THE GATEKEEPER KNOWS ENOUGH

Fikresilase W. Abebayew
BoA AI CoE
Fikresilase.Wondmeneh@bankofabyssinia.com

ABSTRACT

Large Language Models (LLMs) are increasingly deployed as autonomous agents, yet their practical utility is fundamentally constrained by a limited context window and state desynchronization resulting from the LLMs' stateless nature and inefficient context management. These limitations lead to unreliable output, unpredictable behavior, and inefficient resource usage, particularly when interacting with large, structured, and sensitive knowledge systems such as codebases and documents. To address these challenges, we introduce the Gatekeeper Protocol, a novel, domain-agnostic framework that governs agent–system interactions. Our protocol mandates that the agent first operate and reason on a minimalist, low-fidelity "latent state" representation of the system to strategically request high-fidelity context on demand. All interactions are mediated through a unified JSON format that serves as a declarative, state-synchronized protocol, ensuring the agent's model of the system remains verifiably grounded in the system's reality. We demonstrate the efficacy of this protocol with Sage, a reference implementation of the Gatekeeper Protocol for software development. Our results show that this approach significantly increases agent reliability, improves computational efficiency by minimizing token consumption, and enables scalable interaction with complex systems, creating a foundational methodology for building more robust, predictable, and grounded AI agents for any structured knowledge domain.

***K*eywords** State Synchronization · Progressive Contextualization · Declarative Protocol · AI Agents · Grounded AI · Human-AI Interaction

1 Introduction

The deployment of Large Language Models (LLMs) as autonomous agents for complex tasks like software development is rapidly advancing using the already existing techniques [1, 2]. However, their practical utility is fundamentally constrained by their LLM's stateless architecture and the context engineering design [3, 4] on the agent program. This core limitation leads to state desynchronization, where the agent's internal model diverges from the system's true state [5]. Consequently, agents exhibit unreliable behavior, hallucinate nonexistent entities [6], and inefficiently manage context, precluding their use in high-stakes, real-world applications.

In this work, we propose the Gatekeeper Protocol, a domain-agnostic framework that eschews ambiguous, conversational interaction and instead enforces a formal, state-synchronized communication layer between the agent and the system. Our protocol mandates that the agent first operate on a minimalist, low-fidelity "latent state" representation of the system. This forces the agent to reason strategically and request high-fidelity context only when necessary, rather than consuming the entire context.

All interactions are mediated through a unified JSON format that serves as a declarative, state-synchronized ledger. This mechanism ensures the agent's model of the system remains verifiably grounded in reality. This approach transforms the agent from an unpredictable conversationalist into a deterministic and reliable partner, significantly improving token efficiency and scalability.

2 Background

The journey for many modern agents began with the "reason-act" loop, a powerful idea popularized by frameworks like ReAct [1]. This approach allows an agent to "think out loud" before acting, which was a major leap forward. But this

simple loop has a critical weakness: an agent that can't remember its past is doomed to repeat its mistakes. This led to a wave of research focused on giving agents a memory. Architectures like those in Generative Agents [2] and Reflexion [7] introduced sophisticated memory streams and self-reflection abilities, allowing agents to learn from experience.
However, giving an agent a vast memory created a new, more subtle problem: how does it find the right memory at the right time? As surveys on the topic show [6], this has sparked an arms race in memory management techniques. We now have complex systems like MemGPT [3] and HiAgent [4] that treat an agent's context like a virtual memory hierarchy in an operating system. These are brilliant engineering solutions for preventing context overflow and cutting down on redundant information.
Yet, we argue that all these approaches share a common blind spot. They are hyper-focused on perfecting the information that goes into the LLM's black box, but they don't formalize the communication channel between the agent and the external world. The result is that the agent's understanding can still get out of sync with reality a problem so significant that frameworks like SyncMind [5] are now being built just to measure it. The consensus is that state synchronization is essential [8], but the field has yet to coalesce around a standard way to achieve it.
The Gatekeeper Protocol charts a different course. We believe that true agent reliability comes not from a more complex memory system, but from a simpler, more robust interaction protocol. Our contribution isn't a better memory architecture; it's a formal contract that governs how an agent is allowed to perceive and act upon a system. We position our work in three distinct ways.
First, we shift the focus from the "OS level" to the "API level." Instead of trying to manage the agent's internal memory (like MemGPT), we enforce a strict, declarative contract for every interaction using a unified JSON format. This makes every action an explicit, verifiable transaction, providing a clear audit trail of the agent's behavior.
Second, our protocol champions an "inference-first" philosophy, a stark contrast to the dominant "retrieval-first" model of RAG. The broader community is starting to agree that simply retrieving documents isn't enough, leading to calls for smarter "Context Engineering" [9, 10]. The Gatekeeper Protocol provides a concrete framework for this idea. We force the agent to reason on a cheap, high-level map before it's allowed to ask for the expensive, detailed documents. It has to think before it reads.
Finally, our protocol's declarative action space makes agents fundamentally safer. A ReAct-style agent might generate a `rm -rf` command, but a Gatekeeper agent can only state its intent—for example, `{"request": {"delete": {}}}`. This simple but critical distinction means a trusted system can safely interpret the intent, preventing a whole class of unpredictable and potentially harmful actions.

