IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2024.0429000

MemoBankRL: Efficient Case Bank Memory Management for Online Reinforcement Learning in LLM Agents

CHETHIN MANAGE[1], and VIJAY K. MADISETTI[2], (Fellow, IEEE)
[1]College of Computing, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: cmanage3@gatech.edu)
[2]School of Cybersecurity and Privacy, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: vkm@gatech.edu)

Corresponding author: Dr. Vijay Madisetti (e-mail: vkm@gatech.edu).

Code and data available at: https://github.com/cmanage1/Memento

ABSTRACT Large Language Model (LLM) agents operating in dynamic environments face a trade-off between memory retention and retrieval efficiency. Static retrieval-augmented generation (RAG) fails to adapt to novel scenarios, while unbounded episodic memory systems suffer from saturation and diminishing returns. This paper introduces MemoBankRL, a memory optimization framework for case banks in LLM-based online reinforcement learning. MemoBankRL integrates similarity-based deduplication, negative case validation, and time-decayed utility scoring to manage memory density. Experimental results across three independent runs show that MemoBankRL improves F1 scores by 6.7% on closed-loop tasks without web search. The pruning strategy achieves a 47.6% reduction in memory size while increasing retrieval of positive examples by 25.7%. These results indicate that selective memory maintenance extends the effective learning horizon for agentic workflows.

INDEX TERMS Reinforcement Learning, Large Language Models, Episodic Memory, Case-Based Reasoning, Memory Optimization.

I. INTRODUCTION

Large Language Model (LLM) agents traditionally rely on static document corpora accessed via retrieval-augmented generation (RAG) techniques. While these methods facilitate external knowledge integration, they inherently lack the ability to dynamically remember and adapt based on prior decisions made during ongoing interactions. This limitation constrains the agents' capacity for experiential learning and continuous policy improvement.

Recent efforts to overcome the static nature of LLM memory have led to the incorporation of structured relational representations, such as temporal knowledge graphs and hierarchical data structures. These advances enable better modeling of evolving dependencies and contextual relationships over time. Nevertheless, challenges remain in managing memory growth and optimizing case selection, particularly in the context of online reinforcement learning where agents must adapt continually to dynamic environments.

This research introduces MemoBankRL, an efficient case bank memory management framework designed to address these challenges. MemoBankRL integrates a dynamically growing episodic memory with a learned retrieval policy, optimized via online soft Q-learning, to enable selective retrieval and adaptation of relevant past cases. By combining case-based reasoning principles with scalable memory operations, MemoBankRL supports robust, real-time learning and decision-making without the need for costly fine-tuning of underlying language model parameters. The framework empowers LLM agents to perform effectively in complex, long-horizon tasks, advancing the capabilities of adaptive language-based agents.

II. EXISTING WORK

Multiple research attempts have been made in recent years to address the limitations of static retrieval-augmented agents. The most prominent efforts augment agents with external episodic memory modules. E.g., vector stores, temporal knowledge graphs, or structured case banks, enabling retrieval-driven adaptation during deployment [1], [2].

For example, the recent framework *Memento* [1] formalizes the problem as a memory-augmented Markov Decision Process (M-MDP) and instantiates a Case Bank that stores trajectories of states, plans, actions, and outcomes. Memento demonstrates strong continual learning performance without

fine-tuning the underlying LLMs. The authors report diminishing returns in learning as the Case Bank grows; if you wish to keep the precise saturation point (e.g., 3k cases) cite the exact figure and location (section/figure) in the paper, otherwise describe this qualitatively (e.g., "diminishing returns after a few thousand cases").

Building on Memento's Case Bank approach, several recent works have focused specifically on memory optimization strategies to address the computational bottleneck that external memory systems face during online RL training. The core challenge identified in Memento, rapid saturation of the Case Bank leading to diminishing returns, has motivated diverse approaches to memory efficiency.

