Context: Proactive Goal-Directed Intelligence via Composable Sandboxed Programs, Declarative Wiring, and Structured Interaction

Gregory Magarshak*
gmagarshak@faculty.ienyc.edu
Qbix, Inc. & Intercoin, Inc.
New York, USA

ABSTRACT

We present **Context**, the intelligence layer of the Magarshak Architecture, which replaces reactive query-response chatbots with proactive goal-directed agents that advance shared tasks without waiting for user prompts. The architecture rests on three mutually reinforcing mechanisms. *Write-time context assembly* precomputes enriched typed attributes via Groker agents [12], assembling interaction context from a maintained denormalization index as a deterministic pure function of graph state; context blocks are byte-identical across turns between semantic changes, enabling near-100% KV-cache reuse. *Composable sandboxed wisdom programs* form a governed library of LM-generated imperative programs declaratively wired to goal types via typed stream relations, composed via phase ordering, evolved through fitness-based selection, and executed at interaction time without further LM calls. *Proactive goal stream state machines* drive conversations toward terminal states by inspecting graph state and emitting structured interaction content—option arrays, governance affordances, clarification prompts—without awaiting user input. We prove six formal results: (1) the *Context Stability Theorem*, bounding per-turn LM cost as a function of semantic change rate; (2) the *Program Composition Correctness Theorem*, establishing that phase-ordered wisdom program pipelines preserve individual correctness; (3) the *Declarative Wiring Soundness Theorem*, establishing complete and sound event routing; (4) the *Proactive Dominance Theorem*, proving that proactive agents weakly dominate reactive agents on expected turns-to-terminal-state; (5) the *Coordination Overhead Elimination Theorem* and *Quality Preservation Theorem*, establishing that proactive agents are Pareto improvements in multi-participant goal chats—fewer turns, equal or higher artifact quality; and (6) the *Cross-Platform Vote Consistency Theorem*, guaranteeing governance consistency across Telegram, email, web, and mobile. The architecture is implemented in the open-source Qbix / Safebox / Safebots stack.

CCS CONCEPTS

• **Human-centered computing** → **Collaborative and social computing systems and tools**; • **Computing methodologies** → *Natural language processing*; *Knowledge representation and reasoning*.

*Also with IE University NYC.

KEYWORDS

proactive dialogue, goal-directed agents, wisdom library, sandboxed programs, organizational efficiency, knowledge graphs, KV-cache optimization, cross-platform governance

1 INTRODUCTION

Every deployed conversational AI system today is fundamentally reactive. It receives a message and emits a response. For genuinely open-ended questions this is appropriate; for the large majority of goal-directed interactions—building a software capability, reviewing a document, resolving a support ticket, governing a shared artifact—the required next action is often *determinable from prior graph state without any new user input at all.*

Reactive agents in multi-participant goal chats impose a structural inefficiency with no natural corrective: a large fraction of turns are *coordination turns*—turns spent establishing the current state, identifying blockers, and deciding who acts next. These turns produce no progress toward the terminal state; they are pure overhead. We formalize this and prove that proactive agents eliminate it structurally.

Three specific primitives are absent from current architectures: *write-time context* [12], *composable sandboxed programs*, and *proactive state machine intelligence*. This paper introduces all three as an integrated architecture and proves formal properties of each.

Positioning in the Magarshak Architecture

This paper is the third in a series. Magarshak [13] introduces the Magarshak Machine ($\mathcal{MM}$), the SPACER substrate: append-only streams, policy governance, five-phase action execution (Compute → Require → Execute), and the bidirectional relation index. Magarshak [12] introduces Grokers, the write-time comprehension layer: bottom-up inductive enrichment, the Byte-Identity Theorem, and Accumulation Monotonicity. The present paper introduces the **Context** intelligence layer: proactive goal-directed agents, wisdom library composition and declarative wiring, cross-platform rendering, and organizational efficiency theorems.

$$\underbrace{\mathcal{MM}}_{\text{substrate}} \rightarrow \underbrace{\text{Grokers}}_{\text{comprehension}} \rightarrow \underbrace{\text{Context}}_{\text{intelligence}}$$

Contributions

(1) Formal model of the Context architecture (§3).

(2) Wisdom library composition algebra with correctness-preservation theorem (§4).
(3) Declarative wiring model with soundness theorem (§4.2).
(4) Proactive state machine framework with Proactive Dominance Theorem (§5).
(5) Organizational efficiency theorems: Coordination Overhead Elimination and Quality Preservation (§6).
(6) Cross-platform vote consistency theorem (§7).
(7) Dual-traversal Hierarchy-Cache Correspondence Theorem (§8).

