Smart Librarian

AI Book Recommender with RAG (ChromaDB) & Tool Calling

Local Implementation (Sentence-Transformers + GPT4All)

Mariana-Ionela Muntian

August 24, 2025

Abstract

This document presents **Smart Librarian**, a simple AI chatbot that recommends books based on user interests using a Retrieval-Augmented Generation (RAG) pipeline and a separate tool to fetch full summaries. The project runs **fully locally** (no OpenAI key required) with *Sentence-Transformers* for embeddings, *ChromaDB* as a vector store, and *GPT4All* as the local LLM. It can be easily switched to OpenAI GPT + embeddings if desired.

1 Overview

Goal. Build a chatbot that:

1. Stores 10+ book summaries locally.
2. Indexes summaries in a vector store (ChromaDB) for semantic search.
3. Answers user questions conversationally (LLM) and recommends *one* title.
4. Calls a `get_summary_by_title` tool to print the full summary.

Key Components.

- **Data**: `book_summaries.json` with ≥ 10 titles.
- **Embeddings**: `sentence-transformers/all-MiniLM-L6-v2`.
- **Vector Store**: ChromaDB (persistent on disk).
- **LLM**: GPT4All (e.g., `orca-mini-3b-gguf2-q4_0.gguf`).
- **Tool**: `local_get_summary_by_title(title)` uses the local JSON.

2 Requirements & Environment

Dependencies

Listing 1: requirements.txt (local version)

```
chromadb>=1.0.20
rich>=14.0.0
python-dotenv>=1.0.1
sentence-transformers>=3.0.1
gpt4all>=2.8.2
# optional TTS:
# pyttsx3>=2.90
```

Virtual Environment (Windows PowerShell)

```
py -m venv .venv
& ".\.venv\Scripts\Activate.ps1"
pip install -r requirements.txt
```

3 Data: Book Summaries

The dataset is a JSON array. Each entry has: `title`, `themes` (list), `short_summary`, `full_summary`. Example:

Listing 2: book_summaries.json (excerpt)

```
[
  {
    "title": "1984",
    "themes": ["dystopia", "social control", "freedom"],
    "short_summary": "A totalitarian regime monitors everyone. Winston seeks truth.",
    "full_summary": "Orwell's novel depicts a surveillance state where truth is
        manipulated..."
  },
  {
    "title": "The Hobbit",
    "themes": ["adventure", "friendship", "magic"],
    "short_summary": "Bilbo joins dwarves to reclaim treasure from Smaug.",
    "full_summary": "Bilbo Baggins, a comfort-loving hobbit, is recruited by Gandalf
        ..."
  }
  // ... 10+ total
]
```

4 Architecture & Flow

1. **Encode & Index** `short_summary + themes` with Sentence-Transformers.
2. **Store** vectors + docs in ChromaDB.
3. **Retrieve** top-k candidates for a user query (semantic search).
4. **LLM Decision** (GPT4All): pick *one* title and write a short conversational reply.
5. **Tool Call**: `local_get_summary_by_title(title)` prints the full summary.

5 Implementation (Core)

Main App (`app.py`)