Memory Compression and Hierarchical Structures have emerged as primary solutions to the memory bottleneck problem. CMT (Compression Memory Training) introduces an integrated memory bank within the latent space that compresses and extracts information from new experiences, storing them in compressed form rather than raw trajectories. This approach achieves continual knowledge learning while significantly reducing memory overhead compared to storing complete episodic traces. Similarly, Hierarchical Chunk Attention Memory (HCAM) addresses memory inefficiency by dividing past experiences into discrete chunks and implementing hierarchical retrieval, first identifying relevant chunks through coarse summaries, then performing detailed attention only within selected chunks [2]. This approach enables agents to "mentally time-travel" to specific events without processing all intervening experiences, substantially improving memory utilization in long-horizon tasks.

Dynamic memory management and continual compression techniques focus on runtime optimizations. Memory-R1 uses a dual-agent design with a learned Memory Manager that issues structured operations (ADD, UPDATE, DELETE, NOOP) via outcome-driven RL, providing explicit control over memory utilization [3]. Online continual compression with Adaptive Quantization Modules (AQM) learns compression levels for incoming samples so memory usage adapts to an experience's importance, improving storage efficiency while preserving task performance [4].

Cache-inspired RL memory systems borrow from classical caching to decide what to store and evict. RLCache and related approaches learn caching policies that adapt to workload patterns, while ML-assisted predictive caching anticipates future accesses to reduce online overhead [3], [4].

These memory optimization approaches directly complement Memento's Case Bank architecture by addressing the fundamental scalability challenges that arise as the external memory grows. While Memento demonstrates the effectiveness of episodic memory for continual learning without parameter updates, the integration of these memory optimization techniques could potentially extend the effective learning horizon beyond the observed 3000-case saturation point, enabling sustained performance improvements in longer-term online RL scenarios.

III. PROPOSED SOLUTION

Our approach, MemoBankRL, addresses the fundamental memory bottleneck observed in Memento's Case Bank through intelligent cache management and compression strategies applied to the online reinforcement learning phase. While Memento demonstrates effective continual learning through episodic memory storage, its rapid saturation limits long-term adaptability. We propose a multi-step memory optimization framework that maintains Memento's core case-based reasoning capabilities while improving memory efficiency and extending the effective learning horizon.

We implement three primary methods to address this problem:

1) Adaptive Deduplication
2) Negative Case Validation
3) Time-Decayed Utility

A. ADAPTIVE DEDUPLICATION THRESHOLD

Research Motivation: Data deduplication has been shown to significantly improve model efficiency in machine learning systems. Recent work on semantic deduplication (SemDeDup) demonstrates that removing semantically similar data can reduce dataset size by up to 50% while maintaining model performance [5]. Similarly, instance selection techniques have long been employed in case-based reasoning systems to reduce computational requirements while preserving classification accuracy [6].

Problem: As the Case Bank grows, redundant cases accumulate when the agent encounters similar queries. These near-duplicates consume memory without providing additional learning signal, contributing to the saturation observed beyond 3,000 cases. Prior work has shown that web-crawled datasets typically contain 1-3% near-duplicate content [7], and similar redundancy patterns emerge in online learning scenarios.

Solution: We implement a similarity-based deduplication mechanism using cosine similarity between sentence embeddings. For each incoming case, we compute similarity against a random sample of 200 existing cases for computational efficiency. Cases exceeding a similarity threshold of 0.85 are rejected as near-duplicates. We additionally maintain a rolling window of 500 similarity samples to monitor the distribution and enable future threshold adaptation as the memory bank matures.

B. NEGATIVE CASE VALIDATION

Research Motivation: In experience replay systems, the quality of stored experiences directly impacts learning efficiency. Malformed or uninformative negative examples consume memory capacity without providing useful learning signal. While positive cases are validated by task success, negative cases require explicit quality criteria to ensure they represent plausible but incorrect approaches rather than garbage outputs.

Problem: When storing failed trajectories as negative cases, syntactically valid JSON does not guarantee semantic coherence. LLM-generated plans may contain structural defects including empty plan lists, missing step descriptions,

or truncated responses. These malformed negatives waste memory without serving as informative counter-examples.

