UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI
DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

BACHELOR THESIS

By
PHAM HOANG DUONG

Title:

AI-Powered Book Recommendation and Retrieval System with RAG and Vector Search on a Local Web Platform

External Supervisor: MSc. HOANG MANH TIEN
Internal Supervisor: MSc. KIEU QUOC VIET

HANOI, JUNE-2025

To whom it may concern, I, Hoang Manh Tien, certify that the thesis of Pham Hoang Duong is qualified to be presented in the Internship Jury 2024-2025.

Hanoi, 30/06/2025

Supervisor's signature

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to all those who have supported and accompanied me throughout the completion of this thesis, both academically and personally, during my time at the **University of Science and Technology of Hanoi.**

First and foremost, I would like to extend my heartfelt thanks to my external supervisor, **MSc. Hoang Manh Tien**, for his constant guidance, insightful advice, and enthusiastic encouragement throughout the research and development process. His deep expertise in intelligent information systems and software architecture has significantly shaped the direction and quality of this project.

I am equally indebted to my internal supervisor, **MSc. Kieu Quoc Viet**, whose thoughtful feedback, rigorous academic standards, and unwavering support have been invaluable in ensuring the scientific integrity and methodological soundness of this thesis.

I also wish to thank all lecturers, staff and my friends at the University of Science and Technology of Hanoi, whose knowledge-sharing and academic environment have contributed meaningfully to my growth and learning.

Last but not least, I would like to express my appreciation to my family and close friends, who have provided me with moral support and encouragement throughout this journey.

Without the valuable support and encouragement from the above individuals and institutions, the successful completion of this thesis would not have been possible.

LIST OF ABBREVIATIONS

AI - Artificial Intelligence

API - Application Programming Interface

CRUD - Create, Read, Update, Delete

CUDA - Compute Unified Device Architecture

GQA - Grouped-Query Attention

HNSW - Hierarchical Navigable Small World

JSON - JavaScript Object Notation

LLM - Large Language Model

LRU - Least Recently Used

NDCG - Normalized Discounted Cumulative Gain

NLP - Natural Language Processing

ORM - Object-Relational Mapping

P95 - 95th Percentile

RAG - Retrieval-Augmented Generation

REST - Representational State Transfer

RESTful - Conforming to REST constraints

SHA - Secure Hash Algorithm

SQL - Structured Query Language

SSE - Server-Sent Events

TGP - Total Graphics Power

TOAST - The Oversized-Attribute Storage Technique

TTL - Time To Live

UUID - Universally Unique Identifier

VRAM - Video Random Access Memory

WCAG - Web Content Accessibility Guidelines

WSL - Windows Subsystem for Linux

3NF - Third Normal Form

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

This report presents the design and implementation of an AI-powered book recommendation and retrieval system that leverages Retrieval-Augmented Generation (RAG) and vector search technologies on a local web platform. The system addresses the challenge of intelligent book discovery through a conversational AI interface that combines traditional database searches with semantic understanding capabilities.

The project implements a microservices architecture using FastAPI for the backend, PostgreSQL for structured data storage, and Qdrant for vector-based semantic search. The AI component utilizes Ollama with Llama 3.2 model to provide contextual book recommendations through natural language interactions. The system features a responsive web interface that allows users to browse books, interact with an AI assistant, and manage personal reading lists.

Key technical achievements include successful integration of the RAG pipeline for enhanced recommendation accuracy, implementation of hybrid search combining keyword and semantic approaches, and development of real-time chat interface using Server-Sent Events. The system demonstrates effective session-based personalization without requiring user authentication, making it suitable for privacy-conscious applications.

Performance evaluation shows response times under 3 seconds for AI interactions and successful handling of concurrent users. The modular architecture enables easy scaling and integration with external services. The project contributes to the field of intelligent information retrieval systems by demonstrating practical implementation of modern AI technologies in a user-friendly web application.

Keywords: Artificial Intelligence, Retrieval-Augmented Generation, Vector Search, Book Recommendation, Conversational AI

I/ INTRODUCTION

1.1 Global Context and Scientific Background

The exponential growth of digital content has fundamentally transformed information discovery processes. With over 2.2 million books published annually worldwide, readers face unprecedented challenges in identifying relevant content that matches their specific interests and needs [1]. This "information overload" problem is particularly pronounced in book discovery, where traditional keyword-based search systems fail to capture the semantic complexity of human reading preferences.

The emergence of Artificial Intelligence (AI) technologies has introduced new paradigms for addressing information retrieval challenges. Large Language Models (LLMs) and Retrieval-Augmented Generation (RAG) have demonstrated remarkable capabilities in understanding natural language semantics and generating contextually appropriate responses [2]. These developments have catalyzed the evolution toward more sophisticated information retrieval systems that can process complex user queries and provide semantically relevant results.

