An Explicit Memory-Driven Agentic Framework for Power System Simulation

Anonymous Authors[1]

Abstract

In rigorous domains like power system simulation, Foundation Models (FMs) relying on parametric memory or fine-tuning suffer from confident hallucinations, catastrophic forgetting, and privacy risks. To address this, we propose a decoupled, explicit memory-driven agentic framework that restricts the FM to a pure reasoning engine. By offloading knowledge, state, and skill management to external modules, our architecture facilitates more predictable execution and zero-shot skill reuse without weight updates. Experimental results show our approach achieves 87.35% code fidelity accuracy, demonstrating improved performance over parametric baselines and maintaining stability in complex tasks.

1. Introduction

The memorization phenomenon in Foundation Models (FMs), the internalization of training corpora into parametric memory, enables zero-shot generation but often exhibits reliability issues in rigorous engineering domains like power system simulation (e.g., MATPOWER (Zimmerman et al., 2011)). Applying FMs to such workflows reveals two critical vulnerabilities: 1) Parametric misalignment, where general MATLAB paradigms conflict with niche constraints, leading to plausible but incorrect outputs (hallucinations); 2) Working memory dilution, where the context window of the Large Language Model (LLM) is saturated with execution logs during closed-loop debugging, displacing user instructions and compromising semantic integrity.

Traditionally, mitigating these deficiencies relies on Supervised Fine-Tuning (SFT) or Reinforcement Learning from Human Feedback (RLHF). However, these methods exacerbate the privacy-generalization dilemma, exposing confidential grid topologies to data extraction attacks and creating a bottleneck for self-evolution.

To address these gaps, in (Yao et al., 2023), the authors employ the ReAct approach to solve the lack of synergy between reasoning and acting. In (Lewis et al., 2021), the authors adopt Retrieval-Augmented Generation (RAG) to solve the limitations of purely parametric models. In (Packer et al., 2024), the authors propose MemGPT, a system adopting virtual context management to solve context constraints through hierarchical memory tiers. In (Xiang et al., 2026), the authors propose a taxonomy for self-evolving agents, highlighting environment-centric evolution as a key paradigm to solve the scalability bottleneck of human-guided supervision. In (Hu et al., 2026), the authors propose MemoryAgentBench to solve the under-evaluation of memory mechanisms, revealing that existing agents still struggle with core competencies such as selective forgetting and test-time learning. In (Wang et al., 2026), the author develops an LLM agent that automates MATPOWER static analysis using a DeepSeek-OCR enhanced RAG and an error-correction system. While successful for isolated tasks, the model suffered from catastrophic forgetting of physical constraints in multi-turn dialogues and could not retain problem-solving experiences without weight updates.

To overcome these challenges, this paper presents an explicit memory architecture. By externalizing memory mechanisms, the FM is restricted to acting purely as a reasoning engine, thereby replacing unreliable parametric memory with structured, system-level, explicit memory. The main contributions of this paper are:

- **A Decoupled Explicit Memory Architecture:** A framework is proposed to externalize state, knowledge, and procedural skills from the FM. Mapping natural language to predictable grid states effectively mitigates parametric hallucinations and privacy risks.
- **Non-parametric Skill Evolution & Grounding Mechanism:** Mandatory citations and a self-evolving skill manager are introduced to enable continuous learning without weight updates. This addresses the generalization-memorization trade-off and facilitates machine unlearning.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by *The Impact of Memorization on Trustworthy Foundation Models* Workshop @ ICML. Do not distribute.