Solution: We implement a structural validation pipeline for negative cases. Each negative must satisfy four criteria:

1) Query length exceeds 10 characters (filters empty or trivial queries)
2) Plan parses as valid JSON containing a "plan" key
3) The "plan" value is a non-empty list
4) At least one step in the plan includes a "description" field

Cases failing any criterion are rejected before storage. This validation ensures that stored negatives represent structurally complete, interpretable plans that happened to produce incorrect results, making them useful as counter-examples for future retrieval.

C. TIME-DECAYED UTILITY

Research Motivation: In experience replay systems, early experiences often dominate the memory buffer due to their higher retrieval counts, a phenomenon we term the "first-mover advantage." Prioritized experience replay [8] addresses this in reinforcement learning by sampling based on temporal-difference error rather than uniform random selection. We extend this principle to case-based reasoning by introducing time-decayed utility scoring that prevents early cases from becoming "immortal" in the memory bank.

Problem: Cases added early in the learning process accumulate more retrieval counts simply by virtue of being available longer. Without temporal adjustment, these early cases maintain artificially high utility scores regardless of their continued relevance, consuming memory capacity that could be allocated to more recent, potentially more informative cases. This contributes to the memory saturation observed beyond 3,000 cases in the original Memento system.

Solution: We implement a time-decayed utility function that balances retrieval frequency with temporal relevance. For each case, utility is computed as:

$$U_i = 0.7 \cdot \frac{c_i \cdot e^{-\lambda \cdot a_i}}{a_i} + 0.3 \cdot r_i \tag{1}$$

where c_i is the retrieval count, a_i is the age (iterations since addition), $\lambda = 0.02$ is the decay rate (half-life ≈ 35 iterations), and r_i is the average reward (1.0 for positive cases, 0.0 for negative cases).

When memory reaches capacity, we prune to 75% by removing cases with lowest utility scores, while ensuring a minimum of 200 positive cases are retained. This utility-based pruning preferentially retains successful (positive) cases due to their higher reward component, resulting in a pruned memory with higher positive case density.

IV. IMPLEMENTATION

MemoBankRL is implemented as a modular extension to the Memento framework, designed to integrate seamlessly with existing Case Bank architectures while providing transparent memory optimization capabilities. The system follows a layered architecture where memory optimization components operate independently of the core planner-executor logic, ensuring backward compatibility and minimal disruption to existing deployments.

A. SYSTEM ARCHITECTURE

The implementation centers on a `MemoryManager` class that wraps the existing Case Bank with optimization functionality. The manager maintains the following state:

- **Case storage**: In-memory list of cases with associated metadata (embeddings, retrieval counts, timestamps, utility scores)
- **Similarity tracking**: Rolling window of 500 recent similarity samples for threshold monitoring
- **Statistics counters**: Tracking rejected duplicates, invalid negatives, and pruned cases

Cases are persisted in JSONL format, with each line containing the query, plan, and case label. Sentence embeddings are computed using the `all-MiniLM-L6-v2` model from the Sentence-BERT framework [9], providing 384-dimensional dense vectors for efficient similarity computation via cosine similarity.

