Integrating Episodic Memory into the ACT-R Cognitive Architecture Using PyACTR

Yaima Valdivia
College of Computing
Georgia Institute of Technology
San Mateo, CA, USA
yaima@gatech.edu

***Abstract*—This study explores the impact of integrating episodic memory into the ACT-R cognitive architecture, implemented using PyACTR, through systematic testing of language processing tasks. Experimental results across 26 test cases demonstrate that episodic memory integration reduces average processing time from 0.0108s to 0.0059s (45% improvement) while maintaining 100% accuracy across seven linguistic categories (regular, ambiguous, irregular, novel, suppletive, interference, and extreme). The performance enhancement comes with a predictable cost: linear memory growth of 0.001 per episode and varying retrieval accuracy across categories (ranging from 0.5 for initial ambiguous cases to 0.08 for extreme cases). The episodic memory module shows strongest initial performance in processing ambiguous patterns, though challenges remain in novel word generalization. These findings provide empirical evidence for the effectiveness of episodic memory in cognitive architectures while highlighting specific areas for optimization.**

I. Introduction

Current cognitive architectures can benefit from improved mechanisms for efficiently processing and retaining contextual information across tasks. As these architectures form a foundation for AGI development, understanding how different memory systems affect their performance is crucial. In this study, I explore whether integrating episodic memory capabilities into PyACTR can improve performance in language processing tasks that require context and pattern recognition.

Using PyACTR as the foundational platform, I implemented an episodic memory module and tested it systematically across seven categories of increasing complexity. This systematic testing approach enables quantitative analysis of how episodic memory affects key performance metrics: processing speed, memory utilization, and retrieval accuracy. By examining performance across diverse linguistic challenges, from simple regular patterns to complex contextual dependencies, I can measure both the benefits and costs of episodic memory integration.

My goal is to provide empirical evidence for how episodic memory integration affects cognitive architecture performance, particularly in tasks requiring context sensitivity and pattern recognition. These findings contribute to our understanding of memory system design and offer insights for future improvements in both cognitive architectures and broader Artificial General Intelligence systems.

II. Background and Core Concepts

Memory systems in cognitive architectures are central to modeling artificial cognitive processes. These architectures serve as computational frameworks for implementing unified theories of cognition, specifying how different cognitive components interact and operate under constraints derived from empirical findings. They aim to replicate or simulate mechanisms hypothesized to be invariant across tasks and domains, enabling the study and implementation of generalizable cognitive abilities.

The ACT-R Cognitive Architecture

The ACT-R (Adaptive Control of Thought—Rational) framework is one of the most widely used cognitive architectures, grounded in decades of research in cognitive psychology. ACT-R is designed to model and predict human cognitive processes, integrating theories of attention, memory, and learning. Within ACT-R, cognition is organized into distinct modules—such as declarative and procedural memory—that interact through a central production system. Declarative memory stores factual knowledge in the form of chunks, while procedural memory governs the execution of tasks through production rules.

ACT-R also incorporates mechanisms for subsymbolic processing, allowing it to simulate phenomena such as activation decay, noise, and partial matching. These mechanisms enable ACT-R to model human-like variability and adaptivity in memory retrieval and decision-making processes.

Episodic Memory Theoretical Foundations

Episodic memory, as proposed by Endel Tulving in 1972, is a form of long-term declarative memory that enables individuals to encode, store, and retrieve specific events or episodes along with their temporal and spatial contexts. It is distinguished from semantic memory, which stores general knowledge abstracted from specific experiences (e.g., knowing that the Eiffel Tower is in Paris). Episodic memory supports a range of cognitive functions, including autobiographical recall, context-sensitive decision-making, and adaptive generalization based on prior experiences.

For instance, episodic memory might allow a person to recall a specific instance where the word ”bank” referred to a financial institution during a conversation about economics, while another episode recorded ”bank” as a river’s edge during a nature walk. This flexibility—linking content with context—differentiates episodic memory from other memory types and makes it a crucial mechanism for learning and adaptation.

