Procesamiento de Lenguaje Natural

Informe del TP Final

Integrantes:
Nicolas Duclos - D-4385/1

Junio 2025

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Universidad Nacional de Rosario

Av. Pellegrini 250 - 2000 Rosario – Argentina

Sistema RAG y Agente Autónomo para Tiny Towns

1. Introducción

Este informe analiza el desarrollo de un sistema de Retrieval-Augmented Generation (RAG) y un agente autónomo diseñado para responder consultas sobre el juego de mesa Tiny Towns. El sistema integra múltiples componentes de Procesamiento del Lenguaje Natural (NLP), incluyendo búsqueda semántica y por palabras clave híbrida, procesamiento de datos estructurados (tablas estadísticas), consultas en grafos (relaciones entre entidades), y generación de respuestas contextuales mediante modelos de lenguaje (LLMs).

El objetivo principal es evaluar las decisiones técnicas tomadas, identificar limitaciones y proponer mejoras para optimizar el rendimiento del sistema.

2. Descripción del Sistema

El sistema se compone de dos módulos principales:

2.1. Sistema RAG (TinyTownsRAG)

Funcionalidades principales:

- Búsqueda en documentos: Fragmentación de textos, embeddings y recuperación semántica
- Consultas estadísticas: Procesamiento de DataFrames con filtros generados por LLM
- Consultas en grafos: Generación de queries Cypher para Neo4j
- Clasificación de intenciones: Determina si la consulta es sobre reglas, estadísticas o relaciones

2.2. Agente Autónomo (TinyTownsAgent)

Arquitectura ReAct: Combina razonamiento (Reasoning) y acción (Action) mediante herramientas predefinidas.

Herramientas disponibles:

- Búsqueda en documentos (doc_search)
- Consultas tabulares (table_search)
- Búsqueda en grafos (graph_search)
- Acceso a DuckDuckGo y Wikipedia para información externa

3. Justificación Técnica de Modelos y Métodos

3.1. Clasificador de Intención

Modelo seleccionado:

- Primera opción: cardiffnlp/twitter-roberta-base-sentiment-latest (fine-tuning para clasificación)
- Fallback: Few-shot prompting con un LLM (google/gemma-2b)

Decisiones clave:

RoBERTa-base:

- Ventaja: Alto rendimiento en tareas de clasificación de texto
- Limitación: Requiere fine-tuning para adaptarse a intenciones específicas de Tiny Towns

Few-shot prompting:

- Se usan ejemplos predefinidos para cada intención (ej: document_search, statistics)
- Problema: Dependencia de la calidad del LLM (en pruebas, falló por errores 404 en la API)

Alternativas propuestas:

1. Entrenar un modelo específico con datos etiquetados del dominio (ej: BERTimbau para español/portugués)
2. Usar embeddings + clustering (K-means sobre embeddings de consultas históricas)

3.2. Modelo de Embedding

Modelo seleccionado: paraphrase-multilingual-MiniLM-L12-v2 (Sentence Transformers)

Ventajas:

- Multilingüe: Ideal para consultas en varios idiomas
- Eficiencia: 12 capas (menos costoso que modelos grandes como BERT)
- Optimizado para similitud semántica: Usa cosine similarity, clave para RAG

Mejora sugerida: Probar bge-small-en-v1.5 (mejor rendimiento en retrieval según benchmarks recientes).

3.3. Fragmentación de Texto (Text Splitter)

Enfoque implementado:

- Chunk size: 500 tokens
- Overlap: 50 tokens

Justificación:

- Overlap: Evita pérdida de contexto en bordes de fragmentos (ej: una regla dividida en dos chunks)
- Tamaño de chunk: Balance entre capturar contexto y eficiencia. Manuales de juegos suelen tener párrafos largos (500 tokens ≈ 2-3 párrafos)

Problemas detectados: Algunas consultas recuperan chunks sin información relevante.

Mejoras propuestas:

1. Fragmentación semántica: Usar modelos como SemanticChunker (LangChain) para dividir por temas
2. Chunks dinámicos: Ajustar el tamaño basado en la densidad de información (TF-IDF)

3.4. Modelo de Lenguaje (LLM)

Modelos usados:

1. Gemma-2b: Para generación de SQL y Cypher
2. Zephyr-7b-beta: En el agente (ReAct)

Elección de Gemma-2b:

- Ventajas: Modelo liviano (2B parámetros) pero eficaz para tareas estructuradas. Soporta instrucciones en español
- Problema: API no disponible en pruebas (404 Not Found)

Elección de Zephyr-7b-beta:

- Ventajas: Optimizado para diálogos y razonamiento paso a paso (ReAct). Buen balance entre rendimiento y coste computacional

4. Fallos Detectados y Soluciones

4.1. Problemas en la Implementación

1. LLM no disponible:

- Error 404 al llamar a google/gemma-2b
- Solución: Usar un modelo local (Llama-3) o cambiar a OpenAI/Anthropic

2. Neo4j no configurado:

- Error en conexión (bolt://localhost:7687)
- Solución: Usar Neo4j Aura (gratis para pruebas) o implementar un grafo en memoria con NetworkX para desarrollo

3. Clasificador de intención no preciso:

- En pruebas, clasificó mal el 100% de las consultas
- Causa: Posible falta de fine-tuning o ejemplos insuficientes
- Solución: Recopilar más datos etiquetados. Usar embeddings + KNN para clasificación semi-supervisada

4. Respuestas genéricas del RAG:

- Ejemplo: "No pude generar una respuesta"
- Causa: Fallo en el LLM o falta de contexto en los chunks
- Solución: Añadir verificación de fuentes (cross-encoder/ms-marco-MiniLM-L-6-v2). Fine-tuning del LLM con manuales de Tiny Towns

5. Mejoras Propuestas

5.1. Robustez del Sistema

- API Fallback: Si Hugging Face falla, cambiar a otra fuente o modelo local
- Caché de embeddings: Almacenar embeddings precalculados para reducir latencia

5.2. Precisión en Búsqueda

- Re-ranker: Añadir un modelo de re-ranking (ej: BAAI/bge-reranker-base)
- Hybrid Search: Mejorar la ponderación entre TF-IDF y embeddings

5.3. Optimización del Agente

- Memoria a largo plazo: Usar ConversationSummaryMemory (LangChain)
- Autoevaluación: Que el agente valide sus respuestas antes de enviarlas

5.4. Evaluación Cuantitativa

Métricas:

- Precisión de recuperación (Recall@k): % de respuestas correctas en top-k
- Latencia: Tiempo de respuesta por consulta
- Satisfacción del usuario: Encuestas post-interacción

6. Conclusión

El sistema implementado demuestra un enfoque para integrar RAG y agentes autónomos en el dominio del juego de mesa. Sin embargo, los fallos técnicos (APIs no disponibles, bajo rendimiento del clasificador) limitan su eficacia. Las mejoras propuestas se centran en:

1. **Robustez:** Alternativas a APIs externas
2. **Precisión:** Fine-tuning de modelos y mejoras en retrieval
3. **Evaluación:** Métricas cuantitativas para medir rendimiento real