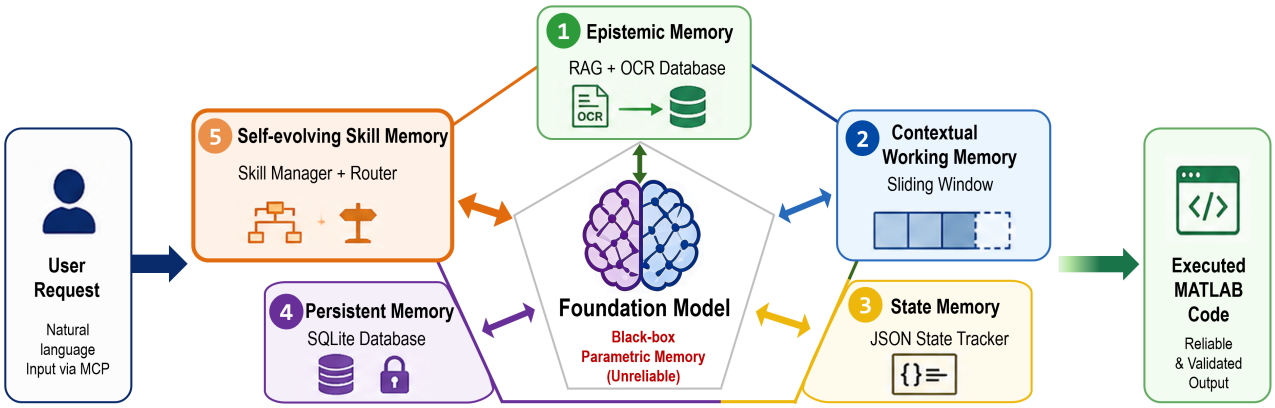

Figure 1. Explicit Memory Architecture encapsulated as an MCP Server

2. The Explicit Memory-Driven Agentic Framework

To mitigate the unreliability of FMs, an agentic framework is proposed that integrates a robust execution engine with the Explicit Memory Architecture, as illustrated in Figure 1. Instead of relying on the FM's internal weights, knowledge, state, and debugging experience are offloaded to external.

Formally, this decoupled reasoning process can be modeled as an extended state transition system. Let U_t denote the user instruction at turn t, and $\mathcal{K}$ represent the external epistemic database (e.g., MATPOWER manuals(Zimmerman & Murillo-Sanchez, 2025a)). To avoid parametric hallucinations, the generation of the simulation code C_t is strictly constrained by an Explicit Memory State Σ_{t-1}. The process aims to maximize the conditional probability:

$$C_t = \arg\max_{C} P_{FM}\left(C \mid U_t, \Sigma_{t-1}, \text{Ret}(U_t, \mathcal{K})\right) \quad (1)$$

where $\text{Ret}(\cdot)$ is the epistemic retrieval function. Furthermore, to maintain consistency across multi-turn dialogues, the semantic belief state Σ is iteratively updated via a deterministic mapping function δ:

$$\Sigma_t = \delta(\Sigma_{t-1}, \text{Extract}(H_t)) \quad (2)$$

where H_t represents the validated execution history at turn t. This formal separation of Σ_t and $\mathcal{K}$ from the FM's internal parameters forms the theoretical foundation of architecture.

2.1. The Base Execution Engine

At its core, the agent operates within a closed-loop MATLAB execution sandbox connected via the Model Context Protocol (MCP) (Anthropic, 2024). To prevent the FM's parametric hallucinations from causing runtime crashes, a defensive pipeline is implemented: **1) Static Pre-check:** Domain-specific constraints are automatically injected before execution. **2) Dynamic Feedback Loop:** MATLAB stack traces are captured to iteratively prompt the FM for error correction. **3) Semantic Validator:** This acts as an alignment guardrail. Even if the FM generates syntactically valid code by utilizing its internal memory, the Validator verifies that the code logically aligns with the user's initial prompt, detecting instances where the FM forgets physical constraints to achieve a successful execution.

2.2. Epistemic and Contextual Working Memory

To ensure the FM relies on verified facts rather than its internal parametric memory, an epistemic memory module is established. DeepSeek-OCR (Wei et al., 2025) is utilized to construct a highly structured Markdown vector database from MATPOWER manuals. Building upon this, Mandatory Citations are introduced: the FM is strictly prompted to append a flag and link its generated code to specific retrieved chunks, prompting the model to rely on external epistemic memory rather than overconfident, potentially obsolete parametric memory.

