$. The asymptotic reduction factor relative to no caching is: $R = (k_s + \bar{k}_u)/(0.1k_s + \bar{k}_u) \xrightarrow{k_s/\bar{k}_u \rightarrow \infty} 10$.*

Proof. Without caching, the per-extraction prefill cost is $k_s + \bar{k}_u$. With 10% cache-hit pricing on the stable prefix, the effective cost is $0.1k_s + \bar{k}_u$. The reduction factor is $R = (k_s + \bar{k}_u)/(0.1k_s + \bar{k}_u)$. Dividing numerator and denominator by \bar{k}_u : $R = (k_s/\bar{k}_u + 1)/(0.1k_s/\bar{k}_u + 1) \rightarrow 1/0.1 = 10$ as $k_s/\bar{k}_u \rightarrow \infty$. \square

Extracted attributes, once written, serve context assembly, preprocessing, relevance scoring, autosuggest, and search indexes—all at zero additional LM cost.

IX. DETERMINISTIC SEMANTIC SEARCH

A. The Precision Problem with Embedding Similarity

For structured domains, typed attribute queries (“streams with $\alpha(v)[\text{status}] = \text{unconfigured}$ ”) have zero false positives by construction—a precision level unachievable by embedding similarity, which computes undifferentiated similarity over raw content.

B. Write-Time Indexing and Query Expansion

Keyword extraction is a first-class observation category: the Groker extracts normalized keyword set $K(v)$ for each v , stored as $\alpha(v)[\text{keywords}]$ and indexed into inverted index $\text{Idx}[k] = \{v \mid k \in K(v)\}$. Query terms are expanded via: stemming; domain ontology traversal (IS-A, SYNONYM-OF edges as typed graph relations); thesaurus lookup; and cached prior LM expansions. Search resolves by set intersection: $\text{score}[v] = |Q \cap K(v)|$.

Theorem 5 (Synonym Cache Convergence). *Let \mathcal{W} be the finite domain vocabulary with $|\mathcal{W}| < \infty$. Assume each query contains at least one term ($|\mathcal{T}_q(n)| \geq 1$ for all n). The LM fallback rate $\rho(n) = |\mathcal{T}_q^{\text{new}}(n)|/|\mathcal{T}_q(n)| \rightarrow 0$ as $n \rightarrow \infty$.*

Proof. Let $S_n = \bigcup_{i=1}^n \mathcal{T}_q(i)$ be the set of all vocabulary terms seen in queries 1 through n . Since $S_n \subseteq \mathcal{W}$ and $|\mathcal{W}| < \infty$, the sequence $|S_n|$ is non-decreasing and bounded above by $|\mathcal{W}|$, so it converges. Therefore $|\mathcal{T}_q^{\text{new}}(n)| = |S_n \setminus S_{n-1}| \rightarrow 0$. Since $|\mathcal{T}_q(n)| \geq 1$ by assumption, the denominator is bounded below, and $\rho(n) = |\mathcal{T}_q^{\text{new}}(n)|/|\mathcal{T}_q(n)| \leq |\mathcal{T}_q^{\text{new}}(n)| \rightarrow 0$. \square

Theorem 5 mirrors Theorem 3: in both cases, LM cost per operation converges to zero for finite-domain workloads as the relevant cache accumulates coverage.

X. IMPLEMENTATION

The architecture is realized across three plugins. **Qbix** provides the stream graph substrate: `Streams::relate()`, `Streams_Category` (maintained transactionally), `Users_Vote`. **Safebox** provides the sandbox (realizing SPACER’s C /Capabilities and E /Execution components [1]): `Protocol.LLM` with Anthropic and OpenAI adapters, `Action.propose` governed write pipeline, `Protocol.Transcription`, `Protocol.Image`. **Safebots** provides the Groker enrichment layer, goal stream architecture, multi-level summary tree, wisdom library management, and preprocessing pipeline.

KV-cache support: For Anthropic models, PERMANENT and SESSION blocks are passed as system content blocks with `cache_control:{type:"ephemeral"}`, establishing explicit breakpoints. For OpenAI models, they are merged into a single system message to maximize stable prefix length for automatic prefix caching.

XI. DISCUSSION

Correctness assumptions. Theorem 2 assumes individually correct Groker invocations. In practice, Grokers use LMs and are stochastic; the governed write pipeline mitigates this via schema validation and fitness-based evolution. “Correct” means operationally: passes schema validation and maintains fitness $\geq f_{\min}$ over the evaluation window.

Finite vocabulary assumption. Theorem 5 requires $|\mathcal{W}| < \infty$. For open-domain deployments the vocabulary is unbounded; in this case $\rho(n)$ converges to the fraction of novel user terms that have no ontology expansion, which may remain bounded away from zero indefinitely. The theorem applies to structured knowledge domains (product catalogs, codebases, organizational ontologies) where the domain vocabulary is finite and indexable.

