Synapse

Neurosymbolic AI course

University of Calabria

A.A. 2025/2026

Frandina Ilaria 264232 - Siciliano Samuele 263633 - Spadafora Pierpaolo 263722

Overview

Synapse: Agentic Architecture for Flashcard Generation

Large Language Models present known limitations when used directly for educational applications: hallucinations, lack of grounding on specific materials, and variable output quality.

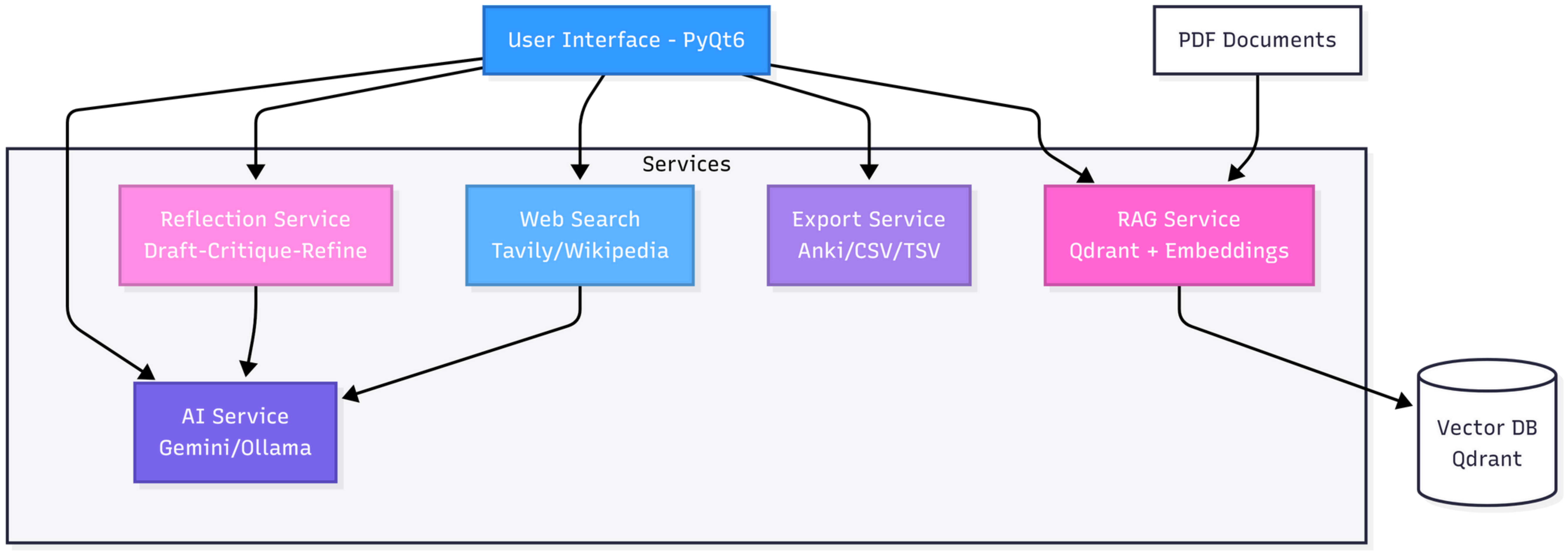
Synapse implements an architecture that combines **4 agentic design patterns** and **advanced functionalities**:

- **RAG** for anchoring to user documents;
- **Reflection** for iterative output improvement;
- **Few-Shot Learning** to guide response format;
- **Tool Use** for external source integration;
- **Export** to standard formats like Anki.

The system automatically generates high-quality flashcards based on study materials provided by the user, applying metacognition principles validated by the literature.

System Architecture

The system adopts a modular service-oriented architecture implemented in Python. The PyQt6 user interface orchestrates the interaction between five specialized services.
The AI Service manages the abstraction for communication with models (Google Gemini or local Ollama). The RAG Service implements a Qdrant vector database for document indexing and retrieval. The Reflection Service orchestrates the draft-critique-refine cycle. The Web Search Service integrates external sources via Tavily. The Export Service produces output in interoperable formats.



RAG Retrieval Augmented Generation

RAG **solves the problem of knowledge cutoff and lack of access** to specific documents by LLMs.
Documents provided by the user are processed, converted into vector embeddings, and indexed in Qdrant.
During inference, the query is transformed into an embedding and used to retrieve the most relevant chunks through **cosine similarity searc**h. These chunks are injected into the prompt context, allowing the LLM to generate responses grounded on the specific materials provided.
This approach ensures that the output is faithful to the user's study content, significantly reducing hallucinations and increasing response relevance to the specific domain.