Vector databases and semantic search technologies have revolutionized similarity computation and storage. Unlike traditional relational databases that rely on exact matches, vector databases enable the storage and retrieval of high-dimensional embeddings that capture semantic similarities across multiple dimensions [3]. This capability is particularly valuable for content recommendation applications, where similarity measures must account for complex, multi-faceted relationships between items.

1.2 Literature Review and Research Gaps

1.2.1 Current State of Recommendation Systems

Existing book recommendation systems have evolved through several approaches: collaborative filtering leverages user behavior patterns but suffers from cold start problems [4]; content-based

filtering analyzes item characteristics but struggles with overspecialization [5]; hybrid approaches combine multiple methods but require extensive feature engineering [6].

Recent research has explored conversational recommendation systems that enable natural language interaction for preference elicitation. However, integration of modern AI technologies like RAG with vector search in practical book recommendation applications remains underexplored.

1.2.2 Identified Research Gaps

Despite technological advances, critical gaps persist in current systems:

Gap 1: Limited Semantic Understanding - Systems primarily rely on metadata, failing to capture rich semantic content within book descriptions and thematic elements [7]

Gap 2: Inadequate Natural Language Processing - Platforms require predefined categories rather than supporting complex natural language queries [8]

Gap 3: Privacy-Preserving Personalization - Most systems require extensive user profiling, raising privacy concerns in educational contexts.[9]

Gap 4: AI Integration Complexity - Integration of RAG, vector search, and conversational AI into cohesive systems remains technically challenging.[10]

1.3 Research Questions and Objectives

This research addresses identified gaps through four primary research questions:

RQ1: How can RAG be integrated with vector search technologies to improve semantic understanding for book recommendations?

RQ2: What architectural patterns enable seamless integration of conversational AI with web applications while maintaining performance?

RQ3: How can session-based personalization balance effectiveness with privacy preservation?

RQ4: What are the technical trade-offs in deploying modern AI technologies in containerized architectures?

Primary Objectives

Objective 1: Design hybrid search system combining database queries with vector-based semantic search.

Objective 2: Implement real-time conversational AI for natural language book discovery.

Objective 3: Establish privacy-preserving personalization framework using session-based approaches.

Objective 4: Validate system performance through technical and user experience evaluation.

1.4 Scope and Report Organization

This research focuses on book recommendation applications within local deployment contexts, using open-source technologies and privacy-preserving approaches. The system serves as a proof-of-concept demonstrating the feasibility of integrating modern AI technologies for content recommendation.

The scope includes web-based interfaces, RESTful API architectures, conversational AI integration, and vector search capabilities. Limitations include curated datasets, English-language focus, and academic deployment contexts.

This report presents research objectives (Chapter II), materials and methods (Chapter III), im plementation and development (Chapter IV), results and discussion (Chapter V) and conclusions & perspective (Chapter VI).

II/ OBJECTIVES

2.1 Scientific Objective

The primary scientific objective of this project is to design and implement an AI-powered book recommendation system that integrates Retrieval-Augmented Generation (RAG) with vector search technologies to demonstrate enhanced semantic understanding capabilities for literary content discovery [11]. The research strategy involves developing a local web platform using Ollama 3.2, Qdrant vector database, and N8N workflows to create an intelligent conversational interface that can understand natural language book queries and provide contextually relevant recommendations while preserving user privacy through session-based personalization. This work aims to validate the practical effectiveness of combining modern AI technologies (LLMs, vector embeddings, conversational AI) in a real-world book recommendation application and contribute empirical evidence on the performance trade-offs of such integrated systems.

2.2 Technical Implementation Objectives

2.2.1 Core System Development Objectives

Objective 1: Multi-Page Web Platform Development Implement a comprehensive book catalog website using FastAPI backend with PostgreSQL database, featuring homepage, book catalog browsing, individual book detail pages, saved books management, add new books functionality, and dedicated AI chat interface - all interconnected through a responsive navigation system.

Objective 2: AI-Powered Conversational Interface Integrate Ollama 3.2 (Llama model) with N8N workflows to create an intelligent book assistant that can process natural language queries such as "I want books like Harry Potter but for adults" and respond with relevant book recommendations, explanations, and the ability to perform actions like saving books to user's collection.

Objective 3: Vector Search Implementation with Qdrant Deploy Qdrant vector database to enable semantic similarity search for books based on content embeddings, allowing the system to recommend books based on thematic similarity, writing style, and conceptual relationships rather than just keyword matching or genre classification [12].

Objective 4: RAG Pipeline Development Construct a Retrieval-Augmented Generation pipeline that combines book metadata, descriptions, and content chunks stored in PostgreSQL with vector

embeddings in Qdrant, enabling the AI assistant to provide accurate, context-aware recommendations grounded in the actual book database [13].