2 RELATED WORK

Task-oriented dialogue. POMDP-based systems [17] and neural dialogue models [4, 8] model conversation as belief-state tracking followed by act selection. They do not address write-time context precomputation, governed imperative programs, or organizational efficiency in multi-participant settings. Context inherits the state machine framing and extends it with deterministic graph-derived context, a growing wisdom library replacing learned policies, and proactive advancement without user prompts.

Proactive dialogue. Proactive systems have been studied in recommendation [6] and knowledge-grounded conversation [16]. These define proactivity as topic introduction or recommendation surfacing. Context defines proactivity at the structural level: a bot acts when a state machine advancement condition is satisfied by graph state, correct by construction rather than approximately correct by training.

Multi-agent collaboration. AutoGen [15], MetaGPT [9], and CAMEL [11] focus on agent-to-agent coordination. Context is concerned with the organizational efficiency of *human-AI-human* interactions: the fraction of turns spent on coordination versus productive progress, and how the wisdom library and proactive state machine reduce that fraction toward zero.

RAG. RAG systems [7, 10] address knowledge access at query time. Context replaces query-time retrieval with write-time enrichment and deterministic graph reads; the key formal distinction is the byte-identity property of deterministically assembled context blocks, enabling near-100% KV-cache reuse not achievable by any RAG system.

LM-generated programs. Using LMs to generate executable code is well-studied [2, 5]. The wisdom library adds: governance (programs reviewed before activation); fitness-based evolutionary selection; the read-only sandbox contract; and declarative phase assignment via typed stream relations.

Organizational AI. Empirical studies [3, 14] measure AI productivity on individual tasks but do not formalize multi-participant coordination overhead or prove bounds on proactivity's effect. Section 6 provides the first formal model and theorems on this question.

3 FORMAL MODEL

We build on the typed stream graph of [12] and the SPACER operational semantics of [13].

Definition 3.1 (Goal Stream). *A* goal stream $\mathcal{G} = (T, Q, q_0, F, \delta, \Lambda, \mathcal{W}_\mathcal{G}, \Pi_\mathcal{G})$ *is a tuple where:* T *is a stream type;* Q *a finite state set;* $q_0 \in Q$ *the initial state;* $F \subseteq Q$ *the terminal states;* $\delta : Q \times \Sigma_{attr} \to Q$ *the transition function, where* $\Sigma_{attr} = \{(field, op, val) \mid field \in K,\ op \in \{=, \neq, >, \geq, <, \leq\},\ val \in A\}$ *is the set of typed attribute conditions;* $\Lambda : Q \to \mathcal{P}(InputMode)$ *the per-state input mode function;* $\mathcal{W}_\mathcal{G}$ *the wisdom library (Def. 3.2); and* $\Pi_\mathcal{G}$ *the proactive advancement conditions (Def. 5.2).*

Definition 3.2 (Wisdom Library). *A* wisdom library $\mathcal{W} = \{p_i\}$ *is a finite set of* wisdom programs*, each* $p_i = (n_i, \phi_i, I_i, O_i, f_i, \ell_i)$ *where* n_i *is a name;* $\phi_i \in \Phi$ *an execution phase;* I_i, O_i *input/output schemas;* $f_i \in [0, 1]$ *fitness; and* ℓ_i *the imperative program text whose denotation* $[\![\ell_i]\!] : I_i \to O_i$ *is the function computed by* ℓ_i*. We write* $p_i(x)$ *as shorthand for* $[\![\ell_i]\!](x)$*. Programs execute in the Safebox sandbox: reads from a pre-loaded immutable input (no live DB queries, corresponding to SPACER's* Compute*); writes only via proposal accumulation (no direct writes, corresponding to* Require*); time* ≤ 50 *ms; memory* ≤ 64 *MB; no network except named Protocols.*

Definition 3.3 (Execution Phase Algebra). *Phases* $\Phi = \{\mathsf{pre}, \mathsf{ctx}, \mathsf{post}, \mathsf{auto}, \mathsf{render}, \mathsf{rel}, \mathsf{agg}, \mathsf{idx}\}$ *are partially ordered by data-flow:* $\mathsf{pre} \prec \mathsf{ctx} \prec \mathsf{agg} \prec \mathsf{post} \prec \mathsf{render}$*;* $\mathsf{rel} \prec \mathsf{agg}$*;* $\mathsf{post} \prec \mathsf{auto}$*;* idx *independent. Programs at incomparable phases (e.g.,* rel *and* ctx*) have no data-flow dependency and may execute in any order or concurrently. Sequential composition* $p;q$ *(with* $\phi(p) \prec \phi(q)$*) runs* p *then* q *on the merged input* $x \oplus p(x)$*. Parallel composition* $p \parallel q$ *(same phase or incomparable phases, disjoint output keys) produces merged output* $p(x) \cup q(x)$*.*