Simultaneously, to counteract the dilution of attention during the dynamic feedback loop, a sliding window mechanism is implemented. This forms a contextual working memory that filters out obsolete trial-and-error noise from MATLAB logs. It ensures the FM remains focused on the immediate task and successfully resolves coreferences without context saturation.

2.3. State Persistence and Procedural Skill Evolution

In long-turn dialogues, FMs are highly prone to catastrophic forgetting of early physical constraints. This is resolved by introducing a semantic belief state memory. After every successful execution, a State Extractor Agent compresses the interaction history into a deterministic format. This state is systematically injected into the system prompt of the next turn, establishing a state anchor that helps mitigate

long-term forgetting.

To further address privacy and the generalization-memorization trade-off, this architecture incorporates system-level evolutions.

Persistent Privacy: The extracted states are stored in an external SQLite database keyed by session ID, physically isolating data to defend against extraction attacks and facilitating Machine Unlearning.

Skill Evolution: When the dynamic feedback loop struggles but eventually succeeds, a Skill Manager abstracts the trajectory into a Markdown Skill. A Skill Router retrieves this procedural memory for future identical tasks, bypassing tedious feedback loops and learning debugging expertise without updating FM weights.

3. Case Study

To evaluate the proposed Memory Architecture, a series of grid analysis tasks is conducted on standard IEEE test systems (Case 14, Case 30, and Case 39).

3.1. Experimental Setup

The Memory Architecture is implemented as an MCP Server (mcp, 2024) using DeepSeek-V4 (DeepSeek-AI, 2026). Power system simulations are performed in MATLAB R2025b (The MathWorks Inc., 2025) with MATPOWER 8.1 (Zimmerman & Murillo-Sanchez, 2025b). To evaluate the proposed architecture, three comparative configurations are established.

M1 (Naive FM): A zero-shot configuration that uses pure parametric memory without RAG or feedback.
M2 (Isolated Agent): Features RAG and feedback but operating as a stateless, single-turn executor.
M3 (Full Agent): The proposed architecture with all five explicit memory modules.

A test suite of 10 tasks of varying complexity is utilized. It includes 5 Easy tasks and 5 Hard tasks. Performance is quantified using the **Code Generation Fidelity (CSGF)** and **Global CSGF Accuracy (GCA)** metrics defined in (Wang et al., 2026).

$$CSGF_i = S_i \times \left(\frac{N_{threshold} - (n_i - 1)}{N_{threshold}} \right) \quad (3)$$

where semantic score $S_i \in \{0, 0.8, 1.0\}$, iteration count n_i , and maximum feedback iterations $N_{threshold} = 5$. GCA measures the average correctness across K tasks via RMSE.

$$GCA = \left(1 - \sqrt{\frac{1}{K} \sum_{i=1}^{K} (1 - CSGF_i)^2} \right) \times 100\% \quad (4)$$

3.2. Quantitative Performance Overview

The performance of all configurations across the benchmark is summarized in Table 1.

Table 1. Quantitative Benchmark Results

AGENT	EASY TASKS (AVG. CSGF)	HARD TASKS (AVG. CSGF)	GCA (%)
M1	0.40	0.00	10.56
M2	0.92	0.64	63.67
M3	**0.96**	**0.88**	**87.35**

The quantitative results highlight the limitations of pure parametric memory. The baseline M1 failed in complex scenarios, resulting in a GCA of only 10.56%. While the M2 (Isolated Agent) achieved reliable performance on easy tasks, its proficiency plummeted to 0.64 on hard tasks due to a lack of procedural memory. The proposed M3 (Full Agent) achieved a GCA of 87.35%, outperforming M2 by 23.68%. As depicted in Figure 2, while the isolated agent (M2) exhibits a catastrophic blind spot on the most complex structural task, M3 maintains a fully robust capability boundary. By proactively retrieving self-evolving skills, M3 effectively addresses this case, achieving zero-shot execution. This demonstrates that explicit memory evolution helps prevent catastrophic debugging failures, demonstrating a more feasible solution compared to model fine-tuning for specialized grid analysis.

