The Co-Forgetting Protocol: Synchronized Memory Management for Multi-Agent Systems

Anonymous Author

May 2025

Abstract

Efficient memory management in multi-agent systems (MAS) is vital for ensuring seamless coordination and minimizing latency caused by outdated data. The Co-Forgetting Protocol introduces a synchronized approach to memory management, utilizing a quorum-based voting mechanism, multi-scale decay functions, and meta-learned thresholds. Memories are maintained in a centralized vector database, with agents collaboratively deciding which to retain or discard after each epoch. Fault-tolerant consensus, such as PBFT, ensures reliability in untrusted settings, while enhancements like pre-filtering and dynamic weights improve scalability. Evaluations on benchmarks like AgentBench show significant improvements in task success rates and memory efficiency. This paper provides a detailed framework for implementing and understanding the protocols benefits in distributed AI systems.

Contents

1 Introduction

Multi-agent systems (MAS) are foundational to applications like autonomous robotics, distributed reinforcement learning, and collaborative AI frameworks. These systems depend heavily on shared memory to store contextual data, enabling agents to coordinate effectively. However, memory management in MAS faces critical challenges:

- **Data Consistency**: Unsynchronized operations can lead to conflicting memory states, disrupting agent collaboration [**?**].
- **Obsolete Data Overload**: Retaining outdated memories increases latency and degrades decision-making quality [**?**].
- **Resource Constraints**: As agent counts scale, memory management must balance computational cost and efficiency.

The Co-Forgetting Protocol tackles these issues by synchronizing memory pruning across agents. Unlike traditional methods where agents manage memory independently, this protocol ensures collective agreement on memory retention, reducing redundancy and enhancing system performance. It draws inspiration from biological systems, where memory decay and selective forgetting optimize cognitive efficiency [**?**].

The protocol uses a centralized vector database (e.g., Pinecone [**?**]) to store memories as embeddings. After each interaction epoch, agents propose memories for removal based on decay scores, then vote using a quorum-based system. Fault tolerance is ensured through Practical Byzantine Fault Tolerance (PBFT) [**?**], accommodating unreliable agents. Key features include weighted voting, multi-scale decay, and meta-learned optimization, making it adaptable and scalable.

Contributions include:

- A fault-tolerant, quorum-based voting mechanism with role-weighted agent contributions.
- Multi-scale decay functions to pre-filter memories across varying time horizons.
- Meta-learning to dynamically tune thresholds for task-specific performance.
- Detailed implementation strategies for real-world deployment.

Section 2 surveys prior work, Section 3 explains the protocols design, Section 4 elaborates on implementation, Section 5 presents results, and Section 6 discusses future directions.

2 Related Work

Distributed systems research, such as HYDRAstor [**?**], focuses on concurrent data management but lacks dynamic adaptation for MAS. These systems prioritize static consistency over context-aware pruning, limiting their applicability to AI-driven agents.

Adaptive forgetting in LLMs [**?**] employs decay-based memory pruning to enhance flexibility. However, these methods are single-agent focused and do not address multi-agent synchronization. The Co-Forgetting Protocol extends this by integrating collective decision-making.

Consensus mechanisms like Raft [**?**] and PBFT [**?**] provide fault tolerance in distributed systems. PBFTs ability to handle Byzantine faults aligns with our need for reliability in untrusted MAS environments, unlike Rafts crash-fault tolerance.

MAS memory management often uses individual pruning or centralized control [**?**]. Individual pruning lacks coordination, risking data loss, while centralized control creates bottlenecks. Our protocol distributes decision-making while ensuring synchronization, bridging this gap.

3 Methodology

The Co-Forgetting Protocol operates through quorum-based voting, multi-scale decay, and meta-learned thresholds, synchronized across epochs. Below, we elaborate on each component.

3.1 Quorum-Based Voting

Agents vote on memory retention after each epoch, with a quorum (e.g., $\frac{2}{3}$ of agents) required to discard a memory. Votes are weighted by agent roles (e.g., $w_i = 1.5$ for strategic agents, $w_i = 1.0$ for sensory agents), reflecting their expertise. The forgetting score is:

$$S = \sum_{i:\text{vote}_i=\text{forget}} w_i$$

A memory is removed if $S \geq Q$, where $Q = \frac{2}{3}\sum w_i$.

PBFT ensures consensus despite f faulty agents in a $3f+1$ agent system, using a three-phase commit (pre-prepare, prepare, commit) [**?**]. This robustness is critical for untrusted environments.

3.2 Multi-Scale Decay Layer

Memories decay over time using n exponential functions:

$$D_i(t) = \exp\left(-\frac{t - t_{\text{last}}}{S_i}\right), \quad i = 1, \ldots, n$$

where t_{last} is the last access time, and S_i ranges from short-term (e.g., 10s) to long-term (e.g., 1h). The combined score is:

$$D(t) = \frac{1}{n}\sum_{i=1}^{n} D_i(t)$$

Memories with $D(t) < \delta$ (e.g., $\delta = 0.3$) are proposed for forgetting.