Definition 3.4 (Context Block Hierarchy). $\mathcal{C}(\mathcal{G}, v) = B_{\mathsf{perm}}(\mathcal{G}) \cdot B_{\mathsf{sess}}(v) \cdot B_{\mathsf{cold}}(v) \cdot B_{\mathsf{dyn}}(\mathcal{G}, v, t)$ *where:* B_{perm}*: goal system prompt, stable for the goal type's lifetime (KV BP1);* B_{sess}*:* $C(v)$ *of [12], stable between semantic changes (KV BP2);* B_{cold}*: multi-level summary tree, included only for* cold sessions *(sessions resuming after the KV-cache TTL has expired, typically* ≥ 5 *minutes of inactivity, requiring the cached prefix to be re-uploaded);* B_{dyn}*: wisdom-program-selected context, per-turn.*

Theorem 3.5 (Context Stability). *The expected per-turn LM input cost is:*

$$\bar{C}_{turn} = 0.1(k_{\mathsf{perm}} + k_{\mathsf{sess}}) + k_{\mathsf{cold}} \cdot \mathbf{1}[cold] + k_{\mathsf{dyn}}$$

where stable blocks cost 10% of full price by the Byte-Identity Theorem [12]. As $T_c(v)/T_t \to \infty$*, the cached component approaches* $0.1(k_{\mathsf{perm}} + k_{\mathsf{sess}})$*—a* $10\times$ *reduction relative to paying full price on the stable prefix.*

Proof. By the Byte-Identity Theorem [12], B_{perm} and B_{sess} are byte-identical across turns between semantic changes; this makes them eligible for KV-cache reuse. The

KV-cache hit probability on B_{sess} equals $1 - T_t/T_c(v)$ under a Poisson change process, approaching 1 as $T_c(v) \gg T_t$. Cache-hit tokens cost 10% of full input price [1], independently of the byte-identity property. The cold block is amortized over the session; the dynamic block is always charged in full. □

4 WISDOM LIBRARY: COMPOSITION AND WIRING

4.1 Program Composition Correctness

Definition 4.1 (Phase-Correct Library). *$\mathcal{W}$ is* phase-correct *for $\mathcal{G}$ if: (a) for every $p, q \in \mathcal{W}$ with $\phi(p) \prec \phi(q)$ and any input $x \in I_p$, the merged input $x \oplus p(x) \in I_q$, where $\oplus$ denotes schema-compatible record merge (prior fields are preserved; q's required fields are provided by x or $p(x)$); (b) the aggregation program accepts the union of all outputs from programs with $\phi \prec$ `agg`; and (c) rendering programs accept the aggregation output.*

Theorem 4.2 (Program Composition Correctness). *Let $\mathcal{W}$ be phase-correct and let $p_1, \ldots, p_n \in \mathcal{W}$ be in valid phase order. If each p_i is individually correct ($\forall x \in I_i$, $p_i(x) \in O_i$), then the composed pipeline $P = p_1 ; \cdots ; p_n$ is correct.*

Proof. Induction on n. *Base*: $P = p_1$ is correct by hypothesis. *Step*: P_{n-1} is correct by inductive hypothesis, producing $y_{n-1} \in O_{n-1}$. Phase-correctness (condition a) gives $x_n = x \oplus y_{n-1} \in I_n$. Individual correctness of p_n gives $p_n(x_n) \in O_n$. □

Corollary 4.3 (Modular Extension). *If $\mathcal{W}$ is phase-correct and a new program p satisfies the phase-correctness conditions at addition time, then $\mathcal{W} \cup \{p\}$ is phase-correct and $P_{\mathcal{W} \cup \{p\}}$ is correct.*

Programs can be added to the wisdom library without breaking existing correctness, provided the addition protocol verifies phase-correctness. Combined with Accumulation Monotonicity [12], the library grows in both coverage and correctness simultaneously.