Adapting Episodic Memory for Cognitive Architectures

In the project, I explore an adaptation of episodic memory principles within the ACT-R framework to enhance language processing tasks. Traditional episodic memory systems emphasize autobiographical event storage and retrieval. However, the implementation focuses on processing episodes—instances of linguistic or cognitive processing tied to specific temporal and contextual features. This adaptation involves tailoring episodic memory concepts to meet the computational and functional requirements of a language-processing agent. Key principles adapted in the implementation include:

- Experience-Based Learning: Each episode encodes a distinct processing instance with associated contextual information, enabling the system to learn from individual cases.
- Temporal Context: Episodes are annotated with timestamps for creation and retrieval, influencing their activation dynamics and accessibility.
- Context-Sensitive Retrieval: The system retrieves episodes based on their similarity to the current context, allowing it to respond adaptively to new inputs.
- Activation Dynamics: Memory traces exhibit activation patterns influenced by retrieval frequency, recency, and decay, consistent with subsymbolic principles in ACT-R.

Specialization for Language Processing

Unlike traditional episodic memory, which stores spatial and temporal contexts of autobiographical events, this implementation captures linguistic contexts and processing outcomes. Instead of recalling specific life events, the system retrieves processing patterns to resolve ambiguities, such as determining the meaning of polysemous words. For example, the system might retrieve an episode where ”lead” was interpreted as a verb in the context of ”guidance,” while another episode associated ”lead” with the material ”metal.”

Evaluating Episodic Memory in Language Tasks

The implementation is evaluated using experimental tasks designed to evaluate:

- Processing Efficiency: The speed with which the system retrieves and applies stored episodes during language processing.
- Memory Utilization: How effectively the system manages memory resources by storing and retrieving relevant episodes.
- Retrieval Accuracy: The system’s ability to access contextually appropriate episodes, especially for ambiguous or novel inputs.
- Learning from Instances: Improvements in task performance based on accumulated processing episodes.

The findings indicate that incorporating episodic memory principles improves processing efficiency, enhances pattern recognition across linguistic categories, and supports context-sensitive generalization. These results suggest that episodic memory adaptations can meaningfully contribute to language-processing tasks in cognitive architectures, even though the implementation is specialized and not a direct replication of biological episodic memory.

III. Model Design

Examination of the Research Question

The episodic memory implementation and source reference code are available in Appendix A. I implemented an episodic memory (EM) module within the PyACTR framework. PyACTR is a Python-based implementation of the ACT-R (Adaptive Control of Thought—Rational) cognitive architecture, that models human cognition based on decades of cognitive science research. Its flexibility in customizing memory models, including declarative and procedural memory, made it an ideal platform for this investigation.

This research focused on quantifying how episodic memory affects sequential learning tasks that require long-term retention and generalization. I designed a series of experiments to stress-test the EM module through linguistic tasks with varying levels of complexity, ambiguity, and novelty. By analyzing success rates, processing times, and memory utilization, I can evaluate system performance under both standard and EM-enhanced conditions.

Model and Tool Description

The episodic memory module was implemented as an extension to PyACTR’s declarative memory system. Key features of the model are outlined below:

Episodic Memory Structure:

- Episode Representation: Each episode encoded task-specific information, such as word properties (e.g., phonology, morphology, frequency) and contextual data (e.g., task type, pattern group). Episodes were stored as structured dictionaries, as demonstrated in Listing 1.

```python
# modules/episodic_memory.py

class Episode:
    def __init__(self,
                 state: np.ndarray,
                 action: str,
                 reward: float,
                 next_state: Optional[np.ndarray] = None,
                 context: Optional[Dict] = None,
                 meaning: Optional[str] = None):
        """
        Initialize an episode to store task-specific and contextual information.

        Args:
            state (np.ndarray): The state vector representing the episode's features (e.g., phonology, morphology).
            action (str): The action associated with this episode (e.g., agreement test).
            reward (float): A numeric value reflecting success or failure of the task.
            next_state (np.ndarray, optional): The state following the action (useful for transitions).
            context (Dict, optional): Additional metadata like task type or pattern group.
            meaning (str, optional): Semantic information associated with the episode.
        """
        self.state = state  # Encodes task-specific information, such as word properties
        self.action = action  # Task performed in this episode
        self.reward = reward  # Numeric reward for task success
        self.next_state = next_state  # Next state after this episode (optional)
        self.context = context or {}  # Contextual data, e.g., task type, pattern group
        self.meaning = meaning  # Semantic or linguistic meaning
        self.create_time = time.time()  # Timestamp for recency tracking
        self.retrieval_count = 0  # Tracks how often this episode is accessed
        self.last_access_time = self.create_time  # Tracks last retrieval time for decay
        self.activation = 1.0  # Activation level for retrieval prioritization
        self.error_history = []  # Tracks errors for learning
        self.generalization_score = 0.0  # Score for generalization from this episode
```