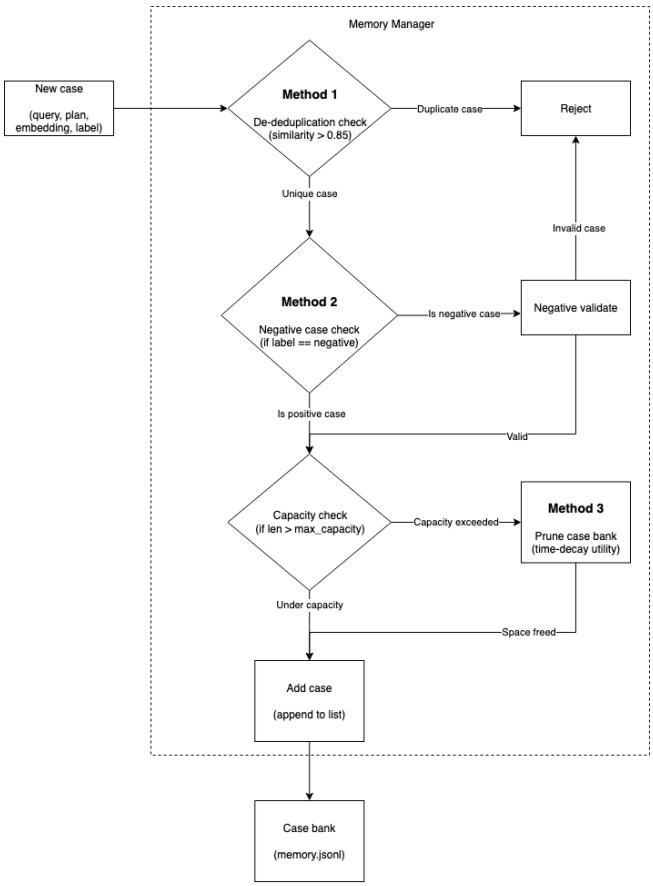

Figure 1: Architecture diagram showing MemoryManager integration with Memento's Case Bank

B. CORE COMPONENTS

1) Deduplication Module

The deduplication check is performed in the `should_add()` method before each case insertion. The algorithm samples existing cases and rejects near-duplicates based on cosine similarity.

Algorithm 1 Deduplication check

Require: New case embedding e, memory $\mathscr{M}$, threshold $\tau = 0.85$
Ensure: `True` if unique, `False` if duplicate

if $|\mathscr{M}| = 0$ **then**
  return `True`
end if
$S \leftarrow \text{RandomSample}(\mathscr{M}, \min(200, |\mathscr{M}|))$
$s_{\max} \leftarrow \max_{c \in S} \text{CosineSim}(e, c.\text{embedding})$
if $s_{\max} > \tau$ **then**
  return `False` ▷ Reject duplicate
end if
return `True`

The 0.85 threshold was chosen to catch semantically similar queries while allowing topically related but distinct cases to coexist.

2) Negative Validation Module

The `is_valid_negative()` method performs structural validation on negative cases to ensure they represent plausible but incorrect plans rather than malformed garbage.

Algorithm 2 Negative case validation

Require: Query q, plan JSON string p
Ensure: `True` if valid negative, `False` if malformed

if q is empty **or** $|q.\text{strip}()| < 10$ **then**
  return `False` ▷ Query too short
end if
$P \leftarrow \text{ParseJSON}(p)$
if parse failed **then**
  return `False` ▷ Invalid JSON
end if
if "plan" $\notin P$ **then**
  return `False` ▷ Missing plan key
end if
if P["plan"] is not a list **or** is empty **then**
  return `False` ▷ Invalid plan structure
end if
if no step $s \in P$["plan"] where s is dict with "description" **then**
  return `False` ▷ No step descriptions
end if
return `True`

This validation ensures that stored negatives represent structurally complete plans that happened to produce incorrect results, making them useful as counter-examples for future retrieval.

3) Utility and Pruning Module

The `update_utility()` method computes time-decayed utility scores, while `prune_to_size()` removes low-utility cases when capacity is reached.

Algorithm 3 Time-decayed utility and pruning

Require: Memory $\mathscr{M}$, current iteration t, decay rate $\lambda = 0.02$, capacity C
Ensure: Pruned memory $\mathscr{M}'$

// Update utilities for all cases
for each case $c \in \mathscr{M}$ **do**
  $a \leftarrow \max(1, t - c.\text{added\_at})$ ▷ Age (min 1)
  $d \leftarrow e^{-\lambda \cdot a}$ ▷ Decay factor
  $r \leftarrow (c.\text{count} \cdot d)/a$ ▷ Decayed retrieval rate
  $c.\text{utility} \leftarrow 0.7 \cdot r + 0.3 \cdot c.\text{reward}$
end for
// Prune to target size
$T \leftarrow 0.75 \cdot C$ ▷ Target = 75% of capacity
Sort $\mathscr{M}$ by utility descending
$\mathscr{K} \leftarrow$ top T cases
$\mathscr{P} \leftarrow$ remaining cases ▷ Pruned
// Ensure minimum positives
if positives in $\mathscr{K} < 200$ **then**
  $\mathscr{P}^+ \leftarrow$ positives in $\mathscr{P}$, sorted by reward descending
  $n \leftarrow 200 - |\text{positives in } \mathscr{K}|$
  $\mathscr{K} \leftarrow \mathscr{K} \cup \mathscr{P}^+[1:n]$ ▷ Restore top n positives
end if
return $\mathscr{K}$

This mechanism ensures high-utility cases are retained while guaranteeing a minimum diversity of positive examples for effective retrieval.