4.2 Declarative Wiring and Policy Graphs

Definition 4.4 (Policy Graph). *The* policy graph *of publisher ρ is a directed graph $\mathcal{P}_\rho = (H, E_\mathcal{P})$ where nodes $H = \{h_i\}$ are event handlers (policies, post-generation hooks, workflow steps, platform adapters), edges (h_i, h_j) denote output-triggers-input, and each h_i is annotated with: stream type pattern τ_i; event type $e_i \in \{$`stream.created`, `stream.updated`, `vote.cast`, `job.completed`, $\ldots\}$; and condition predicate $\gamma_i : Attrs \to \mathbb{B}$. Handlers are declared as typed stream nodes related to target streams via `Safebox/subscribes` relations—no imperative registration code.*

Theorem 4.5 (Declarative Wiring Soundness). *Assume SPACER's event queue provides at-most-once delivery per subscription (each (v, e, t) event is enqueued exactly once per matching handler by Rule* Trigger *[13]). Then for every stream event (v, e, t) within publisher ρ's scope: (i)* (Completeness) *every handler h_i with $\tau(v) \models \tau_i$, $e = e_i$, and $\gamma_i(\alpha(v)) =$ `true` is triggered exactly once; (ii)* (Soundness) *no handler with $\tau(v) \not\models \tau_j$, $e \neq e_j$, or $\gamma_j(\alpha(v)) =$ `false` is triggered.*

Proof. SPACER Rule Trigger [13] enqueues exactly the action invocations subscribed to v filtered by message type. `Safebox/subscribes` relations define the subscription set; γ_i is evaluated in Compute before any side effects—handlers for which $\gamma_i =$ `false` exit before Execute, producing no observable effect. Completeness: any (v, e, t) with matching (τ_i, e_i, γ_i) has h_i in the subscription set; Rule Trigger fires exactly one invocation per subscription; γ_i evaluates true so Execute proceeds. Soundness: non-matching handlers are not in the subscription set (by `Safebox/subscribes` construction); $\gamma_j =$ `false` handlers exit before Execute. □

Corollary 4.6 (Publisher Inheritance Soundness). *If ρ' inherits from ρ via `Safebox/inherits` and overrides handler h_i with h_i', Theorem 4.5 holds for ρ' with h_i replaced by h_i'.*

5 PROACTIVE STATE MACHINE INTELLIGENCE

Definition 5.1 (State Machine Position). *Let $H_T(v) = \langle e_1, e_2, \ldots, e_k \rangle$ be the sequence of attribute update events on stream v from time 0 to time T, ordered by timestamp, where each $e_i \in \Sigma_{attr}$ is an attribute condition that became satisfied at event i. The* position *of goal stream instance v at time T is $q_T(v) = \delta^*(q_0, H_T(v))$ where $\delta^* : Q \times \Sigma_{attr}^* \to Q$ is the standard extension of δ to sequences, defined by $\delta^*(q, \epsilon) = q$ and $\delta^*(q, \sigma \cdot e) = \delta(\delta^*(q, \sigma), e)$.*

Definition 5.2 (Proactive Advancement Condition). *A* proactive advancement condition *(q, γ, μ): $q \in Q$ is the target state; $\gamma : Attrs \times G^T \to \mathbb{B}$ a condition over stream attributes and induced graph; $\mu : Attrs \times G^T \to Messages$ a* proactive message generator *producing structured content (option arrays, governance affordances, clarification prompts) from graph state alone. When $q_T(v) = q$ and $\gamma(\alpha_T(v), G^T) =$ `true`, the system emits $\mu(\alpha_T(v), G^T)$ without awaiting user input.*

Definition 5.3 (Reactive Agent). *An agent policy π_R is* reactive *if it emits output only in response to user messages: for all times T, no output is emitted in any interval $[T, T + \epsilon]$ in which no user message arrives.*

Definition 5.4 (Proactive Agent). *An agent policy π_P is* proactive *if it emits output whenever either (a) a user message arrives, or (b) a proactive advancement condition $(q, \gamma, \mu) \in \Pi_\mathcal{G}$ is satisfied: $q_T(v) = q$ and $\gamma(\alpha_T(v), G^T) =$ `true`.*

Definition 5.5 (Turns-to-Terminal). *$\mathcal{N}^\pi(v)$: the number of interaction turns from q_0 until $q_T(v) \in F$ under agent policy π.*

Theorem 5.6 (Proactive Dominance). *Let π_R (reactive) and π_P (proactive) be agents for the same goal type $\mathcal{G}$, identical in LM response quality when they respond. For any goal stream instance v: $\mathbb{E}[\mathcal{N}^{\pi_P}(v)] \leq \mathbb{E}[\mathcal{N}^{\pi_R}(v)]$, with equality if and only if $\Pi_\mathcal{G} = \emptyset$ or every advancement condition in $\Pi_\mathcal{G}$ has $p_{\text{user}} = 1$ (i.e., every such condition is satisfied exclusively by user messages).*

Proof. Each transition condition c_i is satisfied either by (a) a user message, or (b) a background event (job completion, vote threshold, dependency resolution, elapsed time).