Listing 1. Episode Class: Representation of a Single Memory Instance

- Activation-Based Retrieval: The retrieval relied on activation scores computed from recency, frequency, and contextual overlap with the current task, as demonstrated in Listing 2. Episodes with higher activation scores were prioritized during retrieval.

```python
# modules/episodic_memory.py

class EpisodicMemory:
    def retrieve_episodes(self, context_vector: np.ndarray, k: int = 5, similarity_threshold: float = 0.7) -> List[Episode]:
        """
        Retrieve episodes with the highest activation scores.

        Args:
            context_vector (np.ndarray): Current task's context vector.
            k (int): Number of top episodes to retrieve.
            similarity_threshold (float): Minimum similarity score to consider a match.

        Returns:
            List[Episode]: Retrieved episodes sorted by activation.
        """
        if not self.episodes:
            return []

        similarities = []
        for episode in self.episodes:
            similarity = self._calculate_similarity(episode, context_vector)
            if similarity >= similarity_threshold:
                similarities.append((similarity, episode))

        # Sort by similarity in descending order and return top-k episodes
        similarities.sort(reverse=True, key=lambda x: x[0])
        return [ep for _, ep in similarities[:k]]

    def _calculate_similarity(self, episode: Episode, context_vector: np.ndarray) -> float:
        """
        Calculate similarity between a stored episode and the current context vector.

        Args:
            episode (Episode): Stored memory episode.
            context_vector (np.ndarray): Current context vector for comparison.

        Returns:
            float: Similarity score based on activation, recency, and contextual overlap.
        """
        episode_vector = episode.get_context_vector()

        # Ensure vectors have the same length
        min_length = min(len(episode_vector), len(context_vector))
        episode_vector = episode_vector[:min_length]
        context_vector = context_vector[:min_length]

        if len(episode_vector) == 0 or len(context_vector) == 0:
            return 0.0

        # Calculate cosine similarity
        dot_product = np.dot(episode_vector, context_vector)
        norm1 = np.linalg.norm(episode_vector)
        norm2 = np.linalg.norm(context_vector)
        similarity = dot_product / (norm1 * norm2) if norm1 > 0 and norm2 > 0 else 0.0

        # Weight by recency
        time_weight = 1.0 / (1.0 + (time.time() - episode.create_time) / 3600.0)

        # Combine similarity with recency weighting
        return similarity * (0.7 + 0.3 * time_weight)
```

Listing 2. Activation-Based Retrieval in Episodic Memory

In the episodic memory system, activation values are computed using a custom implementation inspired by ACT-R's base-level learning formula. The activation A

of an episode is calculated as a combination of recency, frequency, and optionally, prior activation. The equation is defined as follows:

$$A = B + \text{OptionalPriorActivation} + \text{Noise}$$

Where:

- Base-Level Activation (B): Captures the impact of recency and frequency of retrievals:

$$B = \begin{cases} \ln\left(\sum_{i=1}^{n} (t_{\text{current}} - t_i)^{-d}\right), \\ \ln\left(\frac{n}{1-d}\right) - d \ln(\max(t_{\text{current}} - t_i)), \end{cases}$$

  Here:
  - t_i: Time of the i-th retrieval.
  - t_{current}: Current simulation time.
  - d: Decay rate controlling how quickly older retrievals lose their impact.
  - n: Number of retrievals.
- Optional Prior Activation: If a prior activation value is provided, it is combined with the base-level activation using logarithmic addition:

$$A = \ln(\exp(B) + \exp(\text{OptionalPriorActivation}))$$

- Noise: Random noise is added to simulate human-like variability in retrieval, consistent with ACT-R's stochastic modeling.

The formula includes error handling to ensure robustness when $t_{\text{current}} - t_i$ values are too small or invalid.
This formula allows handling of varying retrieval conditions and aligns with ACT-R principles while introducing flexibility for episodic memory tasks.