Acyclic dependency assumption. Theorem 4 assumes acyclic dependency graphs. Cyclic dependencies (mutual imports, circular references) require cycle-breaking heuristics; we leave this extension to future work.

Architecture stack. The full Magarshak Architecture is: $\underbrace{\mathcal{M}\mathcal{M}}_{\text{substrate}} \rightarrow \underbrace{\text{Grokers}}_{\text{comprehension}} \rightarrow \underbrace{\text{Context}}_{\text{intelligence}}$ [19]. This paper covers the comprehension layer; the intelligence layer—proactive goal-directed agents, organizational efficiency theorems, cross-platform governance—is treated in [19].

XII. CONCLUSION

GROKERS provides a write-time intelligence architecture with three proved properties. The Byte-Identity Theorem establishes near-100% KV-cache hit rates on deterministically assembled stable prefixes—fundamentally unachievable by retrieval-based systems. The Accumulation Monotonicity Theorem establishes that the wisdom library covers an increasing

fraction of interactions without LM calls, with marginal LM cost non-increasing over time—a property no existing chatbot architecture exhibits. The Dual-Traversal Ordering Theorem establishes the duality of generation and comprehension as the unique correct orderings for their respective tasks, composing into a complete cycle in which comprehension output becomes cached context for future generation. These properties constitute a formal basis for write-time intelligence as a strictly more efficient alternative to query-time retrieval for structured knowledge domains with recurring interaction patterns.

ACKNOWLEDGEMENTS

The stream graph substrate is implemented in the Qbix open-source framework, which realizes the Magarshak Machine SPACER substrate [1].

REFERENCES

- [1] G. Magarshak, “The Magarshak machine: A stream-partitioned model for governed state evolution. the SPACER framework,” *arXiv preprint arXiv:2501.XXXXX*, 2026.
- [2] P. Lewis, E. Petriv, A. Piktus *et al.*, “Retrieval-augmented generation for knowledge-intensive NLP tasks,” *NeurIPS*, vol. 33, pp. 9459–9474, 2020.
- [3] D. Edge, H. Trinh, N. Cheng *et al.*, “From local to global: A Graph RAG approach to query-focused summarization,” *arXiv preprint arXiv:2404.16130*, 2024.
- [4] Y. Ma, Z. Zhang *et al.*, “RepoUnderstander: Agent-enhanced repository-level code comprehension,” *arXiv preprint arXiv:2408.02563*, 2024.
- [5] Anthropic, “Prompt caching,” 2024. [Online]. Available: <https://docs.anthropic.com/en/docs/build-with-claude/prompt-caching>
- [6] M. Chen, J. Tworek, H. Jun *et al.*, “Evaluating large language models trained on code,” *arXiv preprint arXiv:2107.03374*, 2021.
- [7] T. J. Biggerstaff, B. G. Mitbender, and D. Webster, “Program understanding and the concept assignment problem,” *Communications of the ACM*, vol. 37, no. 5, pp. 72–82, 1993.
- [8] GitHub, “GitHub Copilot,” 2023. [Online]. Available: <https://github.com/features/copilot>
- [9] Cognition AI, “Devin: AI software engineer,” 2024. [Online]. Available: <https://cognition.ai>
- [10] C. Packer, S. Wooders, K. Lin *et al.*, “MemGPT: Towards LLMs as operating systems,” *arXiv preprint arXiv:2310.08560*, 2023.
- [11] Mem0 AI, “Mem0: The memory layer for personalized AI,” 2024. [Online]. Available: <https://mem0.ai>
- [12] L. Zheng, L. Yin, Z. Xie *et al.*, “Efficiently programming large language models using SGLang,” in *NeurIPS*, 2024.
- [13] R. Jin *et al.*, “LMCache: An efficient KV cache layer for enterprise-scale LLM inference,” *arXiv preprint arXiv:2410.XXXXX*, 2024.
- [14] A. Fan, M. Lewis, and Y. Dauphin, “Hierarchical neural story generation,” in *ACL*, 2018.
- [15] K. Yang and D. Klein, “DOC: Improving long story coherence with detailed outline control,” in *ACL*, 2022.
- [16] H. Rashkin, A. Celikyilmaz, Y. Choi, and J. Gao, “PlotMachines: Outline-conditioned generation with dynamic plot state tracking,” in *EMNLP*, 2020.
- [17] S. Yao, J. Zhao, D. Yu *et al.*, “ReAct: Synergizing reasoning and acting in language models,” *ICLR*, 2023.
- [18] H. Chase, “LangChain,” 2022. [Online]. Available: <https://langchain.com>
- [19] G. Magarshak, “Context: Proactive goal-directed intelligence via composable sandboxed programs, declarative wiring, and structured interaction,” *arXiv preprint arXiv:2503.XXXXX*, 2026.