Under π_R (Definition 5.3): the reactive agent emits output only in response to user messages. If c_i is satisfied by a background event type (b) at time t_b strictly between two user messages at times $t_{\text{prev}} < t_b < t_{\text{next}}$, the reactive agent cannot emit an advancement message until t_{next}. This introduces at least one turn that would not appear under π_P: the user message at t_{next} (or a subsequent one) is needed to trigger π_R's response to the background event. More precisely, the expected number of turns between c_i becoming satisfiable and π_R acting on it is at least 1, since the reactive agent must wait for a user message.

Under π_P (Definition 5.4): if $(q, \gamma, \mu) \in \Pi_\mathcal{G}$ covers c_i, then immediately when $\gamma(\alpha_T(v), G^T)$ becomes true at time t_b, the system emits $\mu(\alpha_T(v), G^T)$ without any user message. The turn cost is zero additional user-message turns.

Summing over all conditions covered by $\Pi_\mathcal{G}$, the expected total turns $\mathbb{E}[\mathcal{N}^{\pi_P}]$ is reduced by at least $|\{i : c_i \text{ covered}\}|$ relative to $\mathbb{E}[\mathcal{N}^{\pi_R}]$. For uncovered conditions, both policies behave identically. Therefore $\mathbb{E}[\mathcal{N}^{\pi_P}] \leq \mathbb{E}[\mathcal{N}^{\pi_R}]$. □

Corollary 5.7 (Proactive Savings Bound). *Let n_P be the number of transition conditions covered by advancement conditions and p_{user} the probability that a condition is satisfied simultaneously by a user message. The expected turn savings is $\Delta\mathcal{N} \geq n_P(1 - p_{\text{user}})$. For background-event-dominated conditions, $p_{\text{user}} \approx 0$ and savings $\approx n_P$.*

6 ORGANIZATIONAL EFFICIENCY

Definition 6.1 (Turn Classification). *A turn in a multi-participant goal chat is:* progress *(modifies graph state or advances the state machine);* coordination *(communicates state, identifies a blocker, assigns responsibility, without modifying the graph);* governance *(casts a vote, records approval, registers a fork); or* exploratory *(belongs to a working branch thread). The* coordination overhead ratio *is* $\Omega = N_{\text{coord}}/(N_{\text{prog}} + N_{\text{gov}})$.

Theorem 6.2 (Coordination Overhead Structural Decomposition). *Let the coordination turns in any goal chat be partitioned into categories $C_1, \ldots, C_k$ (state-inquiry, blocker-identification, responsibility-assignment, vote-solicitation, etc.), with weight $w_i = N_{C_i}/N_{\text{coord}} \geq 0$ (the fraction of total coordination turns of category C_i), so $\sum_i w_i = 1$. For a proactive agent π_P whose advancement conditions and wisdom programs structurally preempt category C_i with coverage $\text{cov}(C_i) \in [0, 1]$, define $c_{\text{elim}} = \sum_i w_i \cdot \text{cov}(C_i)$. Then in any goal chat instance: $\Omega^{\pi_P} \leq \Omega^{\pi_R} \cdot (1 - c_{\text{elim}})$. For a mature system with complete coverage of C1–C4, $c_{\text{elim}} \to 1$ and $\Omega^{\pi_P} \to 0$.*

Proof. Coordination turns fall into four identifiable categories: C1 (state-inquiry: "What state are we in?"), C2 (blocker-identification: "What's blocking us?"), C3 (responsibility-assignment: "Who needs to act?"), and C4 (vote-solicitation: "Who hasn't voted?"). Under π_P: C1 is preempted by the session-join proactive advancement condition emitting the current state and missing transition requirements before any participant asks; C2 is preempted by a pre-phase wisdom program that detects blocking conditions after every graph mutation; C3 is preempted by deriving responsibility from the goal state machine's role requirements and emitting targeted prompts; C4 is preempted by the governance advancement condition emitting vote affordances when the tally is short specific participants. Each category C_i preempted with coverage coverage(C_i) contributes to $c_{\text{elim}} = \sum_i \text{coverage}(C_i) \cdot \text{weight}(C_i)$. For a mature library with full coverage of C1–C4, $c_{\text{elim}} \to 1$ and $\Omega^{\pi_P} \to 0$. □

Theorem 6.3 (Quality Preservation Under Proactivity). *Let $Q(v)$ be any quality metric that depends only on the content produced by deliberation turns (turns with routing decision* pass *to the LM), not on coordination turns. Then $\mathbb{E}[Q^{\pi_P}(v)] \geq \mathbb{E}[Q^{\pi_R}(v)]$.*

Proof. *Weak inequality.* By Theorem 5.6, π_P saves turns corresponding to proactive advancement conditions. Each such condition fires only when γ is satisfied by *current graph state*—not by LM generation. Content emitted by μ is a deterministic function of $\alpha(v)$ and $N(v)$; none of the saved turns contain LM-synthesized deliberation content. All LM calls for turns requiring genuine synthesis (routing decision pass) are present in both π_R and π_P interactions. Therefore the saved turns contribute zero expected quality to the terminal artifact, and removing them does not decrease $\mathbb{E}[Q]$: $\mathbb{E}[Q^{\pi_P}] \geq \mathbb{E}[Q^{\pi_R}]$.