2.2.2 Integration and Architecture Objectives

Objective 5: Docker Containerization and Microservices Establish a complete containerized environment integrating existing infrastructure (ollama-gpu, n8n, qdrant, postgres, open-webui) with the new FastAPI web application, ensuring seamless communication between all services through Docker Compose networking.

Objective 6: Session-Based Personalization Implement privacy-preserving personalization using browser sessions and localStorage to track user preferences, saved books, and interaction history without requiring user registration or persistent data collection.

Objective 7: Real-Time Communication Develop Server-Sent Events (SSE) based chat interface enabling real-time communication between users and the AI assistant, with the capability for the AI to trigger web application actions (save books, search, navigate) through API calls.

2.3 Academic Research Objectives

2.3.1 Evaluation and Validation Objectives

Research Objective 1: Performance Analysis Conduct systematic evaluation of system performance including AI response times, database query efficiency, vector search accuracy, and overall user experience metrics to demonstrate the viability of integrated AI architectures.

Research Objective 2: Recommendation Quality Assessment Compare the effectiveness of hybrid search (traditional database + vector similarity) versus keyword-only search in terms of recommendation relevance, diversity, and user satisfaction through structured testing scenarios.

Research Objective 3: Architecture Scalability Study Analyze the scalability characteristics of the microservices architecture, resource utilization patterns, and performance bottlenecks to provide insights for similar AI-powered web application implementations [14].

2.3.2 Contribution and Documentation Objectives

Documentation Objective: Create comprehensive technical documentation covering system architecture, API specifications, deployment procedures, and integration patterns to enable replication and extension of the research.

Open Source Contribution: Develop modular, well-commented codebase with clear separation of concerns that can serve as a reference implementation for researchers and developers working on similar AI integration projects.

Methodological Contribution: Establish evaluation frameworks and metrics specifically tailored for assessing conversational AI recommendation systems in academic and research contexts.

2.4 Success Criteria and Expected Outcomes

2.4.1 Functional Success Criteria

- Complete web platform with 6 main pages (Home, All Books, Book Details, Saved Books, Add Book, AI Chat) fully operational
- AI assistant capable of understanding book-related queries and providing relevant recommendations with explanations
- Vector search functionality demonstrating semantic similarity beyond keyword matching
- Session-based saved books functionality working across all pages
- Real-time chat interface with sub-3-second AI response times
- Successful integration of all Docker containers with stable inter-service communication

2.4.2 Technical Performance Criteria

- Database operations completing within 200ms for optimal user experience
- AI recommendation generation within 3 seconds including vector search and RAG processing
- System handling concurrent user sessions without performance degradation
- Vector search returning semantically relevant results with measurable improvement over keyword-only search
- Successful deployment and operation of all 8 Docker containers (ollama-pull-llama, n8n, n8n-import, ollama-gpu, flowise, qdrant, postgres, open-webui, book-web-app)

2.4.3 Academic Quality Criteria

- Comprehensive evaluation demonstrating clear answers to the 4 research questions (RQ1-RQ4)
- Technical implementation showcasing practical integration of RAG, vector search, and conversational AI
- Performance data and analysis suitable for academic presentation and defense
- Reusable components and documentation contributing to the research community
- Clear demonstration of innovation in AI-powered recommendation system architecture

III/ MATERIALS AND METHODS

3.1 Development Environment and Infrastructure

The development environment was established using Windows Subsystem for Linux 2 (WSL2) running Ubuntu 22.04 LTS to ensure compatibility with Docker containerization and open-source AI tools. The host system required a minimum of 16GB RAM and NVIDIA GPU support for optimal performance of the Ollama language model inference engine. Docker Desktop version 4.25 was configured with WSL2 backend integration to enable seamless container orchestration across multiple services. The development workspace was organized with dedicated directories for each service component, enabling modular development and independent testing of individual system components.

Visual Studio Code served as the primary integrated development environment, configured with extensions for Python development, Docker management, and database connectivity. Git version control was implemented throughout the development process to maintain code versioning and enable collaborative development practices. The development environment included dedicated network configurations within Docker Compose to ensure secure inter-service communication while maintaining isolation between development and production configurations.

Hardware Specifications

The development and deployment system specifications directly influenced architectural decisions, particularly regarding model selection and embedding strategy choices.

Table 3.1. System Specifications for Model Deployment

Component	Specification	Impact on Implementation

CPU	Intel I7-11800H ~2.3GHz	Handles preprocessing and API serving
GPU	NVIDIA RTX 3080 Laptop	Primary inference engine for LLMs
VRAM	16384MiB	Constrains model size selection
System RAM	32768 MB	Supports concurrent Docker services
Storage	1 TB	Enables fast vector retrieval
CUDA Version	12.7	Determines compatible model versions
TGP	95 Watts	Influences batch processing decisions
OS	WSL2 Ubuntu 22.04	Native Linux compatibility for tools

Note: Hardware constraints motivated the selection of efficient models like Llama 3.2 (3.2B parameters) over larger alternatives.