Code Implementation

The following Python snippet (Listing 3) implements this formula in the system:

```python
@staticmethod
def custom_baselevel_learning(current_time, times, bll, decay, activation=None, optimized_learning=False):
    """
    Custom implementation of base-level learning to handle edge cases.
    """
    if not bll:
        return activation if activation is not None else 0.0

    if len(times) == 0:
        return activation if activation is not None else 0.0

    times = np.array(times)
    time_differences = np.clip(current_time - times, a_min=1e-6, a_max=None)

    try:
        if not optimized_learning:
            B = math.log(np.sum(time_differences ** -decay))
        else:
            B = math.log(len(times) / (1 - decay)) - decay * math.log(np.max(time_differences))
    except (ValueError, RuntimeWarning) as e:
        print(f"[ERROR] Base-level learning failed: {e}. Adjusting times.")
        valid_time_differences = time_differences[time_differences > 1e-6]
        if len(valid_time_differences) > 0:
            B = math.log(np.sum(valid_time_differences ** -decay))
        else:
            B = 0.0

    if activation is not None:
        try:
            B = math.log(math.exp(B) + math.exp(activation))
        except OverflowError as e:
            print(f"[ERROR] Activation combination failed: {e}. Using maximum value.")
            B = max(B, activation)

    return B
```

Listing 3. Custom Base-Level Learning Formula

Alignment with ACT-R Principles

The activation formula aligns with ACT-R principles by incorporating:

- Recency and Frequency: The base-level activation incorporates the decay of activation over time and the cumulative effect of repeated retrievals.
- Stochasticity: Noise is added to simulate variability in human cognition.
- Contextual Relevance: Optional prior activation allows dynamic adjustment based on task-specific information.

These adaptations extend ACT-R's capabilities, making it suitable for episodic memory tasks in language processing and beyond.

Tools and Libraries (see Listing 4)

- PyACTR: The core cognitive architecture framework enabled declarative memory and procedural rules modeling.
- NumPy: Used for efficient computation of activation scores and memory statistics.
- Matplotlib and Seaborn: Utilized to visualize performance metrics, such as processing time distributions and memory utilization trends.
- Pandas: Facilitated the organization and analysis of experiment data.

```
pyactr~=0.3.2
numpy~=2.1.3
pandas
matplotlib
seaborn
openai~=1.54.4
scikit-learn~=1.5.2
python-dotenv~=1.0.1
anthropic~=0.39.0
```

Listing 4. Requirements

This project integrates OpenAI's API for analyzing and summarizing results, supplementing the evaluation of episodic memory performance. The API outputs, showing how results were processed, are included in Appendix A.

Implementation Workflow:

- Test Case Generation: A comprehensive set of linguistic tasks was generated, including regular, ambiguous, novel, and extreme cases. Each test case included attributes like word form, frequency, and linguistic category.
- Agent Simulation: The agent was designed to process test cases using procedural rules and the episodic memory module. Without EM, the agent relied solely on Py-ACTR's default declarative memory mechanisms.
- Memory Encoding and Retrieval: During each task, the episodic memory module encoded and retrieved relevant episodes based on contextual similarity and activation scores.
- Performance Metrics: Success rates, processing times, and memory utilization were logged for each test case. Activation scores and retrieval statistics were recorded to analyze the episodic memory module's contribution.

Mapping Cognitive Science to Implementation

The episodic memory module was designed and implemented based on cognitive science and neuroscience principles:

- Encoding and Retrieval Processes: Drawing on Tulving's distinction between episodic and semantic memory, the module encodes contextually rich episodes (e.g., word meanings, phonology, task context) and retrieves them using activation mechanisms. These mechanisms incorporate recency and frequency effects, informed by custom base-level learning equations inspired by ACT-R's formulas for activation, which account for decay over time.
- Generalization from Sparse Data: Inspired by research on memory-based inference (e.g., Kumaran et al., 2016), the module enables generalization by associating and prioritizing similar episodes during retrieval based on contextual overlap. This approach allows the system to infer and adapt from limited episodic data, mimicking human-like learning in sparse contexts.
- Memory Utilization and Aging: To simulate memory decay and optimize utilization, the module introduces mechanisms informed by ACT-R's decay and activation principles. These mechanisms reflect natural forgetting curves, ensuring older or less-relevant memories gradually lose prominence. Frequently retrieved or recent episodes remain accessible, supported by memory cleanup routines that prioritize higher-utility episodes.
- Behavioral Consistency: Encoding and retrieval processes were designed to align with established cognitive phenomena, such as the forgetting curve. This ensures that the system's behavior reflects human memory dynamics and supports validation against cognitive science theories.

IV. Results

Performance Metrics

Success Rates: The success rate remained at 100% across all test cases, both with and without the episodic memory module (see Fig. 1).