Listing 3: Key parts of app.py (local RAG + tool)

```
import os, json, re
from dataclasses import dataclass
from typing import List, Any
from rich.console import Console
```

```
from rich.prompt import Prompt
from rich.panel import Panel

import chromadb
from sentence_transformers import SentenceTransformer
from gpt4all import GPT4All

console = Console()
DATA_PATH = os.path.join(os.path.dirname(__file__), "book_summaries.json")
DB_PATH = os.path.join(os.path.dirname(__file__), "chroma_db_local")
COLLECTION_NAME = "books_rag_local"
EMBED_MODEL_NAME = "sentence-transformers/all-MiniLM-L6-v2"
GPT4ALL_MODEL = "orca-mini-3b-gguf2-q4_0.gguf" # small & available

BAD_WORDS = {"idiot", "prost", "jignire", "ur", "urt", "hate", "fuck", "shit"}

@dataclass
class BookDoc:
    title: str
    short_summary: str
    full_summary: str
    themes: list

def load_books() -> List[BookDoc]:
    with open(DATA_PATH, "r", encoding="utf-8") as f:
        return [BookDoc(**b) for b in json.load(f)]

def get_or_create_collection(chroma_client) -> Any:
    try:
        return chroma_client.get_collection(COLLECTION_NAME)
    except Exception:
        return chroma_client.create_collection(COLLECTION_NAME)

def build_index(emb_model: SentenceTransformer, books: List[BookDoc]):
    chroma_client = chromadb.PersistentClient(path=DB_PATH)
    collection = get_or_create_collection(chroma_client)

    try:
        if collection.count() >= len(books): # already indexed
            return
    except Exception:
        pass

    docs, ids, metas = [], [], []
    for b in books:
        doc = f"Title: {b.title}\nThemes: {', '.join(b.themes)}\nShort: {b.
            short_summary}"
        docs.append(doc)
        ids.append(b.title)
        metas.append({"title": b.title, "themes": ", ".join(b.themes)}) # note: string

    vectors = emb_model.encode(docs, normalize_embeddings=True).tolist()
    collection.add(ids=ids, documents=docs, embeddings=vectors, metadatas=metas)

def query_similar(emb_model: SentenceTransformer, query: str, top_k: int = 3):
    chroma_client = chromadb.PersistentClient(path=DB_PATH)
    collection = get_or_create_collection(chroma_client)
    q_emb = emb_model.encode([query], normalize_embeddings=True).tolist()
```

```
    return collection.query(query_embeddings=q_emb, n_results=top_k)

def local_get_summary_by_title(books: List[BookDoc], title: str) -> str:
    for b in books:
        if b.title.strip().lower() == title.strip().lower():
            return b.full_summary
    return "Not found in local database."

def has_profanity(text: str) -> bool:
    t = text.lower()
    return any(bad in t for bad in BAD_WORDS)

def choose_with_llm(gpt: GPT4All, candidates: List[dict], user_q: str):
    cand_block = "\n".join([f"- {c['title']} :: {c['doc']}" for c in candidates])
    prompt = f"""
You are a helpful Romanian librarian bot. Pick EXACTLY ONE title that best fits the
    user's question.
Reply STRICTLY in JSON with keys: "title" and "reply". "reply" must be a short
    Romanian answer (~120 words).

Question: {user_q}

Candidates:
{cand_block}

Example:
{{"title": "1984", "reply": "Recomand '1984' pentru c ..."}}
"""
    out = gpt.generate(prompt, temp=0.2, max_tokens=300)
    m = re.search(r"\{.*\}", out, flags=re.S)
    if m:
        try:
            data = json.loads(m.group(0))
            if isinstance(data, dict) and "title" in data and "reply" in data:
                return data["title"], data["reply"]
        except Exception:
            pass
    # Fallback: top-1 candidate
    title = candidates[0]["title"] if candidates else ""
    reply = f"i recomand {title} (cel mai relevant dup cutarea semantic)."
    return title, reply

def main():
    console.print(Panel.fit("[bold green]Smart Librarian  LOCAL (RAG + GPT4All)[/bold
        green]"))
    books = load_books()
    emb_model = SentenceTransformer(EMBED_MODEL_NAME)
    build_index(emb_model, books)

    # Load local LLM (downloads model on first run)
    gpt = GPT4All(GPT4ALL_MODEL)

    console.print("ntrebare (ex: [i]Vreau o carte despre prietenie i magie[/i]) "
                  "sau [bold]exit[/bold] pentru a iei.\n")

    while True:
        user_q = Prompt.ask("[bold cyan]Tu[/bold cyan]").strip()
        if user_q.lower() in {"exit", "quit", "q"}:
```

```
            console.print("La revedere! "); break
        if has_profanity(user_q):
            console.print("[yellow]Te rog reformuleaz politicos.[/yellow]"); continue

        results = query_similar(emb_model, user_q, top_k=3)
        candidates = []
        for i in range(len(results.get("ids", [[]])[0])):
            candidates.append({
                "title": results["ids"][0][i],
                "doc": results["documents"][0][i],
                "meta": results["metadatas"][0][i]
            })
        if not candidates:
            console.print("[red]N-am gsit candidai.[/red]"); continue

        chosen_title, short_reply = choose_with_llm(gpt, candidates, user_q)
        full = local_get_summary_by_title(books, chosen_title)
        final = f"{short_reply}\n\n[Rezumat complet pentru {chosen_title}]\n{full}"
        console.print(Panel.fit(final))

if __name__ == "__main__":
    main()
```

CLI Usage

```
python app.py
# Then type inside the app (not in the PS prompt):
Vreau o carte despre prietenie i magie
```

6 Tool: `get_summary_by_title`

Implemented locally as `local_get_summary_by_title`. In a cloud/OpenAI version, you would register it as a function/tool via the Chat API and pass the selected title back to the model.

7 Design Notes & Rationale

- **Local-first**: avoids API quota or billing; demonstrates the RAG + tool pattern end-to-end.
- **ChromaDB**: simple persistent store; fast kNN search; metadata kept scalar (lists are joined as strings).
- **SBERT (MiniLM)**: small, fast sentence embeddings sufficient for short summaries & themes.
- **LLM choice**: GPT4All model can be swapped (e.g., Mistral 7B Instruct) for higher quality if the machine allows.

8 Switching to OpenAI (Optional)

If you later add credit, you can swap:

- Embeddings: `text-embedding-3-small` (OpenAI).

- Chat: `gpt-4o-mini` (or similar).

The RAG pipeline (retrieve → pick title → call tool) remains identical.

9 Troubleshooting (Windows)

- **PowerShell activation**: use `& ".venv\Scripts\Activate.ps1"`.
- **pip WinError 32**: close editors/antivirus, then `python -m pip install --upgrade pip` and retry.
- **Chroma metadata error**: pass only scalars; join lists (e.g., themes) into a comma-separated string.
- **GPT4All model 404**: pick an available model (e.g., `orca-mini-3b-gguf2-q4_0.gguf`); clear cache at `%LOCALAPPDATA%\nomic.ai\GPT4All`.
- **HuggingFace symlink warning**: benign; enable Developer Mode or ignore.

10 Evaluation Tips

- Create queries aligned with themes (*friendship*, *magic*, *war*, *freedom*).
- Check whether the chosen title matches the intent; improve by adding more books or a stronger LLM.
- Keep the tool output concise but informative; the full summary follows the short recommendation.

11 Conclusion

Smart Librarian demonstrates a complete, local RAG pipeline with tool calling: semantic retrieval with ChromaDB, a local LLM to select a single title, and a tool to fetch the full summary. It is easy to extend with TTS, a web UI, or a cloud LLM.

Repository name suggestion: `smart-librarian`.
Short repo description: "AI-powered book recommender using RAG (ChromaDB) and summaries tool."