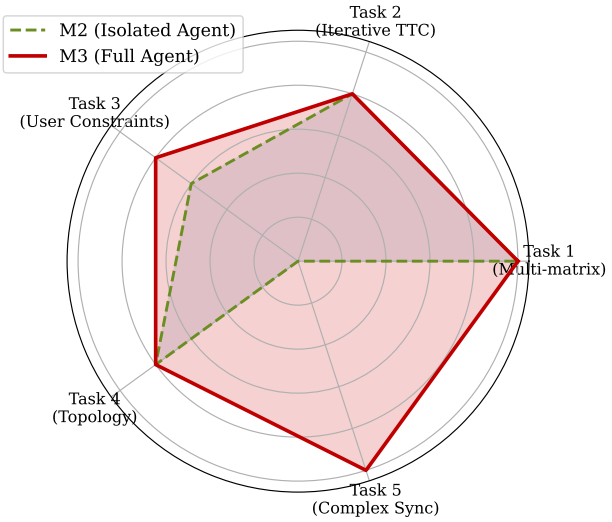

Figure 2. CSGF Capability Boundaries on Hard Tasks

3.3. Transition from Isolated to Continuous Simulation

This part evaluates the agent's ability to combat catastrophic forgetting during prolonged multi-turn interactions.

Scenario

Turn 1: User instructs the agent to modify the active power limit on bus 2 ($P_{max} = 50MW$).

Turn 2: User requests an Optimal Power Flow (OPF) based on the modified grid.

⋮

Turn n: After several unrelated technical queries to exceed the working memory window, the user requests a contingency analysis on the same modified grid.

Observations

In the baseline M2, since each request is processed in isolation, the agent in Turn 2 has no awareness of Turn 1. It generates code loading the default case30, resulting in a semantic failure ($S_i = 0$). In contrast, M3 utilizes its Semantic Belief State tracker to extract the P_{max} constraint into a state and persists it in SQLite. In Turn 2, the agent retrieves this state, generating code that correctly preserves the modifications and achieves a maximum CSGF score. This highlights how explicit memory transforms a stateless model into a persistent simulation agent. At Turn n, even though the original instruction has been displaced from the model's contextual window, the agent still re-injects the persisted state from the database, ensuring the physical consistency of the simulation remains intact throughout the entire session.

3.4. Self-Evolving Skill Acquisition

This part highlights the efficacy of Self-evolving Skill Memory in bypassing repetitive trial-and-error cycles. The detailed workflow of this transition is depicted in Figure 3.

Scenario: The agent is assigned a complex structural task that requires synchronized modifications across interdependent technical arrays. In the target power system environment, such operations are prone to logical failures because foundation models often lack the deep structural awareness needed to maintain consistency between disparate but related data components (e.g., aligning technical specifications with their corresponding economic parameters).

Observations

Initial Encounter: During the first attempt, the foundation model relies on its unreliable parametric memory, leading to an asymmetric modification error. While the model successfully updates the primary technical parameters, it fails to synchronize the auxiliary structural data, causing a critical execution failure. The Dynamic Feedback Loop intercepts this error, capturing the diagnostic stack trace and guiding the model through multiple iterative corrections. Through this closed-loop process, the agent eventually "discovers" the necessary synchronization logic and achieves a validated success.

Skill Acquisition and Zero-shot Reuse: Once the task is successfully validated, the Skill Manager automatically abstracts the failure-to-success trajectory into a permanent Self-evolving Skill. In a subsequent separate session involving a similar structural requirement, the Skill Router proactively retrieves this acquired expertise and injects it into the reasoning context. Consequently, the agent avoids the previously encountered logical pitfalls, enabling accurate zero-shot execution.