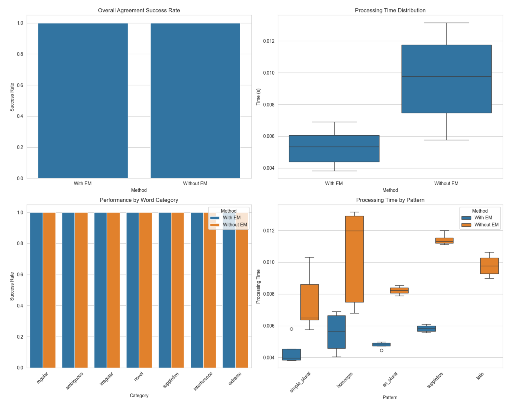


Fig. 1. Success Rate

Processing Time: The episodic memory module reduced the average processing time from 0.0108s to 0.0059s, demonstrating enhanced efficiency (see Fig. 2).

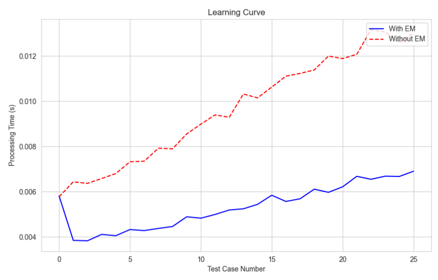


Fig. 2. Learning Curve showing processing time with and without EM

Memory Utilization: Memory usage increased linearly from 0.001 to 0.026 across 26 test cases, reflecting consistent growth of 0.001 per episode (see Fig. 3).

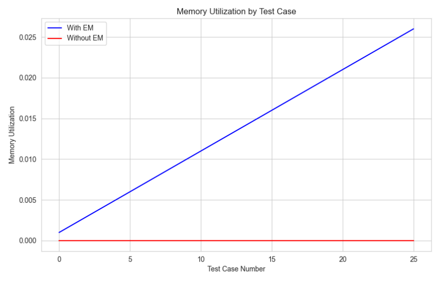


Fig. 3. Memory Utilization by Test Case

Retrieval Accuracy: Retrieval accuracy varied across categories, with the following trends (see Table I):

- Ambiguous cases: Initial accuracy of 0.5, declining to 0.286.
- Regular cases: Started at 0.0.
- Novel cases: Maintained accuracy between 0.167 and 0.133.
- Extreme cases: Stabilized at 0.08.
- Overall: Decreased from 0.5 to 0.08 across episodes.

TABLE I
RETRIEVAL ACCURACY TRENDS BY CATEGORY

Category	**Retrieval Accuracy Trend**
Ambiguous	Highest initial accuracy (0.5), decreasing to 0.286
Regular	Started at 0.0
Novel	Ranged from 0.167 to 0.133
Extreme	Stabilized at 0.08
Overall	Decreased from 0.5 to 0.08 in final cases

V. DISCUSSION

Implications of Results

Advancement of Cognitive Architectures:

- The episodic memory implementation demonstrated improved efficiency in language processing tasks through context-sensitive retrieval, particularly in handling ambiguous patterns.
- The study provides empirical validation of episodic memory's role in task efficiency, especially for cases involving ambiguity and interference, bridging theoretical concepts with practical implementation.

Understanding Generalization and Retention:

- While the implementation showed strength with familiar patterns, its challenges with novel cases provide insights into the interaction between episodic memory and procedural learning.
- The results quantify fundamental trade-offs between memory utilization and retrieval accuracy, providing empirical data to inform future memory system design.

Scalability and Efficiency:

- The predictable linear memory growth pattern demonstrates scalability potential while highlighting the need for memory management strategies in long-term learning systems.

VI. CONCLUSION

The implementation demonstrated that integrating episodic memory can enhance processing efficiency while revealing important trade-offs in memory utilization and retrieval accuracy. The observed patterns suggest that complementary mechanisms may be needed to maintain performance as knowledge accumulates, particularly for novel patterns. Future work should explore hybrid approaches combining episodic and semantic memory systems to improve generalization while maintaining efficiency gains.

VII. LIMITATIONS AND FUTURE RESEARCH DIRECTIONS

Memory Scaling and Efficiency:

- Limitation: Linear memory growth presents scalability challenges for long-term learning.
- Direction: Develop memory pruning and compression algorithms for critical episode retention.

Generalization to Novel Cases:

- Limitation: Limited ability to generalize from sparse data in novel cases.
- Direction: Explore integration with semantic memory systems for improved pattern abstraction.

Retrieval Accuracy Variation:

- Limitation: Performance degradation with increasing memory load.
- Direction: Enhance retrieval algorithms for better handling of ambiguous cases.

