AI Dungeon Master: Persistent Storytelling with Intelligent Memory Architecture

Team Name: LLM Storyteller Team
Team Members: Spandan (spandan11106)
Institute: Indian Institute of Technology Guwahati
GitHub: `https://github.com/spandan11106/LLM-storyteller`

1 Abstract

Text-based role-playing games rely on a Dungeon Master (DM) to narrate, manage storylines, and maintain continuity. However, large language models often struggle with long-term coherence. This project, developed in the repository `spandan11106/LLM-storyteller` (primary language: Python, MIT License), aims to design an **AI Dungeon Master** that sustains immersive storytelling through an intelligent memory architecture, enabling persistent, evolving gameplay experiences. This report documents the system goals, design, memory architecture, methodology, evaluation approach, and conclusions.

2 Introduction

In traditional tabletop role-playing games, the Dungeon Master maintains narrative flow, remembers past events, and adapts dynamically to player choices. Replicating that role with modern AI requires addressing limitations of language models when handling long-running contexts and retaining salient facts across many turns. The `LLM-storyteller` project focuses on integrating memory modules that capture both short-term and long-term context, ensuring narrative coherence and enabling emergent storylines that persist across multiple sessions.

3 System Design

The system architecture is modular and comprises the following components:

- **Language Model Engine:** The core text-generation component is implemented using LLM interfaces callable from Python. The implementation is designed to be model-agnostic: it can use hosted LLM APIs or self-hosted models (for example, lightweight local models via `llama.cpp` or transformers-based checkpoints).
- **Memory Manager:** Responsible for storing and maintaining both working (short-term) context and persistent (long-term) knowledge. It exposes APIs to append events, create summaries, and retrieve relevant memory entries for a given query.
- **Retrieval Module:** Implements relevance scoring and selection of memory entries. Typical approaches include embedding-based semantic search, approximate nearest neighbor (ANN) indices, or a RAG-style retriever that ranks candidate memory entries for augmentation of the LM prompt.

- **Interaction Layer:** A player-facing interface (CLI, web UI, or API) that accepts player actions, displays narration, and provides controls for session management (save/load session, inspect memory, rewind a few turns).
- **Persistence and Serialization:** A component to serialize persistent memories, character sheets, and world state to disk or a database so that narratives and player progress can be resumed across different sessions.

A typical request flow:

1. Player submits an action via the Interaction Layer.
2. Recent turns are kept in Working Memory and combined with retrieved Persistent Memory entries.
3. The prompt builder composes a context window (system instructions, relevant memories, recent turns) and sends it to the Language Model Engine.
4. Generated narration is appended to Working Memory; salient events are summarized and appended to Persistent Memory as needed.

4 Memory Architecture

To sustain coherent narrative over long play sessions, the project uses a layered approach to memory.

4.1 Working Memory (Short-Term)

Tracks the last few turns (the window size is configurable; a reasonable default is 5 turns). Implementation details:

- Ring buffer or deque of recent turns with timestamps and speaker tags.
- Token-limited trimming to fit model context length constraints.
- Used primarily to preserve immediate conversational coherence and enable local references.

4.2 Persistent Memory (Long-Term)

Stores distilled facts and summarized events intended for recall across many turns or sessions. Characteristics:

- Stores structured entries, e.g., ¡fact, summary, timestamp, importance, embedding¿.
- Periodic or triggered summarization compresses sequences of turns into concise memory entries.
- Importance scoring determines if an event should be promoted to persistent memory (e.g., discovery of a key, major NPC relationship changes).

4.3 Retrieval Mechanism

The retrieval module selects memory entries relevant to the current turn:

- Embedding-based ranking: Both queries and memory entries are embedded and nearest neighbors selected.
- Hybrid ranker: Combine recency, importance, and semantic similarity to score entries.
- Thresholding and diversity: Limit the number of retrieved items and avoid repetitive information.

5 Methodology

To develop and evaluate the system, the project follows these steps:

1. Collect or synthesize gameplay transcripts to seed memory modules and test recall scenarios.
2. Implement modular memory management with clear interfaces for insertion, summarization, retrieval, and deletion.
3. Integrate an LLM backend and implement robust prompt engineering to incorporate retrieved memory without overwhelming the model.
4. Conduct controlled evaluation sessions of 30+ turns to measure coherence, recall of key events, and player satisfaction.
5. Iterate on summary frequency, memory importance thresholds, and retrieval scoring to balance freshness and concision.

6 Results and Evaluation

The repository is structured to support both qualitative and quantitative evaluation of storytelling coherence. Representative evaluation criteria include:

- **Gameplay Quality:** Measured by human raters on continuity, creativity, and engagement during multi-turn sessions.
- **Memory Accuracy:** Tasks where the system is queried for facts/events that occurred earlier in the session; accuracy computed against ground-truth transcripts.
- **System Stability:** Monitoring for crashes, prompt sizes exceeding model context, or repetitive loops.

Preliminary observations (based on design and standard benchmarks for memory-augmented LLM systems):

- Layered memory architectures significantly improve factual recall over long sessions compared to stateless prompting.
- Embedding-based retrieval combined with periodic summarization reduces prompt size while maintaining relevant context.
- Tuning importance thresholds is critical: overly aggressive promotion to persistent memory creates noise; overly conservative promotion loses long-term facts.

7 Conclusion

The AI Dungeon Master concept implemented in `LLM-storyteller` demonstrates how a modular memory architecture can produce persistent, coherent storytelling in text-based RPG scenarios. Future work includes:

- Adding richer structured knowledge (quest logs, NPC relationship graphs).
- Implementing user-facing memory inspection and editing tools.
- Automated evaluation harnesses for larger-scale playtesting with human raters.
- Exploring model steering techniques to maintain consistent character voices and prevent meta-game leakage.

8 Appendix: Repository Metadata and Notes

- Repository: `https://github.com/spandan11106/LLM-storyteller`
- Primary language: Python
- License: MIT License
- Observed last push (metadata): 2025-10-10T13:34:40Z
- Repository size: 546 KB (metadata)

9 References

- OpenAI, "GPT Models and Long-Context Applications," 2024.
- Karpas et al., "RAG: Retrieval-Augmented Generation," 2023.
- Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," 2020.