Source of strict improvement (informally). Under π_R, blocking conditions (unconfigured dependencies, pending votes, unresolved ambiguities) that arise from background events may go undetected until a participant explicitly queries for them, potentially allowing artifacts to reach terminal state with unresolved issues. Under π_P, advancement conditions surface these structurally before any terminal transition fires. This provides an additional reason why strict inequality holds in practice for goal types with non-trivial governance, though formalizing this requires additional assumptions on the governance threshold model. □

Theorems 6.2 and 6.3 together establish the central organizational result: proactive agents are *Pareto improvements* over reactive agents—strictly faster to terminal state, equal

or higher artifact quality. No speed-quality tradeoff is incurred.

COROLLARY 6.4 (THREAD ISOLATION EFFICIENCY). *Exploratory conversations isolated to working branch threads cost $\Delta C_{\text{ctx}} = n_E \cdot \bar{\ell}_E \cdot k_t$ tokens per main-stream turn less than under reactive assistance (where n_E is the number of concurrent threads and $\bar{\ell}_E$ is the mean thread length in turns of k_t tokens each). Under reactive assistance, exploratory messages appear in the main stream and are included in its DYNAMIC context block at full token cost. Under proactive assistance with thread isolation, thread messages are posted to separate thread streams excluded from the main stream's SESSION block, contributing zero tokens to subsequent main-stream LM calls.*

COROLLARY 6.5 (TOTAL EFFICIENCY GAIN). *Treating the three efficiency mechanisms as acting on independent components of per-goal-completion cost (coordination turns, LM-call frequency, and per-call token cost respectively), the combined upper bound is: $C^{\pi_P} \leq C^{\pi_R} \cdot (1 - c_{\text{elim}}) \cdot (1 - E(\mathcal{W}^{(t)})) \cdot \frac{k_{\text{dyn}} + 0.1(k_{\text{perm}} + k_{\text{sess}})}{k_{\text{dyn}} + k_{\text{perm}} + k_{\text{sess}}}$ where the three factors capture coordination overhead elimination (Theorem 6.2), LM-call elimination (Grokers Accumulation Monotonicity [12]), and KV-cache savings (Theorem 3.5). The independence assumption is an approximation: in practice, c_{elim} and $E(\mathcal{W}^{(t)})$ interact since eliminated coordination turns also eliminate wisdom library invocations; the product formula provides a conservative upper bound under the approximation that the interactions are negligible relative to the dominant cost reductions.*

7 CROSS-PLATFORM RENDERING AND VOTE CONSISTENCY

DEFINITION 7.1 (OPTIONS ARRAY AND PLATFORM ADAPTERS). *The* options array $\mathcal{O}(v, q)$ *is produced by* post*-phase wisdom programs from $(\alpha(v), N(v), q)$. Each option specifies type, label, and structured payload.* Platform adapters *are* render*-phase wisdom programs with input $I_{\text{render}} = \{options : \mathcal{O}(v, q), session : \{platform, user\_id, locale\}\}$ and output O_{render} a platform-native message payload, that map $\mathcal{O}$ to platform-native representations: Telegram* `InlineKeyboardMarkup`*, Apple* `InteractiveMessage`*, Google* `SuggestionChip` + `RichCard`*, Email* `AMP` + `HTMLFallback`*, Web* `ChipBar`*. Adapters are declaratively wired to goal streams via* `Safebox/adapts` *relations—no imperative dispatch code.*

THEOREM 7.2 (CROSS-PLATFORM VOTE CONSISTENCY). *For all platforms $\rho \in \mathcal{P}$ and all orderings of concurrent votes: (i) $w_T(v, r, u)$ equals the sum of all votes cast by time T, regardless of originating platform; (ii) fork promotion fires if and only if $w_T \geq \theta$, exactly once.*

PROOF. *(i)* All platform-specific webhook handlers route to the single `Users_Vote` action type. SPACER Rule EXECUTE [13] applies delta $\Delta = \{w \mathrel{+}= \texttt{vote\_weight}\}$ in a single transaction. $\mathcal{MM}$'s Local Linearizability theorem [13] guarantees sequential consistency per stream; all votes are serialized by the append-only stream log. Platform of origin affects only the COMPUTE message format, not the serialized Δ.