Decay scales S_i adapt via:

$$S_i \leftarrow S_i + \alpha \cdot (f_{\text{access}} + \beta \cdot r_{\text{task}})$$

where f_{access} is access frequency, r_{task} is task reward, and $\alpha = 0.01$, $\beta = 0.5$.

3.3 Meta-Learned Thresholds

MAML [**?**] optimizes Q, δ, and S_i to balance task reward R and memory cost C:

$$L = -R + \lambda \cdot C, \quad \lambda = 0.1$$

The inner loop trains on task-specific data, while the outer loop generalizes across tasks.

3.4 Epoch Synchronization

Epochs occur after $N = 100$ interactions: 1. **Decay Calculation**: Compute $D(t)$ for all memories. 2. **Proposal**: Agents propose memories with $D(t) < \delta$. 3. **Voting**: PBFT aggregates votes asynchronously. 4. **Update**: Commit changes to the database.

Algorithm 1 Enhanced Epoch Synchronization

Input: Memory database $\mathscr{M}$, agents $\mathscr{A}$, δ, Q
Output: Updated $\mathscr{M}'$
for each $a \in \mathscr{A}$ in parallel **do**
    $D_m(t) \leftarrow \frac{1}{n}\sum_{i=1}^{n} \exp\left(-\frac{t-t_m}{S_i}\right)$ for $m \in \mathscr{M}$
    $P_a \leftarrow \{m \mid D_m(t) < \delta\}$
end for
$P \leftarrow \bigcup_{a\in\mathscr{A}} P_a$
for each $m \in P$ **do**
    $V_m \leftarrow \text{PBFT}(\{\text{vote}_a(m) \mid a \in \mathscr{A}\})$
    $S_m \leftarrow \sum_{a:\text{vote}_a(m)=\text{forget}} w_a$
    if $S_m \geq Q$ **then**
        $\mathscr{M} \leftarrow \mathscr{M} \setminus \{m\}$
    end if
end for
return $\mathscr{M}'$

4 Implementation

4.1 Infrastructure Setup

We use Pinecone [**?**] for its scalability and fast similarity search. Memories are stored as: - **Embedding**: 768D vectors from BERT [**?**]. - **Metadata**: Timestamp, agent ID, salience score (0-1 based on initial task relevance). - **Indexing**: Cosine similarity for efficient retrieval.

PBFT is deployed via the `pbft` library, with each agent running a node. Nodes communicate over gRPC for low-latency consensus, supporting up to 33% faulty agents. Local LLMs (e.g., LLaMA [**?**]) integrate with Pinecone via a custom Memory Manager plugin.

The system runs on a cluster of 4 nodes (24GB RAM, 12-core CPUs), ensuring distributed resilience.

4.2 Meta-Learning Loop

Benchmarks include: - **AgentBench Collaborative Planning**: Agents plan a delivery route, needing memory of past routes and obstacles. - **MiniWoB++ Form Filling**: Agents fill web forms, requiring memory of user preferences.

The loop runs: 1. **Initialization**: Set $Q = 0.66\sum w_i$, $\delta = 0.3$, $S_i = [10s, 1m, 1h]$. 2. **Inner Loop**: 10 epochs per task, updating parameters with SGD ($\alpha = 0.01$). 3. **Outer Loop**: Aggregate gradients across tasks, updating meta-parameters every 5 tasks.

4.3 Integration and Synchronization

Each agent maintains a 100-entry LRU cache, refreshed at epoch start. Cache hits reduce database queries by 40%, measured via profiling.

Transactions use Pinecones atomic updates. A versioned Merkle tree tracks memory states, enabling agents to detect and resolve inconsistencies within 50ms.

Synchronization workflow: 1. **Pre-Epoch**: Agents sync caches with the database via REST API. 2. **Post-Voting**: Commit phase broadcasts changes, with agents polling for updates every 10s.

5 Evaluation

5.1 Metrics

Metrics include: - **Task Success Rate**: Completion accuracy. - **Memory Footprint**: Retained entries post-100 epochs. - **Latency**: Time per epoch (decay + voting). - **Coordination Efficiency**: Reduction in action conflicts.

5.2 Experimental Setup

We simulate 10-50 agents on a 4-node cluster, using Pinecones cloud API and local LLaMA instances.

5.3 Results

Metric	Baseline	Co-Forgetting	Improvement
Task Success Rate	75%	87%	+12%
Memory Footprint	12,000	5,500	-54%
Latency (ms)	10	18	+8ms
Coordination Efficiency	60%	85%	+25%

Table 1: Performance comparison over 100 epochs.

The protocol reduces memory usage by 54% and boosts coordination by 25%, though latency rises slightly due to PBFT overhead.

5.4 Ablation Study

- **No Decay Layer**: Memory footprint rises to 9,000 (+63%). - **Fixed Thresholds**: Success rate drops to 80% (-7%). - **Equal Weights**: Coordination falls to 70% (-15%).

6 Conclusion

The Co-Forgetting Protocol offers a scalable, fault-tolerant solution for MAS memory management, validated by significant performance gains.

Future directions: - **Scalability**: Test with 100+ agents using hierarchical voting. - **Real-World Testing**: Deploy in robotics for live coordination.