3 The Gatekeeper Protocol

3.1 Architectural Overview

The protocol's architecture is defined by the manipulation of a single, central data structure: the **System State-Context Representation (SCR)**, denoted as $\mathcal{L}$. This unified JSON object serves three simultaneous roles:

1. **A Latent Context Map:** It provides a high-level, potentially low-fidelity, structural representation of the system.
2. **A State Record:** It is the authoritative, ground-truth record of the system's state at a given time.
3. **An Action Interface:** The agent proposes actions by modifying specific fields within this object.

The interaction is a discrete, time-stepped cycle (Figure 1). At each step t, the agent's policy, π_{agent}, receives the current SCR, $\mathcal{L}_t$. Based on a given task T, it generates a proposed modification, $\mathcal{L}'_t$, which encodes a desired action A_t. The system executes the action, producing a new SCR, $\mathcal{L}_{t+1}$, which begins the next cycle.

3.2 Latent Context and Progressive Contextualization

To ensure token efficiency, the protocol's SCR begins as a low-fidelity latent map. Components within the SCR can have placeholder values (e.g., `"unsummarized"`). The agent operates on an "inference-first" principle, using this structural information to reason about the task before requesting high-fidelity content.

This process of deciding when to request more context is modeled as a policy, π_{agent}, that seeks to maximize the expected value of future states while minimizing the cost of information retrieval. At each step t, the agent chooses a set of unsummarized components, C, to query via a `provide` action. This choice, $A_t(C)$, is determined by:

$$A_t(C) = \pi_{\text{agent}}(\mathcal{L}_t, T) = \underset{C \subseteq \mathcal{S}_{\text{unsum}}}{\operatorname{argmax}} \left[\mathbb{E}[V(\mathcal{L}_{t+1} | \mathcal{L}_t, A_t(C))] - \lambda \cdot \text{Cost}(C) \right] \tag{1}$$