(ii) After each vote, `Users_Total` is triggered by the `vote.cast` event (Theorem 4.5, completeness). By $\mathcal{MM}$'s per-publisher sequential consistency [13], all invocations of `Users_Total` for a given goal stream are serialized: no two invocations execute concurrently on the same publisher. This prevents the race condition in which two concurrent invocations both read `promotionFired` = `false` before either sets it. The serialized invocation checks $w_T \geq \theta$ in COMPUTE; on first crossing, emits the promotion event in CALL and sets `promotionFired` = `true` in EXECUTE. SPACER guarantees that all operations within a single EXECUTE phase are applied atomically in one database transaction [13]; therefore the weight update and the flag set are committed together, preventing any interleaved invocation from reading an intermediate state. Subsequent invocations read `promotionFired` = `true` in COMPUTE and exit without firing again. □

COROLLARY 7.3 (PLATFORM-AGNOSTIC GOVERNANCE). *The governance state of a goal stream is identical regardless of which platform participants use. Organizational teams operating across Telegram, email, web, and Apple Business Messages share a single governance ledger.*

8 DUAL-TRAVERSAL AND THE CACHE HIERARCHY

The Dual-Traversal Ordering Theorem of [12] establishes that top-down generation and bottom-up comprehension are the unique correct orderings for their respective tasks. We derive the connection to the context assembly hierarchy.

THEOREM 8.1 (HIERARCHY-CACHE CORRESPONDENCE). *In top-down artifact generation over DAG H with d depth levels: (i) Level-0 architecture occupies PERMANENT blocks (byte-identical across all leaf generations); section/module context occupies SESSION blocks (byte-identical within a section); leaf requirements occupy DYNAMIC blocks (per-leaf). (ii) Total generation cost: $\bar{C}_{\text{gen}} = \sum_\ell N_\ell \big(0.1 k_\ell^{\text{cached}} + k_\ell^{\text{dyn}}\big)$ versus $\sum_\ell N_\ell (k_\ell^{\text{cached}} + k_\ell^{\text{dyn}})$ without caching—savings on the cached component approach $10\times$ as $k_\ell^{\text{dyn}} / k_\ell^{\text{cached}} \to 0$, and decrease as dynamic context grows relative to cached context.*

PROOF. By Dual-Traversal Ordering [12]: level-0 architecture is generated first and assembled as the system prompt (PERMANENT). Section context is generated before its leaves and assembled as the SESSION block. Leaf requirements are dynamic. By the Byte-Identity Theorem [12], PERMANENT and SESSION blocks are byte-identical for all leaf generations within the same section. At 10% cache pricing, each is computed once and reused N_ℓ times at 10% cost. □

9 IMPLEMENTATION

The Context architecture is implemented across the Qbix / Safebox / Safebots stack.

Table 1: Magarshak Architecture Implementation Stack

Layer	Plugin	Key streams
Substrate ($\mathcal{MM}$)	Qbix	`Streams_Category`
		`Users_Vote`
Comprehension	Qbix	`streams_category`
(Grokers)	Safebox	`Action.propose`
Intelligence	Safebots	`Safebots/goal`
(Context)		`Safebots/dialog`
		`Safebots/thread`
		`Safebots/artifact`
Platform	Safebox	`Protocol.Telegram`
adapters	Protocols	`Protocol.Apple`
		`Protocol.Email`

Goal streams are created as `Safebots/goal` nodes. Wisdom programs are related to the goal via `Safebox/wisdom` typed relations (declarative wiring per Theorem 4.5). Platform adapters are selected by matching session platform metadata to `Safebox/adapts` relations. After every graph mutation, the postprocessing pipeline re-evaluates all $\Pi_{\mathcal{G}}$ conditions for the affected stream; non-null results emit proactive content immediately.

10 DISCUSSION

Pareto claim. Theorems 5.6 and 6.3 together establish a Pareto improvement: fewer turns, equal or higher quality. This is structural: proactive advancement conditions fire on graph state conditions, not LM generation triggers; the eliminated turns are provably coordination turns, not deliberation turns. This holds for the class of proactive agents defined here—not for all possible chatbots claiming to be "proactive."

Domain dependence. c_{elim} is small for open-ended creative collaborations; for structured task execution (capability building, document review, support resolution), we expect c_{elim} to be substantial, as C1–C4 coordination patterns are well-defined and mechanically preemptable; empirical measurement is left to future work (see the proposed methodology in §10).