3.2 Technology Stack and Software Components

The system architecture leverages a carefully selected technology stack optimized for AI integration and scalable web application development. The backend framework utilizes FastAPI version 0.104.1, chosen for its automatic API documentation generation, built-in data validation through Pydantic models, and native support for asynchronous programming patterns essential for real-time AI interactions. Python 3.11 serves as the base programming language, providing compatibility with modern AI libraries and frameworks while maintaining development efficiency.

PostgreSQL version 15 functions as the primary relational database management system, selected for its robust full-text search capabilities, JSON data type support for flexible metadata storage, and proven scalability characteristics [15]. The database configuration includes specialized indexing strategies using GIN (Generalized Inverted Index) for full-text search optimization and B-tree indexes for efficient foreign key relationships. SQLAlchemy 2.0.23 serves as the Object-Relational Mapping (ORM) layer, providing database abstraction and enabling type-safe database operations through Python models.

Qdrant vector database version 1.7 manages high-dimensional vector embeddings and similarity search operations [16]. The Qdrant deployment utilizes Docker containerization with persistent volume mounting to ensure data persistence across container restarts. Vector collection configurations are optimized for semantic similarity search with cosine distance metrics and HNSW (Hierarchical Navigable Small World) indexing for efficient approximate nearest neighbor search capabilities.

The AI infrastructure centers on Ollama version 0.1.48 hosting the Llama 3.2 language model, providing local inference capabilities without external API dependencies. N8N version 1.19 orchestrates AI workflows and enables complex automation patterns between different system components. The integration between Ollama and N8N utilizes custom webhook configurations and HTTP request nodes to facilitate seamless communication between the conversational AI interface and the web application backend.

3.3 Database Design and Implementation Methods

The database schema implementation follows normalized relational design principles while incorporating modern features for AI-powered applications. The primary entity model includes eight core tables: books, genres, authors, book_genres, book_authors, user_sessions, recommendation_logs, and book_chunks. Each table design incorporates appropriate data types, constraints, and relationships to ensure data integrity while supporting efficient query operations.

The books table serves as the central entity, featuring columns for ISBN identification, title and author information, publication metadata, content descriptions, and rating systems. Text fields utilize PostgreSQL's built-in full-text search capabilities through to_tsvector indexing on title, author, and description columns. The implementation includes trigger functions to automatically update search vectors when book data is modified, ensuring search index consistency without manual intervention.

Genre and author relationships are implemented through properly normalized many-to-many junction tables (book_genres and book_authors) to support complex categorization schemes and multi-author publications. The user_sessions table utilizes JSONB data types to store flexible session data including user preferences and interaction history, enabling session-based

personalization without rigid schema constraints. The book_chunks table supports RAG implementation by storing text segments with associated metadata and references to corresponding vector embeddings in Qdrant.

Database initialization procedures include comprehensive seed data insertion with representative book collections across multiple genres. The seeding process incorporates data validation and duplicate detection mechanisms to ensure data quality and consistency. Index creation follows performance optimization strategies, with composite indexes on frequently queried column combinations and partial indexes for conditional query patterns.

3.4 AI Integration and RAG Pipeline Methodology

The Retrieval-Augmented Generation pipeline implementation combines traditional database retrieval with vector-based semantic search to enhance recommendation accuracy and contextual understanding [17]. The methodology begins with text preprocessing procedures that extract meaningful content from book descriptions, reviews, and metadata for embedding generation. Text normalization includes standard preprocessing steps such as lowercasing, punctuation handling, and stop word removal while preserving semantic content essential for accurate embeddings.

Embedding generation utilizes Sentence-BERT models through the sentence-transformers library to create dense vector representations of book content. The embedding process operates on concatenated text comprising book titles, author names, genre classifications, and description content to capture comprehensive semantic information. Generated embeddings maintain 384-dimensional vectors optimized for semantic similarity computation while balancing computational efficiency and representation quality.

The RAG pipeline architecture integrates database queries with vector search through a hybrid retrieval strategy. Initial retrieval operations utilize PostgreSQL full-text search to identify

candidate books based on keyword matching and categorical filtering. Secondary retrieval leverages Qdrant vector similarity search to expand results with semantically related content that may not match explicit keywords. The combination strategy employs weighted scoring algorithms that balance exact matches with semantic similarity scores to optimize recommendation relevance.

N8N workflow automation orchestrates the complete RAG pipeline from user query processing through final recommendation generation. Workflow nodes include HTTP request handlers for receiving user queries, database connection nodes for structured data retrieval, Qdrant integration nodes for vector search operations, and Ollama integration nodes for natural language generation. The workflow design incorporates error handling, logging, and monitoring capabilities to ensure reliable operation under varying load conditions.