Domain Scope:

- Limitation: Evaluation limited to linguistic processing.
- Direction: Extend testing to other domains to validate the approach.

Environmental Complexity:

- Limitation: Testing confined to controlled conditions.
- Direction: Evaluate in dynamic environments for robustness assessment.

VIII. DISTRIBUTION OF RESPONSIBILITIES

Please refer to Appendix A: Task List, that provides a detailed distribution of responsibilities for the semester.

IX. ABBREVIATIONS AND ACRONYMS

AGI	Artificial General Intelligence
AI	Artificial Intelligence
EM	Episodic Memory
ACT-R	Adaptive Control of Thought-Rational
BLL	Base-Level Learning
PyACTR	Python Adaptive Control of Thought—Rational
API	Application Programming Interface

ACKNOWLEDGMENT

I would like to express my gratitude to the tools and services that supported the development of this paper:

- Grammarly: For comprehensive proofreading support, enhancing clarity and coherence.
- Otter.ai: For transcribing spoken thoughts, facilitating efficient documentation.
- Speechify: For reading aloud and summarizing academic literature.
- Claude by Anthropic and ChatGPT by OpenAI: For insightful feedback, data interpretation, and improving content clarity.

I also acknowledge the developers of PyACTR and the Python scientific computing community for providing the tools essential to implementing and testing the episodic memory module.

Finally, I extend my gratitude to my professors at the Georgia Institute of Technology for their guidance and support throughout this project.

Ethical Considerations

AI tools used in this research were applied responsibly as aids in proofreading, data interpretation, and content enhancement. The intellectual contributions, analyses, and conclusions presented in this paper are solely my own.

References

[1] D. Hassabis, D. Kumaran, C. Summerfield, and M. M. Botvinick, “Neuroscience-inspired artificial intelligence,” *Neuron*, vol. 95, no. 2, pp. 245–258, Jul. 2017. [Online]. Available: https://doi.org/10.1016/j.neuron.2017.06.011

[2] E. Tulving, “Episodic memory: From mind to brain,” *Annual Review of Psychology*, vol. 53, pp. 1–25, Feb. 2002. [Online]. Available: https://doi.org/10.1146/annurev.psych.53.100901.135114

[3] F. E. Ritter, F. Tehranchi, and J. D. Oury, “ACT-R: A cognitive architecture for modeling cognition,” *Wiley Interdisciplinary Reviews: Cognitive Science*, vol. 10, no. 3, pp. e1488, May 2019. [Online]. Available: https://doi.org/10.1002/wcs.1488

[4] PyACTR Documentation. “PyACTR: A Python-based cognitive architecture,” [Online]. Available: https://github.com/jakdot/pyactr

[5] S. J. Gershman and N. D. Daw, “Reinforcement learning and episodic memory in humans and animals: An integrative framework,” *Annual Review of Psychology*, vol. 68, pp. 101–128, Jan. 2017. [Online]. Available: https://doi.org/10.1146/annurev-psych-122414-033625

[6] H. Eichenbaum, “Memory: Organization and control,” *Annual Review of Psychology*, vol. 68, pp. 19–45, Jan. 2017. [Online]. Available: https://doi.org/10.1146/annurev-psych-010416-044131

[7] P. Thagard, *Mind: Introduction to Cognitive Science*, 2nd ed. Cambridge, MA, USA: MIT Press, 2005. [Online]. Available: https://mitpress.mit.edu/9780262701099/mind/

Appendix

Appendix A: GitHub Repository

The complete source code for this project, including scripts, data, and additional resources, is available on GitHub.
You can access it via the following link:

