

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

Gruppo Synapse

November 24, 2025

## Abstract

Questo documento fornisce un'analisi tecnica approfondita di *Synapse*, un sistema intelligente progettato per supportare lo studio universitario attraverso la generazione automatica di materiale didattico. Il report esplora l'architettura del sistema, focalizzandosi sull'integrazione di tecnologie **Retrieval-Augmented Generation (RAG)**, metodologie **Neurosymbolic** (Reflection Pattern), integrazione con servizi di ricerca web e funzionalità di esportazione avanzate.

## Contents

# 1 Introduzione

L'evoluzione dei Large Language Models (LLM) ha aperto nuove possibilità nel campo dell'educazione assistita. Tuttavia, l'uso "naive" di questi modelli presenta limitazioni significative: allucinazioni, superficialità nelle risposte e mancanza di aderenza ai materiali di studio specifici.

*Synapse* nasce per risolvere questi problemi attraverso un'architettura ibrida che combina:

- **Componente Neurale:** La capacità generativa e di comprensione del linguaggio naturale degli LLM (Gemini, Ollama).
- **Componente Simbolica:** Regole logiche, vincoli strutturali e flussi di controllo deterministici che guidano l'AI.
- **Knowledge Retrieval:** Un sistema RAG per ancorare le risposte a documenti verificati.

## 2 Architettura del Sistema

Il sistema è costruito in Python e segue un'architettura modulare a servizi.

### 2.1 Panoramica dei Servizi

Il core logico risiede nella directory `services/`, che disaccoppia le responsabilità:

- `ai_service.py`: Astrazione per l'interazione con gli LLM (supporta Google Gemini e modelli locali via Ollama).
- `rag_service.py`: Gestisce l'indicizzazione dei documenti, il chunking e il retrieval vettoriale.
- `reflection_service.py`: Implementa la logica neurosymbolic per la generazione e validazione delle flashcard.
- `web_search_service.py`: Fornisce contesto aggiuntivo dal web quando i documenti locali non sono sufficienti.
- `export_service.py`: Gestisce la conversione dei dati in formati interoperabili (CSV, Anki).

### 2.2 Interfaccia Utente

L'interfaccia grafica è realizzata con **PyQt6**, offrendo un'esperienza desktop nativa e reattiva. La struttura UI (`ui/`) separa la logica di presentazione (`'main_window.py'`, `'subject_window.py'`) dai dialoghi di configurazione (`'settings_dialog.py'`).

## 3 Neurosymbolic AI: Il Pattern di Reflection

Una delle innovazioni principali di *Synapse* è l'implementazione del pattern di **Reflection**. Invece di accettare passivamente l'output dell'LLM, il sistema instaura un ciclo di auto-miglioramento.

### 3.1 Il Ciclo Draft-Critique-Refine

Il processo di generazione di una flashcard segue tre fasi distinte, orchestrate da `reflection_service.py`

#### 3.1.1 1. Draft (Generazione della Bozza)

L'LLM viene istruito per agire come un esperto di metacognizione. Il prompt impone regole "atomiche" basate sui principi di Andy Matuschak, richiedendo un output JSON strutturato.

```
1 prompt = f"""Sei un esperto di apprendimento e metacognizione...
2 Segui queste 5 REGOLE ASSOLUTE:
3 1. Focalizzata: La domanda deve riguardare UN SOLO concetto.
4 2. Precisa: Non deve essere ambigua.
5 3. Coerente: La risposta deve essere l'unica corretta.
6 4. Chiedi il "Perch ": Preferisci domande sulle implicazioni.
7 5. Sforzo Cognitivo: La risposta NON deve essere intuibile dalla domanda
8 .
9 Restituisci la risposta in formato JSON..."""
```

Listing 1: Prompt per la generazione della bozza

#### 3.1.2 2. Critique (Analisi Critica)

In questa fase, il sistema assume un ruolo "avversario". Un secondo prompt chiede all'LLM di valutare la bozza appena generata. Non si tratta di una semplice revisione, ma di una validazione rispetto a criteri specifici.

```
1 prompt = f"""Sei un critico esperto...
2 Valuta la flashcard ESCLUSIVAMENTE secondo queste 5 REGOLE:
3 1. Focalizzata: Chiede un solo concetto? 0      troppo ampia?
4 2. Precisa:      ambigua?
5 3. Contesto: La risposta      corretta e basata SOLO sul contesto?
6 ...
7 Fornisci una critica COSTRUTTIVA in 2-3 frasi."""
```

Listing 2: Prompt per la critica

#### 3.1.3 3. Refine (Raffinamento)

Se la critica evidenzia difetti (rilevati tramite keyword matching come "non focalizzata", "vaga", ecc.), il sistema invoca nuovamente l'LLM passando:

1. La flashcard originale.
2. La critica ricevuta.
3. Il contesto originale.

Il modello è forzato a produrre una nuova versione che risolva specificamente i problemi segnalati.

## 4 Retrieval-Augmented Generation (RAG)

Il modulo RAG (`rag_service.py`) è il cuore della "memoria" di Synapse. Questa tecnologia permette di superare il limite della conoscenza statica degli LLM, fornendo accesso dinamico ai documenti caricati dall'utente (PDF, slide).

### 4.1 Come funziona: Un Esempio Semplice

Per comprendere il funzionamento, consideriamo un esempio pratico. Immaginiamo che l'utente chieda: *"Cos'è il pattern di Reflection?"*.