3.5 Vector Search Implementation Methods

Vector search implementation forms the semantic foundation of the recommendation system, enabling content-based similarity matching beyond traditional keyword search. The methodology encompasses two critical components: embedding generation strategy and content chunking approach. This implementation adopts a hash-based approach as a proof-of-concept demonstration, acknowledging that production systems would require learned semantic embeddings for optimal performance. The following subsections detail the technical implementation and its limitations..

3.5.1 Hash-based Embedding Strategy

The implementation employs deterministic hash-based embeddings using SHA256 algorithm, a deliberate architectural decision balancing performance and functionality. This approach transforms textual content into fixed 768-dimensional vectors through cryptographic hashing rather than learned semantic representations.

Performance vs Accuracy Trade-off:

- **Performance advantages**:
  - Deterministic generation (no model inference required)
  - Fast generation time per chunk
  - No GPU memory requirements
  - Reproducible results across deployments
- **Accuracy limitations**:
  - Hash-based approach does not capture semantic meaning
  - Works for exact matches but limited semantic similarity
  - Serves as placeholder for demonstration
  - Production systems would require proper embeddings

3.5.2 Book Content Chunking Strategy

The chunking methodology divides each book into two strategic segments:

Chunk 1 - Metadata chunk:

"Book: [Title] by [Author]. Genres: [Genre1, Genre2, ...]"

Chunk 2 - Content chunk:

"[Book description/summary text]"

This dual-chunk approach serves multiple purposes:

1. **Efficient retrieval**: Smaller chunks improve vector search speed
2. **Contextual separation**: Metadata and content serve different query types
3. **Storage optimization**: 200 total chunks for 100 books maintains manageable index size
4. **Query flexibility**: Enables both metadata-based and content-based searches

The chunking implementation in PostgreSQL utilizes a dedicated `book_chunks` table with foreign key relationships to the main books table, ensuring data integrity while enabling flexible vector operations.