Recursive chunking strategy

Retrieval quality critically depends on the document chunking strategy. A naive approach based on fixed length risks breaking semantic units, degrading the quality of the retrieved context.
Synapse implements **Recursive Character Text Splitting**: the algorithm attempts to preserve the semantic structure of the text using a hierarchy of separators. It first splits by paragraphs, then by lines, then by sentences, and only as a last resort cuts at the character level.
This approach maintains the semantic coherence of chunks, ensuring that each fragment contains a complete unit of meaning. The result is a measurable improvement in the quality of generated responses.

Reflection Iterative Output Improvement

The Reflection pattern implements a self-improvement cycle to increase the quality of generated flashcards. The process consists of three distinct phases:

- **Draft Generation**: the LLM generates a flashcard following a structured prompt based on Andy Matuschak's atomicity principles (single concept, precision, required cognitive effort).
- **Critique**: in a second call, the LLM assumes the role of reviewer and critically evaluates the produced flashcard against explicit criteria (focus, precision, contextual correctness, cognitive effort, conceptual depth).
- **Refinement**: if the critique identifies specific problems, the system invokes the LLM again providing the original flashcard, received critique, and context. The model produces an improved version that resolves the identified issues.

This approach transforms mediocre outputs into flashcards that meet high quality standards.

Initial Output (Draft):
- Question: **"What are design patterns?"**
- Answer: **"They are reusable solutions to common problems"**

Critical Analysis (Critique):
"The flashcard violates the cognitive effort principle: the answer is implicit in the question. Moreover, it remains at the level of superficial definition without exploring the 'why' or practical implications. Need to reformulate to require conceptual understanding."

Refined Output (Refine):
- Question: **"Why do design patterns improve software maintainability?"**
- Answer: **"They provide a shared vocabulary and validated architectures that make code more understandable for other developers and facilitate future modifications without introducing regressions"**

Few-Shot Learning

Guided Output Formatting

Few-Shot Learning addresses the challenge of obtaining consistently structured and formatted outputs from LLMs. By providing concrete examples in the prompt, the model learns the expected output format and style.

In Synapse, this pattern is applied to flashcard generation: **the prompt includes 2-3 example flashcards** that demonstrate desired characteristics, atomicity, appropriate cognitive effort level, clear question formulation, and comprehensive yet concise answers.

The LLM observes these examples and replicates the pattern when generating new flashcards, ensuring consistency across the entire deck. This approach significantly reduces variability in output quality and eliminates the need for extensive post-processing or format corrections.

Tool Use Web Search Integration

The Tool Use pattern addresses situations where the user's uploaded documents do not contain sufficient information to answer a query or generate appropriate flashcards. In these cases, the system performs web searches based on the user's question to retrieve complementary or updated information.

The implementation uses **Tavily** as the primary search service. When a user poses a question, the system formulates a search query and retrieves relevant content from the web. **The retrieved information is then processed and integrated into the LLM's context**, allowing it to generate responses that combine both the user's study materials and current external knowledge.

The system implements a fallback mechanism for resilience: if Tavily is unavailable or not configured, it automatically switches to **Wikipedia API** to provide at least basic definitions and general concepts. This ensures the system remains functional even without external service dependencies.

This dynamic integration maintains the same generation pipeline while expanding the knowledge base beyond static documents, enabling the system to handle a broader range of educational queries.

Export

Interoperability and Standard Formats

The system implements export functionality to standard formats to ensure integration with existing study tools.

The **Anki Package (.apkg) format** represents the most advanced export: the system directly generates an SQLite database with Anki's internal structure, including card model definitions, custom CSS for visualization, and organization in decks by subject. This allows direct import without the need for parsing or manual conversions.

Standard tabular formats (**CSV** and **TSV**) are also supported, ensuring maximum interoperability with spreadsheets, databases, and other flashcard management software. The separation of concerns through a dedicated service facilitates future extension toward additional formats.

Key System Features

Contextual grounding: RAG ensures that responses are anchored to the provided documents, reducing hallucinations and increasing domain-specific relevance.

Quality assurance: the Reflection pattern implements an automatic quality control mechanism that identifies and corrects flashcards not conforming to pedagogical design principles.

Dynamic updating: web search integration allows supplementing static knowledge with current information when necessary.

Privacy-first: support for fully local deployment via Ollama, eliminating dependencies on cloud services for users with data privacy requirements.

Integrated workflow: direct export to Anki eliminates manual conversion overhead, integrating seamlessly into the existing study pipeline.

Conclusions

Synapse demonstrates how the integration of agentic design patterns can address the known limitations of Large Language Models: hallucinations, lack of domain-specific grounding, and inconsistent output quality.

The architecture combines **RAG** for document-based retrieval, **Reflection** for iterative refinement, **Few-Shot** for output formatting, **Tool Use** for external knowledge integration, and **Export** for data interoperability. This combination produces a system that automatically generates structured content anchored to user-provided documents with automated quality control.

The modular service-oriented implementation ensures extensibility and maintainability while maintaining separation of concerns and testability.

Thank you

for your attention