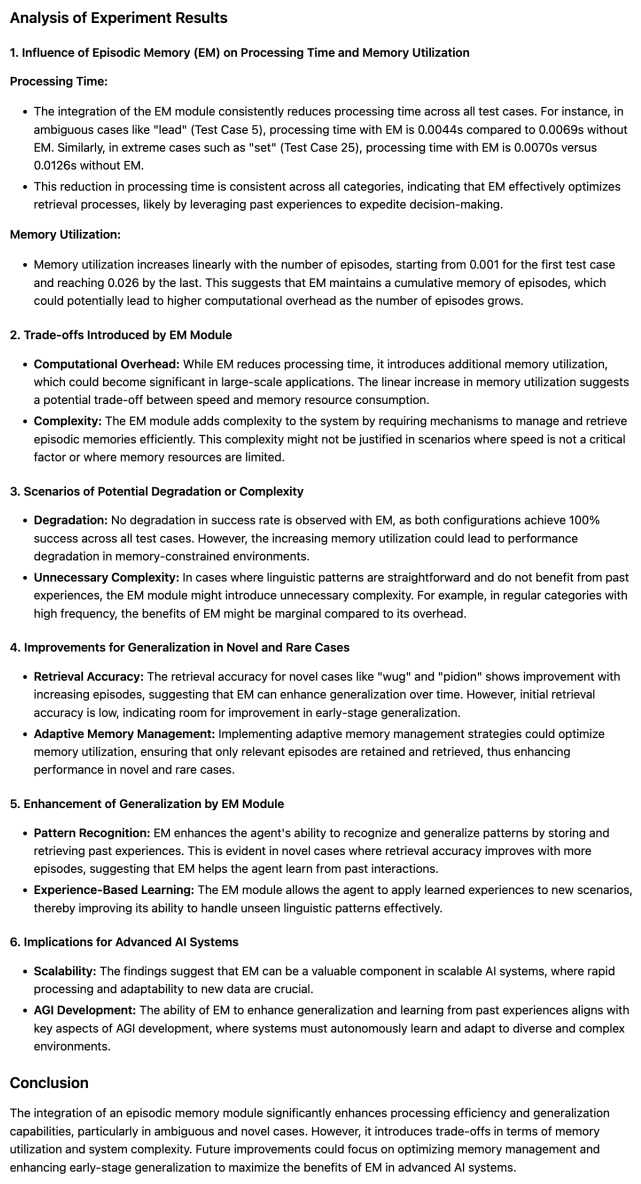
GitHub Repository: episodic-memory-pyactr

The repository includes:

- Python scripts for running experiments and analysis.
- Test cases and episodic memory module implementation.
- Output data, logs, and visualizations.
- Detailed documentation and instructions for usage.

For questions, contributions, or feedback, please visit the repository or submit an issue through GitHub.

Appendix B: LLM Analysis



Analysis of Experiment Results

1. Influence of Episodic Memory (EM) on Processing Time and Memory Utilization

Processing Time:

- The integration of the EM module consistently reduces processing time across all test cases. For instance, in ambiguous cases like "lead" (Test Case 5), processing time with EM is 0.0044s compared to 0.0069s without EM. Similarly, in extreme cases such as "set" (Test Case 25), processing time with EM is 0.0070s versus 0.0126s without EM.
- This reduction in processing time is consistent across all categories, indicating that EM effectively optimizes retrieval processes, likely by leveraging past experiences to expedite decision-making.

Memory Utilization:

- Memory utilization increases linearly with the number of episodes, starting from 0.001 for the first test case and reaching 0.026 by the last. This suggests that EM maintains a cumulative memory of episodes, which could potentially lead to higher computational overhead as the number of episodes grows.

2. Trade-offs Introduced by EM Module

- **Computational Overhead:** While EM reduces processing time, it introduces additional memory utilization, which could become significant in large-scale applications. The linear increase in memory utilization suggests a potential trade-off between speed and memory resource consumption.
- **Complexity:** The EM module adds complexity to the system by requiring mechanisms to manage and retrieve episodic memories efficiently. This complexity might not be justified in scenarios where speed is not a critical factor or where memory resources are limited.

3. Scenarios of Potential Degradation or Complexity

- **Degradation:** No degradation in success rate is observed with EM, as both configurations achieve 100% success across all test cases. However, the increasing memory utilization could lead to performance degradation in memory-constrained environments.
- **Unnecessary Complexity:** In cases where linguistic patterns are straightforward and do not benefit from past experiences, the EM module might introduce unnecessary complexity. For example, in regular categories with high frequency, the benefits of EM might be marginal compared to its overhead.

4. Improvements for Generalization in Novel and Rare Cases

- **Retrieval Accuracy:** The retrieval accuracy for novel cases like "wug" and "pidion" shows improvement with increasing episodes, suggesting that EM can enhance generalization over time. However, initial retrieval accuracy is low, indicating room for improvement in early-stage generalization.
- **Adaptive Memory Management:** Implementing adaptive memory management strategies could optimize memory utilization, ensuring that only relevant episodes are retained and retrieved, thus enhancing performance in novel and rare cases.

5. Enhancement of Generalization by EM Module

- **Pattern Recognition:** EM enhances the agent's ability to recognize and generalize patterns by storing and retrieving past experiences. This is evident in novel cases where retrieval accuracy improves with more episodes, suggesting that EM helps the agent learn from past interactions.
- **Experience-Based Learning:** The EM module allows the agent to apply learned experiences to new scenarios, thereby improving its ability to handle unseen linguistic patterns effectively.