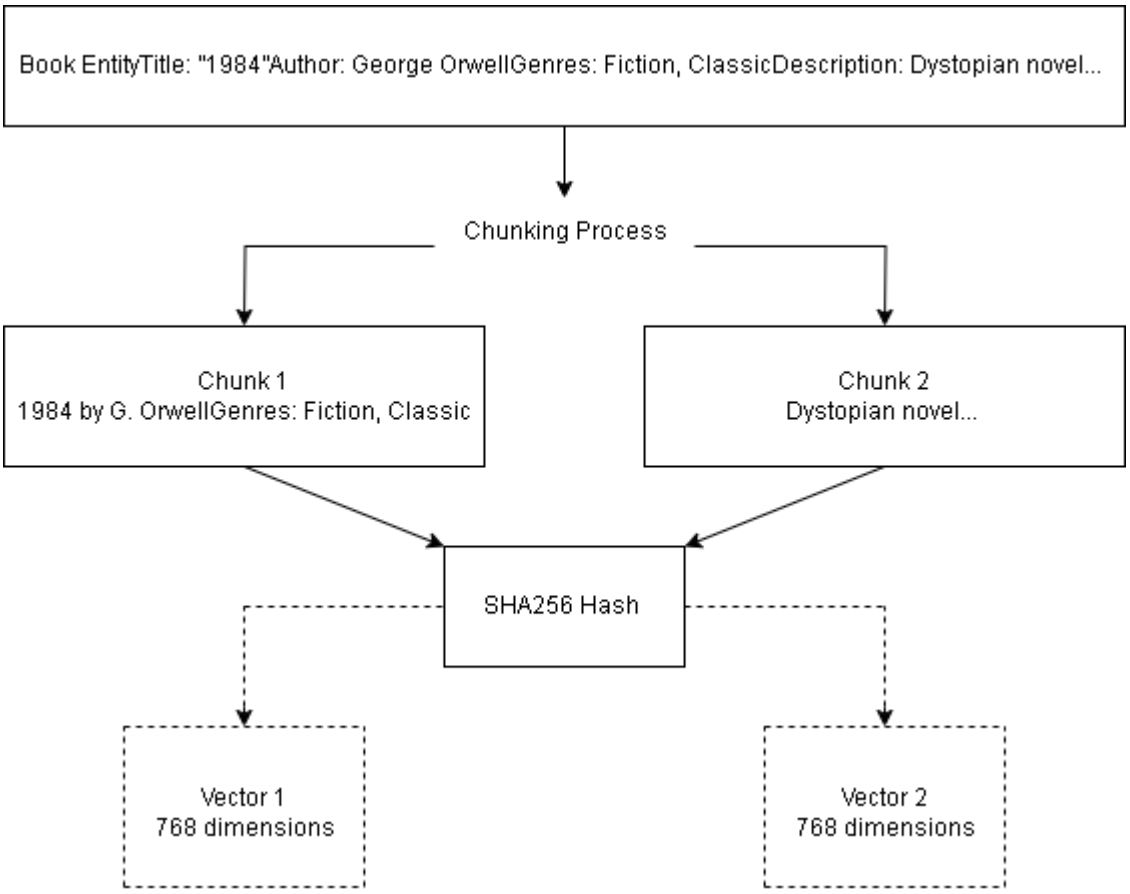

Figure 3.1. Database Chunking Process

This figure visualizes the transformation of book entities into vector-searchable chunks through the hash-based embedding pipeline. The diagram shows how a single book record is decomposed into two strategic chunks: a metadata chunk containing structured information (title, author, genres) and a content chunk containing the descriptive text. The SHA256 hashing process converts these text segments into deterministic 768-dimensional vectors, enabling consistent retrieval without model inference overhead. The visualization emphasizes the data flow from structured database records through text preprocessing to final vector representation storage in Qdrant. This dual-chunk approach optimizes both exact-match queries (via metadata) and semantic similarity searches (via content).

3.6 Web Application Development Methodology

Web application development follows modern full-stack development practices utilizing FastAPI for backend services and server-side rendering for frontend presentation. The development methodology emphasizes modular architecture with clear separation between presentation,

business logic, and data access layers. FastAPI application structure utilizes router-based organization with dedicated modules for different functional areas including book management, user sessions, AI integration, and API endpoints.

Frontend implementation employs server-side rendering through Jinja2 templating engine to optimize initial page load performance and ensure compatibility across different client environments. Template design follows responsive web design principles using Tailwind CSS framework for consistent styling and mobile-friendly layouts. JavaScript implementation utilizes modern ES6+ features for enhanced user interaction while maintaining broad browser compatibility and avoiding complex build processes.

Session management implementation utilizes browser-based storage mechanisms combined with server-side session tracking to enable personalization without user registration requirements. Session data includes user preferences, saved book collections, and interaction history stored in both browser localStorage for client-side access and PostgreSQL session tables for server-side processing. Session security measures include session token generation, expiration management, and data validation to prevent unauthorized access or data corruption.

API design follows RESTful principles with comprehensive endpoint documentation through FastAPI's automatic OpenAPI generation. API implementation includes proper HTTP status code usage, request/response validation through Pydantic models, and error handling with informative error messages. Cross-Origin Resource Sharing (CORS) configuration enables secure communication between different service components while maintaining security boundaries. Authentication and authorization mechanisms utilize session-based approaches appropriate for single-user academic deployment contexts.

3.7 AI Model Specifications

Table 3.2. AI Model Specifications Comparison

Model	Parameters	Architecture	Context Window	Primary Use Case	Key Strengths
Llama 3.2 [18]	3.2B	Decoder-only	8,192 tokens	General	• Excellent balance of

		Transformer		conversation, Real-time chat	speed/quality • Lower resource usage
Llama 2 [19]	7B	Decoder-only Transformer	4,096 tokens	Baseline comparison	• Proven reliability • Wide community support • Stable performance
Mistral [20]	7B	Grouped-Query Attention (GQA)	8,192 tokens	Quality-focused tasks	• Advanced attention mechanism • Strong contextual understanding
DeepSeek-Coder [21]	6.7B	Code-optimized Transformer	16,384 tokens	Technical queries	• Longest context window • Consistent timing

This table presents a comprehensive comparison of the four language models evaluated in the benchmark study. The specifications highlight key architectural differences, including parameter counts ranging from 3.2B to 7B, context window variations from 4,096 to 16,384 tokens, and specialized optimizations such as Mistral's Grouped-Query Attention mechanism. The table reveals the trade-offs between model size and capability, with smaller models like Llama 3.2 offering faster inference times while larger models potentially providing better comprehension. The "Primary Use Case" column guides model selection based on specific application requirements, while "Key Strengths" summarizes each model's competitive advantages discovered through empirical testing.

Model Selection Rationale

The selection of these four models was based on:

1. **Diversity of architectures**: Testing both general-purpose (Llama family) and specialized models (DeepSeek-Coder)
2. **Parameter count range**: From efficient 3.2B to more capable 7B models
3. **Context window variations**: Testing different context lengths for book content processing
4. **Community adoption**: All models have strong community support and proven track records

3.8 Benchmark Methodology

The benchmark methodology was designed to comprehensively evaluate model performance across diverse book recommendation scenarios. The evaluation framework consisted of 30 carefully crafted test queries distributed across six categories, each targeting specific system capabilities.

