AI Dungeon Master: Memory-Driven Narrative System

Technical Report

TEAM NEXUS

Ruchir Sharma
Sathvik Prabhu

Contents

Project: Memory-Based Narrative System (2025)

1 Abstract

Tabletop role-playing games thrive on consistent storytelling and world coherence—roles traditionally upheld by a Dungeon Master (DM). This project presents an AI Dungeon Master that leverages Large Language Models (LLMs) and a hybrid memory architecture to deliver persistent, evolving, and interactive storytelling experiences.

While LLMs can generate compelling text, they lack long-term recall and contextual consistency. This system addresses these limitations by integrating:

- **Working memory** for short-term interaction recall,
- **Episodic memory** for long-term contextual recall using vector embeddings, and
- **Semantic + graph-based world memory** to maintain relationships between entities and events.

This modular design enables the AI DM to remember past events, track characters, and maintain continuity across multiple gameplay sessions—achieving the project's goals of creativity, coherence, and persistence.

2 System Architecture Overview

The system is implemented as five modular components, each in its own Python file:

Module	Responsibility
`vector_store.py`	Implements episodic (long-term) memory using vector embeddings.
`graph_store.py`	Maintains entity-relationship knowledge graph using Neo4j.
`llm_client.py`	Handles LLM communication using Groq-hosted models (LLaMA 3.1 8B).
`memory_manager.py`	Core controller integrating all memory layers.
`main.py`	Implements the interactive REPL loop for storytelling.
`utils.py`, `config.py`	Helper utilities and configuration handling.

3 Memory Architecture

A robust memory system balancing short-term and long-term recall is the cornerstone of this project. Our architecture satisfies this through a three-tiered design:

3.1 Working Memory (Short-Term)

Implemented in `MemoryManager` via a deque buffer (`self.working_buffer`).

- Stores the last $n \approx 5$ player–DM interactions.
- Maintains immediate conversational context and recent decisions.
- Enables short-term recall scoring in evaluation (15 points).

3.2 Episodic Memory (Long-Term)

Implemented in `EpisodicStore` (`vector_store.py`):

- Stores events as text + embeddings using `SentenceTransformer('all-MiniLM-L6-v2')`.
- Supports semantic retrieval (RAG) using cosine similarity.
- Persists data across sessions in `episodic_store.json`.
- Enables consistent long-term storytelling across ∼30 turns.

3.3 Semantic Graph Memory (World Knowledge)

Implemented in `GraphStore` using Neo4j:

- Nodes represent entities (Player, NPC, Item, Location, etc.).
- Edges represent relationships (e.g., `Kael KNOWS Goblin`).
- Updated dynamically through entity extraction from LLM outputs.

3.4 Integrated Memory Retrieval (RAG Loop)

Each player input triggers:

1. Retrieval of recent actions from working memory.
2. Semantic search from episodic memory for similar past events.
3. Entity extraction → graph query → world context synthesis.

Combined context is then passed to the LLM for narrative continuation, ensuring creativity with consistency.

4 Module-by-Module Explanation

4.1 vector_store.py – Episodic Memory

- Stores narrative events and embeddings.
- Supports `add_event()` and `query()` for retrieval.
- Provides persistence via JSON serialization.

Key Concepts: Semantic embeddings, cosine similarity search, persistent vector storage, and retrieval-augmented generation (RAG).

4.2 graph_store.py – Graph Memory

- Uses Neo4j graph database to maintain world knowledge.
- `merge_entity()` and `merge_relationship()` create and link nodes.
- Supports context retrieval for prompts and world summaries.

Key Concepts: Knowledge graphs, Cypher queries, relationship-based retrieval, and entity normalization.

4.3 llm_client.py – LLM Integration Layer

- Connects to Groq-hosted `LLaMA 3.1` model.
- Two LLM roles:
  - **Story LLM** – creative narrative generation.
  - **Extractor LLM** – structured entity extraction.

Key Concepts: Prompt conditioning, JSON-based structured output, and regex cleanup for robustness.

4.4 memory_manager.py – Core Orchestrator

- Connects all three memory layers.
- Builds dynamic context using working buffer + episodic + graph memory.
- Executes full generation cycle (`generate_and_update()`).

Key Concepts: Context assembly, memory balancing, and multi-source retrieval.

4.5 main.py – Interaction Loop

- Provides REPL-based gameplay.
- Commands include `/memory`, `/context`, `/dump`, and `/reset`.

5 How the System Meets Project Evaluation Criteria

Criterion	Implementation Evidence	Points
Gameplay & Robustness	REPL-based, stable gameplay	25
Short-Term Recall	Working memory via deque	15
Long-Term Recall	Vector-based semantic recall (RAG)	25
Architecture & Code Quality	Modular, documented design	15
Presentation & Report	Detailed documentation provided	20
Bonus – Character Memory	Persistent Neo4j entity state	+10

6 Workflow and Data Flow (Flowchart)

The following flowchart illustrates how data and control move through the AI Dungeon Master's architecture—from player input, through memory retrieval, to narrative generation and world updates.

