Wie ich den Buch-Chatbot mit RAG entwickelt habe

Mein Vorgehen bei der Entwicklung

Ich habe schrittweise einen intelligenten Chatbot aufgebaut, der nicht nur Text verarbeitet, sondern echtes inhaltliches Verständnis für literarische Werke entwickelt.

Dabei bin ich wie folgt vorgegangen:

Systemdesign und Architektur

Ich entschied mich für eine dual angelegte Vektordatenbank, die sowohl feingranulare Textabschnitte als auch komplette Kapitel verwaltet. Diese Aufteilung ermöglicht es, je nach Fragestellung die optimale Informationsquelle auszuwählen. Besonderen Wert legte ich auf die intelligente Klassifizierung der Fragen, um unterschiedliche Antwortstrategien anwenden zu können.

Die vier Hauptentwicklungsschritte

1. **PDF-Verarbeitung** – die grundlegende, aber entscheidende Basis
2. **Kapitelerkennung** – zuverlässige Identifizierung römischer Zahlen und Kapitelmarkierungen
3. **Semantische Suche** – Implementierung der Vektoreinbettungen
4. **QA-System** – Aufbau der spezialisierten Antwortmechanismen

Datenaufbereitung und Textverarbeitung

Textbereinigung

Bei der Arbeit mit PDF-Dateien stellte ich fest, dass der extrahierte Text oft mit unerwünschten Elementen versehen war. Ich entwickelte daher mehrere Bereinigungsschritte:

- Entfernung überflüssiger Zeilenumbrüche und Leerzeichen
- Elimination von Archiv-Wasserzeichen
- Bereinigung von Seitenzahlen und Formatierungsartefakten

Chunking-Strategie

Nach einigen Experimenten fand ich die optimale Balance für die Textaufteilung:

- **Feine Chunks** mit 800 Zeichen Länge und 120 Zeichen Überlappung
- **Kapitel-basierte Segmente** für zusammenhängende Inhalte
- **Intelligente Trennung** an natürlichen Textgrenzen

Die Textaufteilung konfigurierte ich bewusst mit abgestuften Trennzeichen – von Absätzen über Sätze bis hin zu einzelnen Wörtern – um semantisch sinnvolle Einheiten zu erhalten.

Auswahl der Tools und Frameworks

LangChain als Rückgrat

Ich wählte LangChain wegen seiner umfassenden Funktionalität für RAG-Systeme. Besonders überzeugt hat mich die Möglichkeit, komplexe Verarbeitungsketten einfach zu konstruieren und die integrierten Prompt-Templates.

FAISS für effiziente Suche

Für die Vektorsuche entschied ich mich für FAISS, da es hohe Performance bei gleichzeitig einfacher Handhabung bietet. Die Möglichkeit, die Datenbanken lokal zu speichern und wieder zu laden, war ein wichtiges Kriterium.

HuggingFace Embeddings

Das Modell "all-MiniLM-L6-v2" erwies sich als idealer Kompromiss zwischen Geschwindigkeit und Qualität.

Ollama mit Llama3.2

Für die Sprachgenerierung setzte ich auf Ollama, weil es lokale Verarbeitung ohne Cloud-Abhängigkeiten ermöglicht. Die Temperatur konfigurierte ich bewusst niedrig (0.2), um faktentreue und konsistente Antworten zu gewährleisten.

Einsatz von KI-Tools im Entwicklungsprozess

Im Entwicklungsprozess setzte ich punktuell auf KI-Unterstützung:

- **Google Colab mit Gemini** als Entwicklungsumgebung für Echtzeit-Code-Vervollständigung und Fehleranalyse.
- **ChatGPT** nutzte ich als Sparringspartner für Architekturfragen, um verschiedene RAG-Ansätze zu diskutieren und Vor-/Nachteile abzuwägen.
- Besonders wertvoll war der Einsatz beim Prompt-Engineering, wo ich durch schnelles Testen verschiedener Strategien die Antwortqualität systematisch optimierte

Fazit

Die Entwicklung war eine produktive Zusammenarbeit zwischen meiner fachlichen Expertise und der effizienzsteigernden Unterstützung durch KI-Tools. Das Ergebnis ist ein System, das sowohl technisch robust als auch inhaltlich intelligent arbeitet