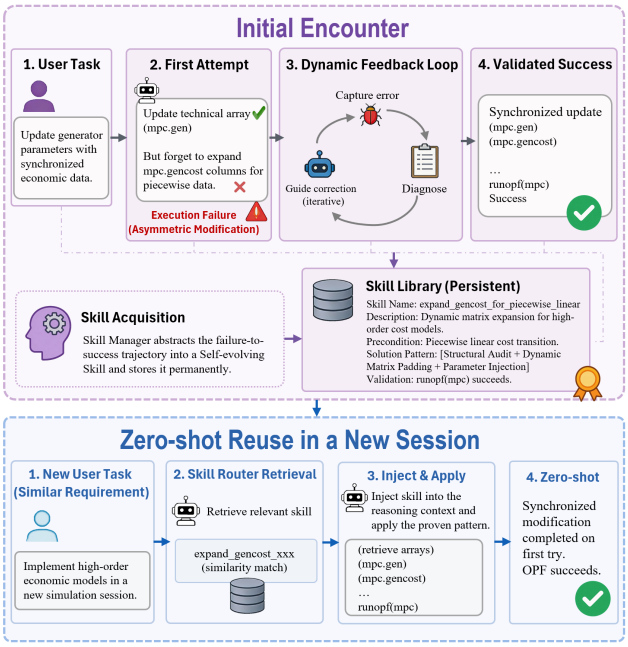

Figure 3. Detailed workflow of Self-evolving Skill Acquisition. The agent abstracts a successful structural synchronization logic from a multi-iteration debugging trajectory and achieves zero-shot proficiency in a subsequent, independent session.

4. Conclusion

This paper presents an Explicit Memory Architecture that offloads knowledge management from FMs to external modules, aiming to mitigate hallucinations and privacy risks in power system simulations. By integrating epistemic grounding, state persistence, and skill evolution, our framework maintains consistency across long-turn dialogues without weight updates. Experiments show an 87.35% GCA, outperforming baselines by 23% and reducing repetitive debugging loops. This decoupled design provides structural protection for industrial data and facilitates machine unlearning via database deletions. Future work includes extending the framework to dynamic simulations and multi-agent systems.

References

Model context protocol. https://github.com/modelcontextprotocol, 2024.

Anthropic. Introducing the model context protocol. https://modelcontextprotocol.io, 2024. Accessed: 2026-03-10.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026.

Hu, Y., Wang, Y., and McAuley, J. Evaluating memory in llm agents via incremental multi-turn interactions, 2026. URL https://arxiv.org/abs/2507.05257.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., tau Yih, W., Rocktäschel, T., Riedel, S., and Kiela, D. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021. URL https://arxiv.org/abs/2005.11401.

Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S. G., Stoica, I., and Gonzalez, J. E. Memgpt: Towards llms as operating systems, 2024. URL https://arxiv.org/abs/2310.08560.

The MathWorks Inc. Matlab version: 25.2.0.3042426 (r2025b) update 1, 2025. URL https://www.mathworks.com.

Wang, Q., Yang, S., and Zhu, Y. Agentic application in power grid static analysis: Automatic code generation and error correction. In *2026 the 9th International Conference on Energy, Electrical and Power Engineering (CEEPE 2026)*, April 2026.

Wei, H., Sun, Y., and Li, Y. Deepseek-ocr: Contexts optical compression, 2025. URL https://arxiv.org/abs/2510.18234.

Xiang, Z., Yang, C., Chen, Z., Wei, Z., Tang, Y., Teng, Z., Peng, Z., Li, Z., Huang, C., He, Y., et al. A systematic survey of self-evolving agents: From model-centric to environment-driven co-evolution. 2026.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. React: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

Zimmerman, R. D. and Murillo-Sanchez, C. E. *MATPOWER User's Manual, Version 8.1*, 2025a. Online.

Zimmerman, R. D. and Murillo-Sanchez, C. E. Matpower, 2025b. URL https://matpower.org. Free, open-source tools for electric power system simulation and optimization.

Zimmerman, R. D., Murillo-Sanchez, C. E., and Thomas, R. J. Matpower: Steady-state operations, planning and analysis tools for power systems research and education. *IEEE Transactions on Power Systems*, 26(1):12–19, 2011. doi: 10.1109/TPWRS.2010.2051168.