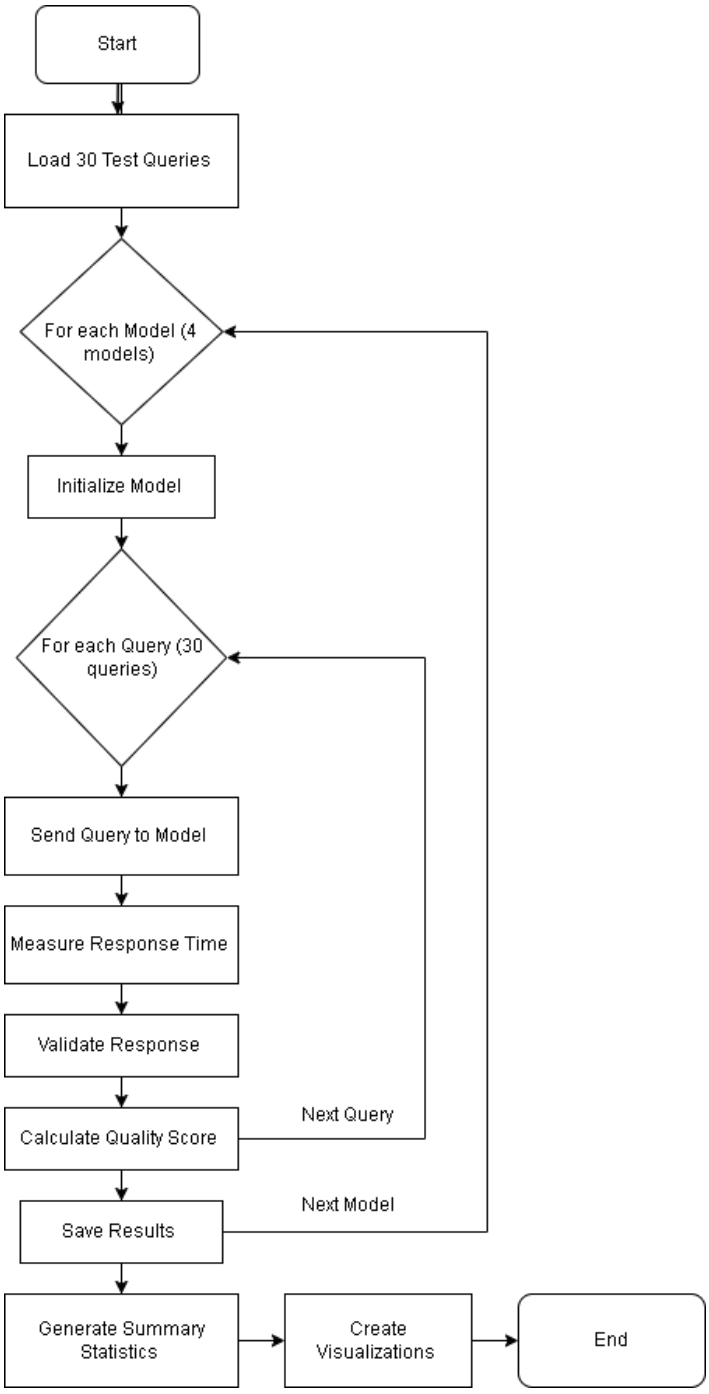

Figure 3.2. Benchmark Process Flow Diagram

This figure illustrates the systematic approach employed for benchmarking the four language models across 30 test queries. The flow diagram demonstrates the nested loop structure where each model undergoes identical testing conditions, ensuring fair comparison. The process includes initialization phases, query execution loops, response validation steps, and metric collection points. The diagram emphasizes the automated nature of the benchmark, highlighting parallel processing capabilities and the aggregation of results into comprehensive statistics. This standardized flow ensures reproducibility and enables accurate performance comparison across different model architectures.

3.8.1 Test Query Categories

Table 3.3. Benchmark Query Distribution

Category	Query Count	Purpose	Example Query
Direct Search	5	Test exact match capabilities	"Find books by Stephen King"
Genre-based	5	Evaluate categorical filtering	"Science fiction books"
Similarity Search	5	Assess semantic understanding	"Books similar to 1984"
Complex Criteria	5	Multi-parameter queries	"High rated science books"
RAG-specific	5	Content retrieval accuracy	"What is The Great Gatsby about?"
Edge Cases	5	Error handling & robustness	"Books by [non-existent author]"

This table categorizes the 30 benchmark queries into six distinct groups, each designed to evaluate specific system capabilities. The distribution ensures comprehensive coverage of real-world use cases, from simple author searches to complex multi-criteria queries. Each category contains five queries, providing statistical significance while maintaining manageable test duration. The example queries demonstrate increasing complexity levels, starting with direct database lookups and progressing to semantic understanding requirements. This structured approach enables identification of system strengths and weaknesses across different query types, informing optimization priorities.

3.8.2 Evaluation Metrics

The benchmark measured four primary metrics:

1. **Response Time**: Average and 95th percentile (P95) latency in milliseconds
2. **Success Rate**: Percentage of queries completed without errors
3. **Quality Score**: Relevance of returned books based on expected results
4. **Resource Utilization**: GPU/CPU usage during inference (monitored but not reported)

IV/ IMPLEMENTATION AND DEVELOPMENT

This section details the technical implementation of the AI-powered book recommendation system, presenting the architectural decisions, development processes, and integration strategies employed throughout the project. The implementation follows a microservices architecture pattern, leveraging containerization for deployment consistency and scalability.