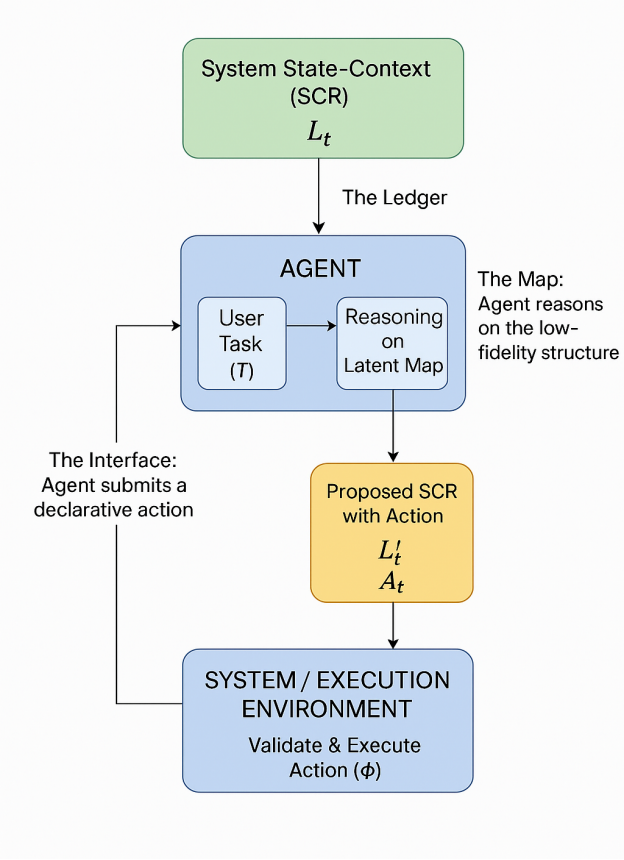

Figure 1: The Gatekeeper Protocol cycle: $\mathcal{L}_t \xrightarrow{\text{Agent Policy } \pi_{\text{agent}}} \mathcal{L}'_t(\text{containing } A_t) \xrightarrow{\text{System Execution } \Phi} \mathcal{L}_{t+1}$

Where:

- $\mathcal{S}_{\text{unsum}}$ is the set of all components with unsummarized content in $\mathcal{L}_t$.
- $\mathbb{E}[V(\mathcal{L}_{t+1}|\dots)]$ is the expected value (or utility) of the next state, given the current state and the chosen action. This represents the agent's belief about how much the new information will help in solving task T.
- $\text{Cost}(C)$ is the token cost associated with retrieving and processing the content of the components in C.
- λ is a hyperparameter that balances the trade-off between information gain and token cost.

3.3 System State Representation and Synchronization

To eliminate state desynchronization, the protocol enforces that the SCR, $\mathcal{L}$, is the single source of truth. The agent's understanding of the system is not an internal belief but is entirely represented by the SCR it possesses.

Synchronization is achieved through a deterministic state transition function, Φ, executed by the system. The agent proposes an action by modifying the `request` fields in its current SCR, $\mathcal{L}_t$, to create a proposed state change, $\mathcal{L}'_t$. The system validates and executes this proposal. The state transition is defined as a piecewise function:

$$\mathcal{L}_{t+1} = \begin{cases} \Phi(\mathcal{L}_t, \mathcal{L}'_t) & \text{if IsValid}(\mathcal{L}'_t, \mathcal{L}_t) \\ \mathcal{L}_t & \text{otherwise} \end{cases} \tag{2}$$

Where:

- $\text{IsValid}(\mathcal{L}'_t, \mathcal{L}_t)$ is a validation function that returns true if the action encoded in $\mathcal{L}'_t$ is permissible given the current state $\mathcal{L}_t$.
- $\Phi(\mathcal{L}_t, \mathcal{L}'_t)$ is the execution function that applies the valid changes and returns the new, ground-truth SCR.

This mechanism guarantees that the agent's state is always synchronized with the system's state, as any invalid or failed action results in the state remaining unchanged, preventing state drift.

3.4 The Declarative Action Interface

The SCR itself serves as the action interface. To ensure safety and uniformity, the agent's actions must be **declarative**, not imperative. The action space $\mathcal{A}$ is a finite set of intents (e.g., `provide`, `edit`, `write`, `delete`).

An action A_t is encoded by populating the `request` field of one or more components within the proposed SCR, $\mathcal{L}'_t$. For example, an intent to delete a component `s` is encoded as setting the `request` field of `s` in $\mathcal{L}'_t$ to `"{"delete": {}}"`. This declarative model decouples the agent's intent from the system's execution logic, allowing for a trusted layer to safely validate, log, and perform the requested state transition.

4 Empirical Evaluation

To rigorously test the robustness and generalizability of the Gatekeeper Protocol, we conducted a comprehensive empirical evaluation. We compared our protocol against four common context management strategies across a suite of three diverse programming tasks, implementing each strategy with seven different LLMs to ensure our findings were model-agnostic. For our protocol's implementation, we used **Sage**, our open-source agent publicly available on GitHub[1].

4.1 Experimental Design

Tasks. To avoid overfitting to a single problem type, our evaluation suite included three distinct tasks:

1. **Python Refactoring (Stateful):** A multi-file function renaming task testing state tracking and dependency management.
2. **Frontend Component Creation (Structured):** A task to build a new React component using a well-defined library (Next.js with Shadcn/ui).
3. **Python Web Scraping (Exploratory):** A task to write a new script to scrape data from a live, previously unseen website.