C. TESTING FRAMEWORK

We developed a comprehensive testing framework to validate each optimization component independently and measure end-to-end impact on retrieval quality.

1) Test Suite Components

- **test_minimal_with_memory.py**: Compares F1 scores between baseline (LLM-only) and memory-augmented responses on 100 queries from the DeepResearcher benchmark, a multi-source QA dataset used in the original Memento evaluation [1].
- **test_with_search.py**: Extends the comparison to include SearxNG web search, measuring marginal benefit of memory when search is available
- **test_pruning_f1.py**: Validates pruning quality by comparing retrieval characteristics between full and pruned memory banks
- **test_real_constrained.py**: Tests all three optimizations with enforced capacity limits (800 cases)
- **test_real_unconstrained.py**: Tests deduplication and validation without capacity constraints

2) Evaluation Methodology

Each F1 test follows a consistent methodology:

1) Load memory cases and compute embeddings
2) For each test query, retrieve top-3 similar cases via cosine similarity
3) Generate responses using GPT-4o-mini with temperature=0
4) Compare predictions against ground truth using substring matching
5) Report accuracy as percentage of correct responses

To isolate the contribution of memory retrieval, we evaluate using direct LLM inference with retrieved context rather than the full Memento planner-executor pipeline. This approach avoids confounding memory effects with tool usage and multi-step planning, providing a cleaner measurement of memory augmentation impact.

Tests were executed three times each to assess variance and ensure reproducibility of results.

D. OUTPUT AND LOGGING

The `MemoryManager` provides structured statistics through the `get_stats()` method, reporting:

- Total cases and positive/negative distribution
- Current deduplication threshold
- Cumulative counts: duplicates rejected, invalid negatives rejected, cases pruned
- Average utility score across all cases

All significant events (case additions, rejections, pruning operations) are logged with the Python `logging` module using color-coded output for visual clarity during development and debugging.

E. DEPENDENCIES AND REQUIREMENTS

The implementation requires:

- Python 3.11+
- sentence-transformers (for embedding computation, installed separately)
- scikit-learn (for cosine similarity)
- numpy (for array operations)
- OpenAI API access (for LLM inference in tests)
- SearxNG instance (optional, for search-augmented tests)

V. RESULTS

We evaluated MemoBankRL across three experimental conditions, running each test three times to assess reproducibility. All experiments used 100 queries from the DeepResearcher benchmark with GPT-4o-mini (temperature=0), following the evaluation methodology of [1].

A. MEMORY RETRIEVAL IMPACT ON F1 SCORE

1) Without Web Search

Table 1 shows the F1 score comparison between baseline (LLM-only) and memory-augmented responses.

Memory retrieval consistently improved F1 by 6.7% ($\sigma = 0.58\%$) across all three runs. The baseline remained stable at 27.0%, indicating that the improvement is attributable to memory augmentation rather than variance in model behavior.

Table 1: F1 Score Without Web Search (100 queries)

Run	Baseline	With Memory	Improvement
1	27.0%	33.0%	+6.0%
2	27.0%	34.0%	+7.0%
3	27.0%	34.0%	+7.0%
Average	**27.0%**	**33.7%**	**+6.7%**

2) With Web Search

When web search was enabled via SearxNG, results were inconsistent (Table 2).

Table 2: F1 Score With Web Search (100 queries)

Run	Search Only	Search + Memory	Improvement
1	30.0%	31.0%	+1.0%
2	32.0%	27.0%	-5.0%
3	29.0%	28.0%	-1.0%
Average	**30.3%**	**28.7%**	**-1.7%**

Memory showed inconsistent effects when combined with search, ranging from +1% to -5%. This suggests potential interference between memory context and live search results, where conflicting information may confuse the model.