Correctness assumptions. Theorem 4.2 assumes individually correct wisdom programs. Mitigation: governed write pipeline requires proposals before effects; phase-correctness validation runs at addition time; fitness-based evolution replaces low-quality programs.

Measurement methodology. The organizational efficiency theorems provide formal bounds but not empirical measurements. We propose: record goal chat sessions; classify turns by Definition of turn classification; compute Ω across sessions; compare π_R and π_P on identical goal type specifications. This methodology is the subject of ongoing work.

11 CONCLUSION

We have presented **Context**, the intelligence layer of the Magarshak Architecture, with six proved formal results establishing that: write-time context assembly reduces per-turn LM cost up to 10×; phase-ordered wisdom library composition preserves correctness under modular extension; declarative wiring is complete and sound; proactive agents weakly dominate reactive agents on turns-to-terminal-state; proactive agents are Pareto improvements in multi-participant goal-directed collaboration—faster to terminal state, equal or higher artifact quality; and cross-platform governance is consistent across all connected messaging surfaces. Together with the Magarshak Machine substrate [13] and the Grokers comprehension layer [12], the Context architecture provides a complete formal foundation for AI-assisted collaborative systems that are not merely responsive but actively intelligent, grow more capable and less expensive with every interaction, and make organizations provably more efficient at the work they are already trying to do.

ACKNOWLEDGMENTS

The Qbix, Intercoin, and Safebots open-source projects provided the implementation context for these results.

REFERENCES

[1] Anthropic. 2024. Prompt Caching. *Anthropic Documentation* (2024). https://docs.anthropic.com/en/docs/build-with-claude/prompt-caching

[2] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, et al. 2021. Program Synthesis with Large Language Models. *arXiv* arXiv:2108.07732 (2021).

[3] Erik Brynjolfsson, Danielle Li, and Lindsey R. Raymond. 2023. Generative AI at Work. *NBER Working Paper* 31161 (2023).

[4] Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, et al. 2019. Hello, GPT-2! How Close Are We to Human-Level Dialogue Systems?. In *ACL Workshop on NLP for Conversational AI*.

[5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, et al. 2021. Evaluating Large Language Models Trained on Code. *arXiv* arXiv:2107.03374 (2021).

[6] Yang Deng, Wenqiang Lei, Wai Lam, and Tat-Seng Chua. 2023. A Survey on Proactive Dialogue Systems: Problems, Methods, and Prospects. *arXiv* arXiv:2305.02750 (2023).

[7] Darren Edge, Ha Trinh, Newman Cheng, et al. 2024. From Local to Global: A Graph RAG Approach to Query-Focused Summarization. *arXiv* arXiv:2404.16130 (2024).

[8] Matthew Henderson. 2015. Machine Learning for Dialog State Tracking: A Review. In *Proceedings of The First International Workshop on Machine Learning in Spoken Language Processing*.

[9] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, et al. 2023. MetaGPT: Meta Programming for Multi-Agent Collaborative Framework. *arXiv* arXiv:2308.00352 (2023).

[10] Patrick Lewis, Ethan Petriv, Aleksandra Piktus, et al. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[11] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. CAMEL: Communicative Agents for "Mind" Exploration of Large Scale Language Model Society. *arXiv* arXiv:2303.17760 (2023).

[12] Gregory Magarshak. 2026. Grokers: Bottom-Up Inductive Comprehension and Write-Time Intelligence over Typed Knowledge Graphs. *arXiv* arXiv:2502.XXXXX (2026).

[13] Gregory Magarshak. 2026. The Magarshak Machine: A Stream-Partitioned Model for Governed State Evolution. The SPACER Framework: Streams, Policy, Actions, Capabilities, Execution, and Relations in Reactive Distributed Systems. *arXiv* arXiv:2501.XXXXX (2026).

[14] Shakked Noy and Whitney Zhang. 2023. Experimental Evidence on the Productivity Effects of Generative Artificial Intelligence. *Science* 381, 6654 (2023), 187–192.

[15] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2024. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation Framework. *arXiv* arXiv:2308.08155 (2024).

[16] Jun Xu, Haifeng Wang, Zhengyu Niu, Hua Wu, and Wanxiang Che. 2019. Conversational Graph Grounded Policy Learning for Open-Domain Conversation Generation. In *ACL*.

[17] Steve Young, Milica Gašić, Blaise Thomson, and Jason D. Williams. 2013. POMDP-based statistical spoken dialog systems: A review. *Proc. IEEE* 101, 5 (2013), 1160–1179.