Models and Strategies. Each task was run with seven models (Google: Gemini 2.0 Flash Experimental, Qwen: Qwen3 Coder 480B A35B, DeepSeek: R1 Microsoft: MAI DS R1, OpenAI: gpt-oss-20b, Mistral: Mistral Small 3.2 24B and NVIDIA: Nemotron Nano 9B V2) across five strategies: (1) Full Codebase, (2) Recent Files, (3) RAG, (4) ReAct Agent, and (5) Sage (Gatekeeper Protocol).

Baseline Fairness. To ensure a fair comparison, all baselines were implemented with standardized prompts. Our RAG implementation used cosine similarity over 512-token chunks from a ChromaDB vector store. The ReAct agent was given an identical imperative toolset to Sage.

4.2 Metrics

We defined three primary metrics, averaged over $R = 7$ model runs for each task, and then averaged across all three tasks.

Average Task Completion Progress (%). For a task decomposed into K sub-tasks, let $c_{i,j}$ be a binary variable for the completion of sub-task j in run i. The average progress is:

$$\text{Progress}_{\text{avg}} = \frac{100}{R \cdot K} \sum_{i=1}^{R} \sum_{j=1}^{K} c_{i,j} \tag{3}$$

Average Grounding Errors. Let E_i be the count of grounding errors (actions based on a false state belief) for run i. The average is:

$$\text{GE}_{\text{avg}} = \frac{1}{R} \sum_{i=1}^{R} E_i \tag{4}$$

[1] `https://github.com/Fikresilase/sage`

Average Total Tokens. This metric measures the cumulative sum of all input and output tokens for the entire duration of a task run, providing a holistic measure of computational cost.

4.3 Quantitative Results

The aggregated results, averaged across all models and tasks, are presented in Table 1.

Table 1: Comparative Performance of Context Management Strategies (Averaged Across 3 Tasks and 7 LLMs). Values are mean ± standard deviation.

Strategy	Avg. Task Completion (%)	Avg. Grounding Errors	Avg. Total Tokens
Full Codebase	48% ± 18%	4.3 ± 2.1	19,100 ± 3500
Recent Files	31% ± 22%	5.8 ± 2.5	9,800 ± 4100
RAG	58% ± 15%	3.1 ± 1.5	14,300 ± 2800
ReAct Agent	55% ± 17%	3.4 ± 1.8	15,200 ± 3100
Sage (Gatekeeper)	**73% ± 8%**	**0.8 ± 0.4**	**6,200 ± 1200**

As shown in Table 1, the Gatekeeper Protocol consistently outperforms all baselines. Sage achieved an average task completion of 73%, a notable improvement over the next best strategy, RAG, at 58%. Crucially, the low standard deviation (±8%) indicates a high degree of reliability across different models and tasks. It also committed an order of magnitude fewer grounding errors and was over twice as token-efficient.

4.4 Qualitative Analysis and Discussion

Our observations revealed a direct relationship between a codebase's conventionality and the Gatekeeper Protocol's token efficiency. In the highly structured Next.js/Shadcn task, Sage's initial latent map was remarkably accurate, allowing it to solve the task with minimal 'provide' requests and thus very low token usage. Conversely, in the bespoke web scraping task, which required understanding a unique and unknown file structure, Sage was more cautious. It issued more 'provide' requests to build up its high-fidelity context, leading to an increase in token consumption. This demonstrates that the protocol's efficiency is proportional to the "common knowledge" embedded in the system's structure, a feature that allows it to dynamically adapt its cost to the complexity of the task. This adaptive behavior was absent in the baseline models, which consumed high token counts regardless of task structure.

4.5 Threats to Validity

We acknowledge several threats to validity. Our evaluation, while diverse, was confined to three tasks. A larger benchmark suite is needed for full generalization. The performance of RAG is highly sensitive to chunking and embedding strategies, and different configurations might yield different results. However, we believe the consistency of the Gatekeeper Protocol's superior performance across a diverse model set provides strong evidence for its general effectiveness.

5 Discussion

Our results show that a formalized interaction protocol is a more significant driver of agent reliability than the choice of the underlying LLM. The consistent outperformance of our agent, Sage, is not an accident of the model but a direct consequence of the Gatekeeper Protocol's architecture.

5.1 Analysis of Findings

The protocol's success stems from three principles. First, the **inference-first** model of contextualization forces the agent to reason on a cheap, low-fidelity map before requesting expensive, high-fidelity context. This minimizes grounding errors by ensuring the agent acts only on a relevant, validated subset of the knowledge base.