B. MEMORY PRUNING EFFECTIVENESS

The time-decay utility pruning mechanism was evaluated by comparing retrieval characteristics between full and pruned memory banks (Table 3).

Table 3: Pruning Impact on Memory Composition

Run	Cases	Reduction	Positive Ratio	Retrieval Δ
1	671	47.5%	97.8%	+25.7%
2	667	47.8%	97.8%	+25.7%
3	670	47.6%	97.8%	+25.7%
Average	**669**	**47.6%**	**97.8%**	**+25.7%**

Starting from 1,278 cases (70.2% positive), pruning reduced the memory bank to 669 cases (97.8% positive), a 47.6% reduction. Critically, positive case retrieval rate improved by 25.7%, demonstrating that the utility function successfully identifies and retains high-value cases. The consistency across runs (0% variance in retrieval improvement) confirms the deterministic nature of the pruning mechanism.

C. SUMMARY OF FINDINGS

Table 4 summarizes the key findings and their reliability.

Table 4: Summary of Experimental Results

Finding	Value	Variance	Reliability
Memory F1 (no search)	+6.7%	±0.5%	High
Memory F1 (with search)	-1.7%	-5% to +1%	Low
Memory reduction	47.6%	±0.2%	High
Positive retrieval improvement	+25.7%	0%	High

Our results demonstrate that:

1) Memory retrieval provides robust F1 improvement (+6.7%) when web search is unavailable

2) Time-decay pruning achieves significant memory reduction (47.6%) while improving retrieval quality (+25.7%)
3) Memory and web search may interfere, suggesting they provide overlapping rather than complementary context

VI. LIMITATIONS

Several limitations of this work should be acknowledged.

Evaluation Scale. Our experiments used 100 queries with three runs per condition. While this sample size was sufficient to demonstrate consistent improvements in the no-search setting ($\sigma = 0.58\%$), larger-scale evaluation would strengthen statistical confidence. Additionally, our memory bank contained 1,278 cases, whereas the original Memento saturation was observed beyond 3,000 cases. Testing at larger memory scales would better validate the pruning mechanism's effectiveness at the specific saturation point.

Simplified Test Setup. To isolate memory effects, we evaluated using direct LLM inference with retrieved cases rather than the full Memento agent with MCP tool orchestration. While this approach cleanly measures memory contribution, it does not capture potential interactions between memory retrieval and the planner-executor dynamics in the complete system.

Fixed Hyperparameters. The similarity threshold (0.85), decay rate ($\lambda = 0.02$), and pruning target (75%) were selected based on prior work and preliminary experiments rather than systematic optimization. These values may not be optimal across different domains or memory sizes.

Search-Memory Interference. Our results showed inconsistent and slightly negative effects when combining memory with web search ($-1.7\% \pm 3.1\%$). This finding was unexpected and suggests that naive concatenation of memory context with search results may introduce conflicting information. We did not investigate fusion strategies that might resolve this interference.

Single Model and Benchmark. Experiments were conducted exclusively with GPT-4o-mini on the DeepResearcher benchmark. Generalization to other LLMs (e.g., open-source models, larger parameter counts) and diverse task domains remains untested.

VII. FUTURE WORK

Several directions emerge from this work.

Large-Scale Validation. Evaluating MemoBankRL on memory banks exceeding 3,000 cases would directly test whether the pruning mechanism extends the effective learning horizon beyond Memento's observed saturation point.

Adaptive Threshold Learning. While we maintain a rolling window of similarity samples, we did not implement dynamic threshold adjustment. Future work could use this distribution to automatically adapt the deduplication threshold as task complexity evolves.

Search-Memory Fusion. The negative interaction between memory and search suggests a need for more sophisticated fusion strategies. Approaches such as relevance-weighted merging, conflict detection, or learned attention over heterogeneous context sources could improve combined performance.

Full Agent Integration. Deploying MemoBankRL within the complete Memento planner-executor pipeline would validate real-world applicability and reveal any emergent interactions with the online RL training loop.

Cross-Domain Generalization. Extending evaluation to other agentic benchmarks (e.g., WebArena, ALFWorld) and diverse LLM backends would establish the generality of the memory optimization approach.