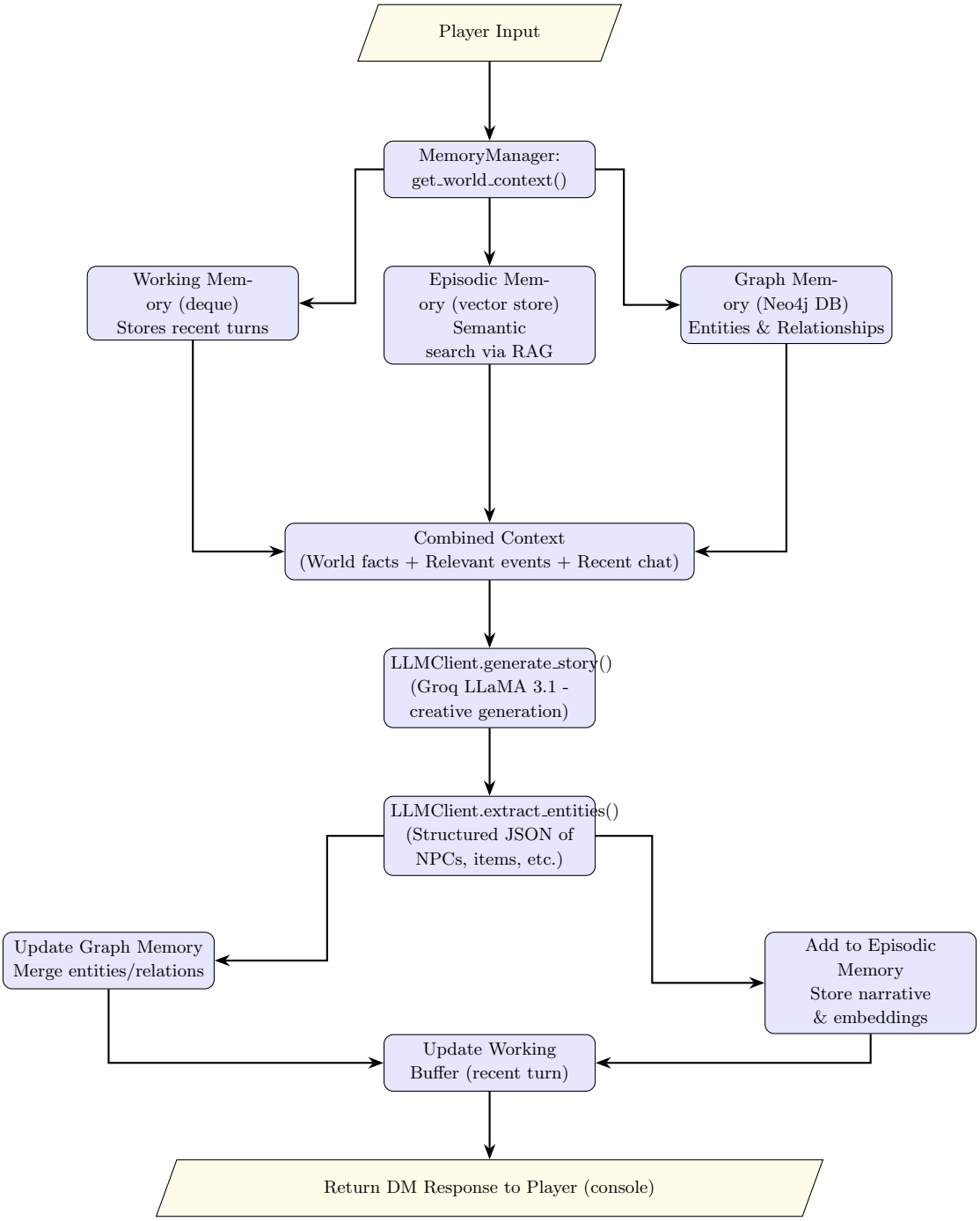

Figure 1: Workflow of the AI Dungeon Master System

This diagram shows how all components interact to form a dynamic feedback loop that maintains short-term, long-term, and semantic memory continuity.

7 Key Highlights and Innovations

- **Hybrid Memory Architecture:** Combines embeddings, symbolic graph reasoning, and short-term buffers.
- **Dynamic Entity Evolution:** Characters and NPCs persist and evolve over turns.
- **RAG-Based Recall:** Semantic, not keyword-based, retrieval ensures nuanced continuity.
- **Self-Healing Extraction:** Regex cleanup handles imperfect LLM JSON outputs.
- **Resettable & Modular:** Memory layers can reset for new campaigns.

8 Conclusion and Future Enhancements

This project successfully realizes a memory-augmented AI Dungeon Master capable of generating, remembering, and reasoning about story elements persistently. By combining vector-based episodic memory, graph-based world knowledge, and working memory buffers, the system maintains narrative continuity and logical consistency across multiple sessions.

Future Work:

- Integrate emotion/state tracking for NPCs.
- Add quest management and dynamic objectives.
- Optimize vector retrieval with FAISS or Chroma.
- Expand to a web-based interactive interface.