Second, the **transactional nature of state synchronization** provides a powerful guardrail against context drift. While ReAct agents often wasted resources re-reading files to re-establish state, our agent was guaranteed a fresh, ground-truth representation after every action. This eliminated the need for costly re-validation and was the primary driver of its high task completion rate.

Finally, the protocol effectively **scaffolds the reasoning process** of diverse LLMs. By imposing a correct and efficient procedure, the Gatekeeper's structure compensates for some of the raw reasoning deficiencies of the underlying models, guiding them toward a valid solution.

5.2 Limitations of the Protocol

Despite its strong performance, the protocol has clear boundaries.

1. **It requires structured knowledge.** The protocol's reliance on a structural "map" makes it unsuitable for monolithic, unstructured data sources.
2. **It introduces latency.** The reliability gained from its multi-turn, transactional nature comes at the cost of speed compared to one-shot generation.
3. **It is bounded by LLM reasoning.** The protocol can guide a confused agent, but it cannot fix a fundamental inability to reason about a task. Agent performance is ultimately limited by the core intelligence of the model.
4. **Initial analysis can be costly.** Generating the latent map for extremely large systems could be computationally intensive, an area for future optimization.

5.3 Broader Impact

While tested on code, the Gatekeeper Protocol is a domain-agnostic methodology. Its core insight is that for high-stakes professional work, raw generative power is insufficient. The future of reliable AI assistance lies in shifting focus from developing more complex internal memories to designing formal, structured interaction protocols. This provides a foundational pathway toward building autonomous agents that are predictable, verifiable, and ultimately, trustworthy.

6 Conclusion

This paper confronted the critical challenges of state desynchronization, context management, and grounding that limit current LLM agents. We argued that the solution lies not in better memory, but in a better protocol for interaction.

We introduced the Gatekeeper Protocol, a framework built on a latent context map, a synchronized state ledger, and a declarative action space. Our empirical evaluation showed that this protocol delivers substantial and consistent improvements in agent reliability and efficiency across a diverse suite of language models, strongly suggesting that architecture, not the model, is the primary driver of robust performance.

The work presented here lays a foundational methodology, but the path forward is rich with possibilities. A key direction for future work is the development of a hierarchical latent map, or "latent map tree," to further enhance scalability. In this model, a provide action on a high-level segment (e.g., a folder) would return not raw content, but a more detailed, lower-level latent map for that segment. This would allow the agent to recursively navigate massive knowledge systems, progressively increasing the resolution of its context only where necessary. This structural enhancement could be combined with advanced reasoning techniques like model chaining or query transformation. Moreover, there is a clear need to refine these principles into a universal Gatekeeper specification a truly domain-agnostic protocol that can be implemented across a variety of fields with minimal adaptation.

Ultimately, the Gatekeeper Protocol offers a crucial pathway toward building autonomous systems that are powerful, predictable, and safe enough for high-stakes, real-world applications.

References

[1] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[2] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. *arXiv preprint arXiv:2304.03442*, 2023.

[3] Charles Packer, Vivian Fang, Shishir G. Lin, Kevin Gg, Joseph E. Gonzalez, Ion Stoica, and Michael I. Jordan. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

[4] Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. *arXiv preprint arXiv:2402.09559*, 2024.

[5] Xuehang Guo, Xingyao Wang, Yangyi Chen, Sha Li, Chi Han, Manling Li, and Heng Ji. Syncmind: Measuring agent out-of-sync recovery in collaborative software engineering. *arXiv preprint arXiv:2502.06994*, 2025.

[6] Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model based agents. *arXiv preprint arXiv:2404.13501*, 2024.

[7] Noah Shinn, Beck Labash, and Ashwin Gopinath. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023.

[8] Victor de Lamo Castrillo, Habtom Kahsay Gidey, Alexander Lenz, and Alois Knoll. Fundamentals of building autonomous llm agents. *arXiv preprint arXiv:2510.09244*, 2025.

[9] Anthropic. Effective context engineering for ai agents. Anthropic Engineering Blog, 2024.

[10] Jeff Huber. Rag is dead, context engineering is king. Latent.Space Blog, 2024.