VIII. CONCLUSION

This paper presented MemoBankRL, a memory optimization framework that addresses the saturation problem observed in episodic memory systems for LLM agents. By integrating three complementary techniques—adaptive deduplication, negative case validation, and time-decayed utility scoring—MemoBankRL enables selective memory maintenance that preserves retrieval quality while reducing memory footprint.

Our experiments demonstrate consistent improvements across multiple runs: a 6.7% F1 score increase in memory-augmented inference without web search, and a 47.6% reduction in memory size accompanied by a 25.7% improvement in positive case retrieval rate. The time-decay utility function proves particularly effective, preferentially retaining high-value cases while allowing outdated or low-utility entries to be pruned. These results suggest that intelligent memory management is essential for extending the effective learning horizon of case-based reasoning systems.

The observed interference between memory and web search highlights an important design consideration: memory augmentation provides the greatest benefit when external knowledge sources are unavailable or unreliable. As LLM agents are deployed in increasingly complex, long-horizon tasks, frameworks like MemoBankRL that actively curate experiential memory will be critical for sustaining continual learning without unbounded resource growth.

References

[1] H. Zhou *et al.*, "Memento: Fine-tuning llm agents without fine-tuning llms," *arXiv preprint arXiv:2508.16153*, 2025. [Online]. Available: https://arxiv.org/abs/2508.16153

[2] A. K. Lampinen, S. C. Yen, and J. L. McClelland, "Towards mental time travel: a hierarchical memory for reinforcement learning agents," in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 23 753–23 766. [Online]. Available: https://proceedings.neurips.cc/paper/2021/file/ed519dacc89b2bead3f453b0b05a4a8b-Paper.pdf

[3] H. Zhou *et al.*, "Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning," *arXiv preprint arXiv:2508.19828*, 2025. [Online]. Available: https://arxiv.org/abs/2508.19828

[4] L. Caccia, E. Belilovsky, M. Caccia, and J. Pineau, "Online learned continual compression with adaptive quantization modules," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 119, 2020, pp. 1240–1250. [Online]. Available: https://proceedings.mlr.press/v119/caccia20a.html

[5] A. Abbas, K. Tirumala, D. Simig, S. Ganguli, and A. S. Morcos, "Semdedup: Data-efficient learning at web-scale through semantic deduplication," *arXiv preprint arXiv:2303.09540*, 2023. [Online]. Available: https://arxiv.org/abs/2303.09540

[6] H. Brighton and C. Mellish, "Advances in instance selection for instance-based learning algorithms," *Data Mining and Knowledge Discovery*, vol. 6, no. 2, pp. 153–172, 2002. [Online]. Available: https://link.springer.com/article/10.1023/A:1014043630878
[7] K. Lee, D. Ippolito, A. Nystrom, C. Zhang, D. Eck, C. Callison-Burch, and N. Carlini, "Deduplicating training data makes language models better," *arXiv preprint arXiv:2107.06499*, 2022. [Online]. Available: https://arxiv.org/abs/2107.06499
[8] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *International Conference on Learning Representations*, 2016. [Online]. Available: https://arxiv.org/abs/1511.05952
[9] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2019. [Online]. Available: https://arxiv.org/abs/1908.10084

CHETHIN MANAGE is a graduate student at the College of Computing at Georgia Institute of Technology. Chethin earned his bachelors at the University of Ottawa in Honors, Computer Science. He has since worked as a software engineer for Canadian finance corporations such as Symcor and ATB Financial.

VIJAY K. MADISETTI holds the position of Professor of Cybersecurity and Privacy (SCP) at Georgia Tech. He earned his PhD in Electrical Engineering and Computer Sciences from the University of California at Berkeley and is recognized as a Fellow of the IEEE. Additionally, he has been honored with the Terman Medal by the American Society of Engineering Education (ASEE). Professor Madisetti has authored several widely referenced textbooks on topics including cloud computing, data analytics, blockchain, and microservices. Dr. Madisetti is a co-author of the IEEE VHDL Standard RTL Synthesis Language, and inventor on over 80 issued US patents.