6. Implications for Advanced AI Systems

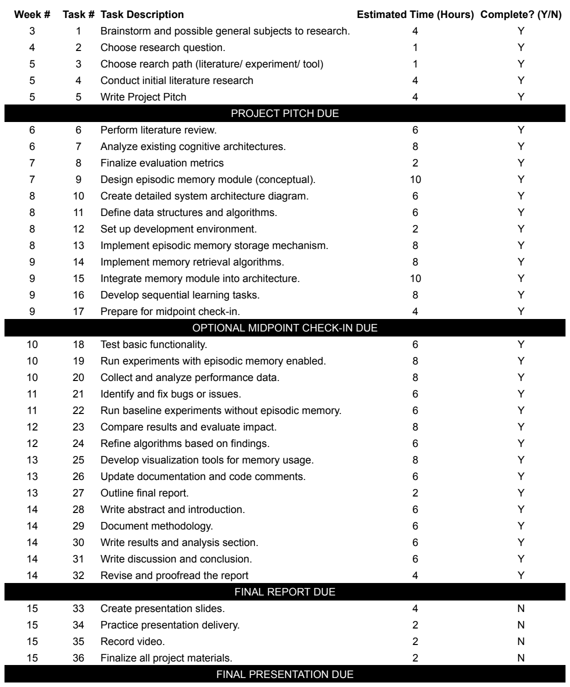
- **Scalability:** The findings suggest that EM can be a valuable component in scalable AI systems, where rapid processing and adaptability to new data are crucial.
- **AGI Development:** The ability of EM to enhance generalization and learning from past experiences aligns with key aspects of AGI development, where systems must autonomously learn and adapt to diverse and complex environments.

Conclusion

The integration of an episodic memory module significantly enhances processing efficiency and generalization capabilities, particularly in ambiguous and novel cases. However, it introduces trade-offs in terms of memory utilization and system complexity. Future improvements could focus on optimizing memory management and enhancing early-stage generalization to maximize the benefits of EM in advanced AI systems.

Fig. 4. Detailed LLM Analysis Output

APPENDIX C: TASK LIST



Week #	Task #	Task Description	Estimated Time (Hours)	Complete? (Y/N)
3	1	Brainstorm and possible general subjects to research.	4	Y
4	2	Choose research question.	1	Y
5	3	Choose rearch path (literature/ experiment/ tool)	1	Y
5	4	Conduct initial literature research	4	Y
5	5	Write Project Pitch	4	Y
		PROJECT PITCH DUE		
6	6	Perform literature review.	6	Y
6	7	Analyze existing cognitive architectures.	8	Y
7	8	Finalize evaluation metrics	2	Y
7	9	Design episodic memory module (conceptual).	10	Y
8	10	Create detailed system architecture diagram.	6	Y
8	11	Define data structures and algorithms.	6	Y
8	12	Set up development environment.	2	Y
8	13	Implement episodic memory storage mechanism.	8	Y
9	14	Implement memory retrieval algorithms.	8	Y
9	15	Integrate memory module into architecture.	10	Y
9	16	Develop sequential learning tasks.	8	Y
9	17	Prepare for midpoint check-in.	4	Y
		OPTIONAL MIDPOINT CHECK-IN DUE		
10	18	Test basic functionality.	6	Y
10	19	Run experiments with episodic memory enabled.	8	Y
10	20	Collect and analyze performance data.	8	Y
11	21	Identify and fix bugs or issues.	6	Y
11	22	Run baseline experiments without episodic memory.	6	Y
12	23	Compare results and evaluate impact.	8	Y
12	24	Refine algorithms based on findings.	6	Y
13	25	Develop visualization tools for memory usage.	8	Y
13	26	Update documentation and code comments.	6	Y
13	27	Outline final report.	2	Y
14	28	Write abstract and introduction.	6	Y
14	29	Document methodology.	6	Y
14	30	Write results and analysis section.	6	Y
14	31	Write discussion and conclusion.	6	Y
14	32	Revise and proofread the report	4	Y
		FINAL REPORT DUE		
15	33	Create presentation slides.	4	N
15	34	Practice presentation delivery.	2	N
15	35	Record video.	2	N
15	36	Finalize all project materials.	2	N
		FINAL PRESENTATION DUE		

Fig. 5. Task List Overview