1. **Retrieval (Recupero):** Il sistema converte la domanda in un vettore numerico (embedding) e cerca nel database (Qdrant) i frammenti di testo (chunks) dei PDF che sono semanticamente più vicini alla domanda.
2. **Augmentation (Arricchimento):** I frammenti trovati vengono inseriti nel prompt di sistema. Il prompt diventa simile a:

*"Usa SOLO le seguenti informazioni per rispondere: [Testo estratto dal PDF: 'La Reflection è un pattern...']. Domanda: Cos'è il pattern di Reflection?"*

3. **Generation (Generazione):** L'LLM genera la risposta basandosi esclusivamente sul contesto fornito, garantendo che la spiegazione sia fedele al materiale didattico e non generica.

### 4.2 Chunking Ricorsivo

La qualità del retrieval dipende drasticamente da come i documenti vengono suddivisi (chunking). Un approccio ingenuo, che taglia il testo ogni  $N$  caratteri fissi, rischia di spezzare frasi a metà o separare concetti legati tra loro, rendendo il contesto incomprensibile per l'LLM.

Synapse adotta una strategia di **Recursive Character Text Splitting**, che mira a preservare la semantica del testo rispettando la struttura naturale del linguaggio. L'algoritmo non taglia arbitrariamente, ma cerca di dividere il testo usando una gerarchia di separatori, in ordine di priorità:

1. **Paragrafi (\n\n):** Tenta prima di dividere il testo in blocchi logici completi (paragrafi).
2. **Linee (\n):** Se un paragrafo è ancora troppo lungo per essere gestito, lo suddivide per righe.
3. **Frasi (. ):** Se necessario, scende a livello di singola frase.
4. **Caratteri:** Solo come ultima risorsa taglia a metà una frase.

Questo approccio garantisce che ogni "chunk" inviato al modello contenga, per quanto possibile, un pensiero completo e coerente, migliorando significativamente la qualità delle risposte generate.

## 4.3 Vector Store e Embedding

Il sistema utilizza **Qdrant** in modalità embedded (salvataggio su disco locale) per memorizzare i vettori. Questo evita la necessità di server esterni complessi da configurare.

Per gli embedding, il sistema è ibrido:

- **Locale (Ollama):** Utilizza modelli come `nomic-embed-text`, garantendo privacy totale e funzionamento offline.
- **Cloud (Gemini):** Utilizza le API di Google per embedding ad alte prestazioni (`gemini-embedding-001`), ideale per chi non ha hardware potente.

La ricerca avviene tramite **Cosine Similarity**, con una soglia di rilevanza configurabile (default 0.25) per scartare risultati rumorosi.

## 5 Integrazione Ricerca Web

Per garantire che le risposte siano sempre aggiornate e complete, Synapse integra un modulo di ricerca web (`web_search_service.py`) che interviene quando i documenti locali non contengono le informazioni necessarie.

### 5.1 Funzionamento del Servizio

Il sistema non si limita a effettuare una semplice ricerca su Google, ma utilizza uno strumento specializzato per agenti AI (in questo caso, **Tavily**) ottimizzato per l'estrazione di contenuti. L'obiettivo non è fornire link all'utente, ma recuperare *contesto grezzo* di alta qualità da iniettare nel prompt dell'LLM.

Il processo avviene in tre step:

- **Query Optimization:** L'LLM riformula la domanda dell'utente in una query di ricerca ottimizzata (es. da "chi è il presidente attuale?" a "current president of Italy 2024").
- **Content Extraction:** Il motore di ricerca visita le pagine più rilevanti in parallelo, rimuovendo pubblicità, menu e boilerplate, ed estraendo solo il testo informativo.
- **Context Aggregation:** I frammenti di testo estratti da diverse fonti vengono aggregati in un unico blocco di contesto che viene passato all'LLM per la generazione della risposta finale.

### 5.2 Strategia di Fallback

Il servizio è progettato per essere resiliente: 1. **\*\*Motore AI (Tavily):\*\*** Utilizzato come fonte primaria per la sua capacità di aggregare e pulire i dati da molteplici fonti web in tempo reale. 2. **\*\*Wikipedia API:\*\*** Se il servizio primario non è raggiungibile o configurato, il sistema effettua automaticamente un fallback sulle API pubbliche di Wikipedia per ottenere definizioni e concetti generali.

## 6 Esportazione e Interoperabilità

Per essere veramente utile, il materiale generato deve essere utilizzabile negli strumenti di studio quotidiani. Il servizio `export_service.py` gestisce questa necessità.

### 6.1 Generazione Pacchetti Anki (.apkg)

La funzionalità più avanzata è la creazione diretta di pacchetti per **Anki**, il software di ripetizione spaziata più diffuso. Il codice interagisce direttamente con il database SQLite interno di Anki:

```
1 cursor.execute('''
2     CREATE TABLE notes (
3         id INTEGER PRIMARY KEY,
4         guid TEXT NOT NULL,
5         mid INTEGER NOT NULL,
6         ...
7         flds TEXT NOT NULL, -- Contiene Front e Back separati
8         ...
9     )
10 ''')
```

Listing 3: Creazione struttura database Anki

Il sistema genera un file `.apkg` valido che include:

- La struttura del database (`collection.anki2`).
- I modelli di carte (CSS personalizzato per una visualizzazione pulita).
- I deck organizzati per materia.

Questo permette all'utente di importare centinaia di flashcard in Anki con un doppio click, mantenendo formattazione e tag.