Report Tecnico: Synapse - Neurosymbolic AI e RAG per l'Apprendimento

Gruppo Synapse

November 24, 2025

Abstract

Questo documento illustra l'architettura tecnica di *Synapse*, un sistema intelligente per la generazione di materiale didattico. Il progetto integra tecnologie di **Retrieval-Augmented Generation (RAG)** per la gestione della conoscenza e metodologie **Neurosymbolic** (in particolare il pattern di *Reflection*) per garantire la qualità pedagogica degli output generati dai Large Language Models (LLM).

1 Introduzione

L'obiettivo di Synapse è superare i limiti dei tradizionali generatori di quiz/flashcard basati su LLM, che spesso producono contenuti superficiali o affetti da allucinazioni. Abbiamo adottato un approccio ibrido che combina la capacità generativa delle reti neurali con regole logiche e strutturate (simboliche) per guidare il processo di creazione.

2 Neurosymbolic AI: Il Pattern di Reflection

Il cuore "intelligente" del sistema non è una singola chiamata all'LLM, ma un flusso di lavoro iterativo noto come **Reflection**. Questo pattern emula il processo di revisione umana: generazione, critica e raffinamento.

2.1 Perché la Reflection?

Gli LLM tendono a soddisfare la richiesta dell'utente nel modo più diretto possibile ("Lazy AI"), spesso ignorando sfumature pedagogiche complesse. Imporre tutte le regole in un unico prompt spesso fallisce. La Reflection scompone il problema:

1. Un agente "Creatore" genera una bozza.
2. Un agente "Critico" analizza la bozza senza doverla generare.
3. Un agente "Revisore" applica le correzioni.

2.2 Implementazione e Prompting

Di seguito mostriamo come abbiamo ingegnerizzato i prompt per ciascuna fase (codice estratto da `reflection_service.py`).

2.2.1 Fase 1: Draft (Generazione della Bozza)

Il sistema istruisce l'LLM ad agire come un esperto di metacognizione. Il prompt impone regole "atomiche" basate sui principi di Andy Matuschak.

```
prompt = f"""Sei un esperto di apprendimento...
Segui queste 5 REGOLE ASSOLUTE:
1. Focalizzata: La domanda deve riguardare UN SOLO concetto.
2. Precisa: Non deve essere ambigua.
3. Coerente: La risposta deve essere l'unica corretta.
4. Chiedi il "Perch ": Preferisci domande sulle implicazioni.
5. Sforzo Cognitivo: La risposta NON deve essere intuibile dalla domanda
   .

Restituisci la risposta in formato JSON..."""
```

Listing 1: Prompt per la generazione della bozza

2.2.2 Fase 2: Critique (Analisi Critica)

Questa è la fase più "simbolica". Chiediamo all'LLM di valutare l'output precedente secondo criteri rigidi, quasi come se eseguisse un test unitario semantico.

```
prompt = f"""Sei un critico esperto...
Valuta la flashcard ESCLUSIVAMENTE secondo queste 5 REGOLE:
1. Focalizzata: Chiede un solo concetto?
2. Precisa:    ambigua?
3. Contesto: La risposta    basata SOLO sul contesto?
...
Fornisci una critica COSTRUTTIVA. Se non rispetta le regole, spiega COSA
    migliorare."""
```

Listing 2: Prompt per la critica

2.2.3 Fase 3: Refine (Raffinamento)

Se la critica è negativa, il sistema passa la bozza originale e la critica al modello per generare una versione migliorata.

```
prompt = f"""
FLASHCARD ORIGINALE: ...
CRITICA RICEVUTA: {critique}
Migliora la flashcard tenendo conto della critica.
Assicurati che la flashcard migliorata affronti i problemi evidenziati."
   ""
```

Listing 3: Prompt per il raffinamento

Questo ciclo si ripete fino a quando la critica è positiva o si raggiunge un numero massimo di iterazioni (es. 2), garantendo un controllo di qualità automatico.

3 Retrieval-Augmented Generation (RAG)

Per vincolare le risposte ai soli materiali di studio forniti (PDF), abbiamo implementato una pipeline RAG avanzata.

3.1 Chunking Ricorsivo

Uno degli aspetti più critici del RAG è come si divide il testo. Un taglio netto a N caratteri può spezzare frasi o concetti a metà. Abbiamo implementato un algoritmo di **Recursive Character Text Splitting** in `rag_service.py`.

L'algoritmo prova a dividere il testo usando una gerarchia di separatori:

1. `\n\n` (Paragrafi): Cerca di mantenere intatti i paragrafi.
2. `\n` (Linee): Se un paragrafo è troppo lungo, divide alle nuove righe.
3. `.` (Frasi): Se ancora troppo lungo, divide ai punti.
4. `" "` (Parole): Ultima risorsa, divide agli spazi.

Questo approccio preserva la semantica del testo molto meglio del chunking a dimensione fissa.

3.2 Architettura del Sistema RAG

- **Vector Store:** Utilizziamo **Qdrant** locale per memorizzare i vettori. Ogni chunk è associato a metadati (ID documento, nome file) per la tracciabilità.
- **Embedding:** Il sistema è agnostico. Supporta:
  - **Locale:** Ollama con modello *nomic-embed-text* (privacy totale).
  - **Cloud:** Google Gemini Embedding (prestazioni elevate).
- **Retrieval:** Usiamo la *Cosine Similarity* con una soglia di score (configurabile via `.env`, es. 0.25) per filtrare risultati poco pertinenti.

4 Conclusioni

L'integrazione di queste tecnologie ha prodotto risultati tangibili:

- **Qualità:** Le flashcard generate tramite Reflection sono risultate più profonde e meno banali rispetto alla generazione diretta (Zero-Shot).
- **Affidabilità:** Il RAG con chunking ricorsivo ha eliminato le allucinazioni, costringendo il modello a citare solo fatti presenti nei PDF.
- **Flessibilità:** L'architettura modulare permette di scambiare i modelli (Ollama/Gemini) senza riscrivere la logica applicativa.