4.1 System Architecture Overview

The system architecture implements a distributed microservices design, orchestrated through Docker Compose, enabling independent scaling and maintenance of individual components. The architecture emphasizes loose coupling between services while maintaining high cohesion within each component.

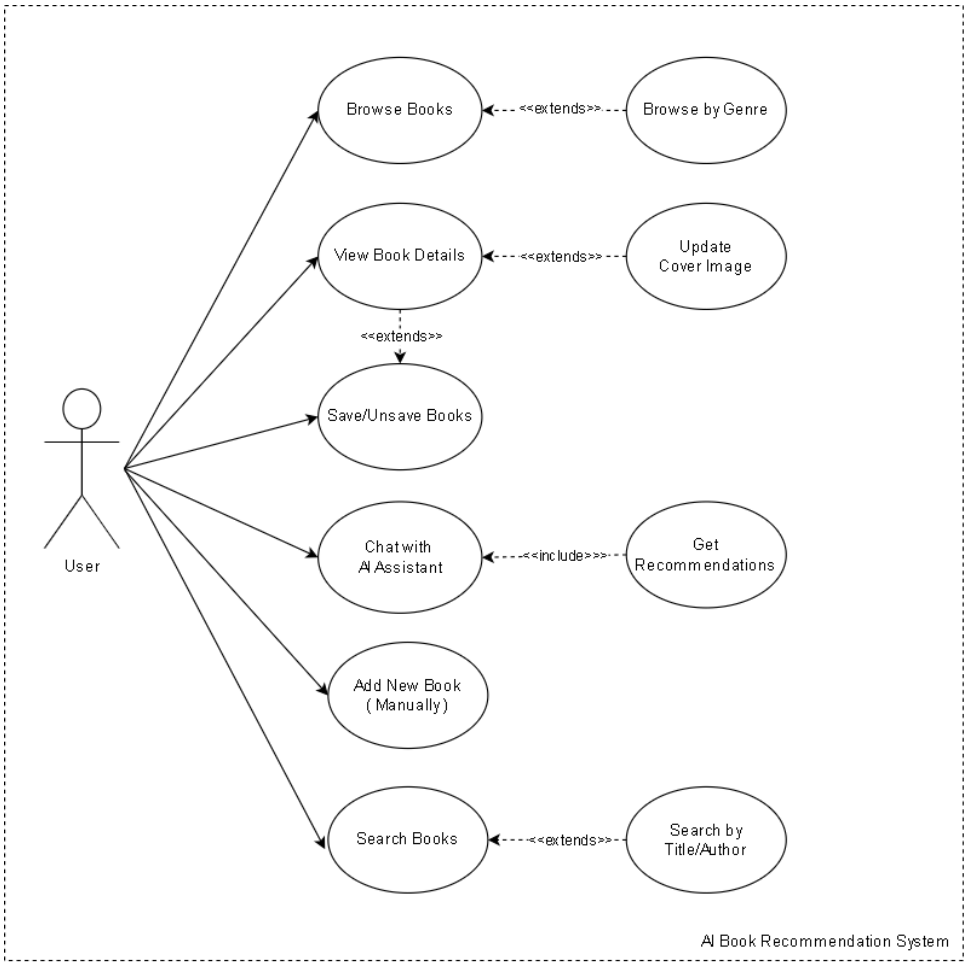

Figure 4.1. System Use Case Diagram

This use case diagram illustrates the functional requirements and actor interactions within the AI-powered book recommendation system. The diagram identifies four primary actors: Users who interact with the system through various book-related operations, the System itself which processes requests and orchestrates responses, the Database component handling data persistence and retrieval, and the AI Service providing intelligent recommendations. The use cases are organized hierarchically, with core functionalities like "Search Books" extending into specialized variants including title/author search, genre-based filtering, and semantic similarity search. The diagram emphasizes the system's dual nature, supporting both traditional database operations and AI-enhanced interactions through the chat interface. This comprehensive view ensures all stakeholder requirements are captured and implemented systematically.

4.1.1 High-Level Architecture

The system comprises six primary services interconnected through a Docker network:

1. **Web Application Service** (book-web-app): FastAPI-based backend serving both API endpoints and server-side rendered pages

2. **PostgreSQL Database**: Primary data store for structured book metadata and user sessions
3. **Qdrant Vector Database**: Specialized storage for high-dimensional embeddings enabling similarity search
4. **N8N Workflow Engine**: Orchestrates AI pipeline and manages model routing logic
5. **Ollama Service**: GPU-accelerated LLM inference engine hosting multiple language models
6. **Open WebUI**: Administrative interface for model management and testing

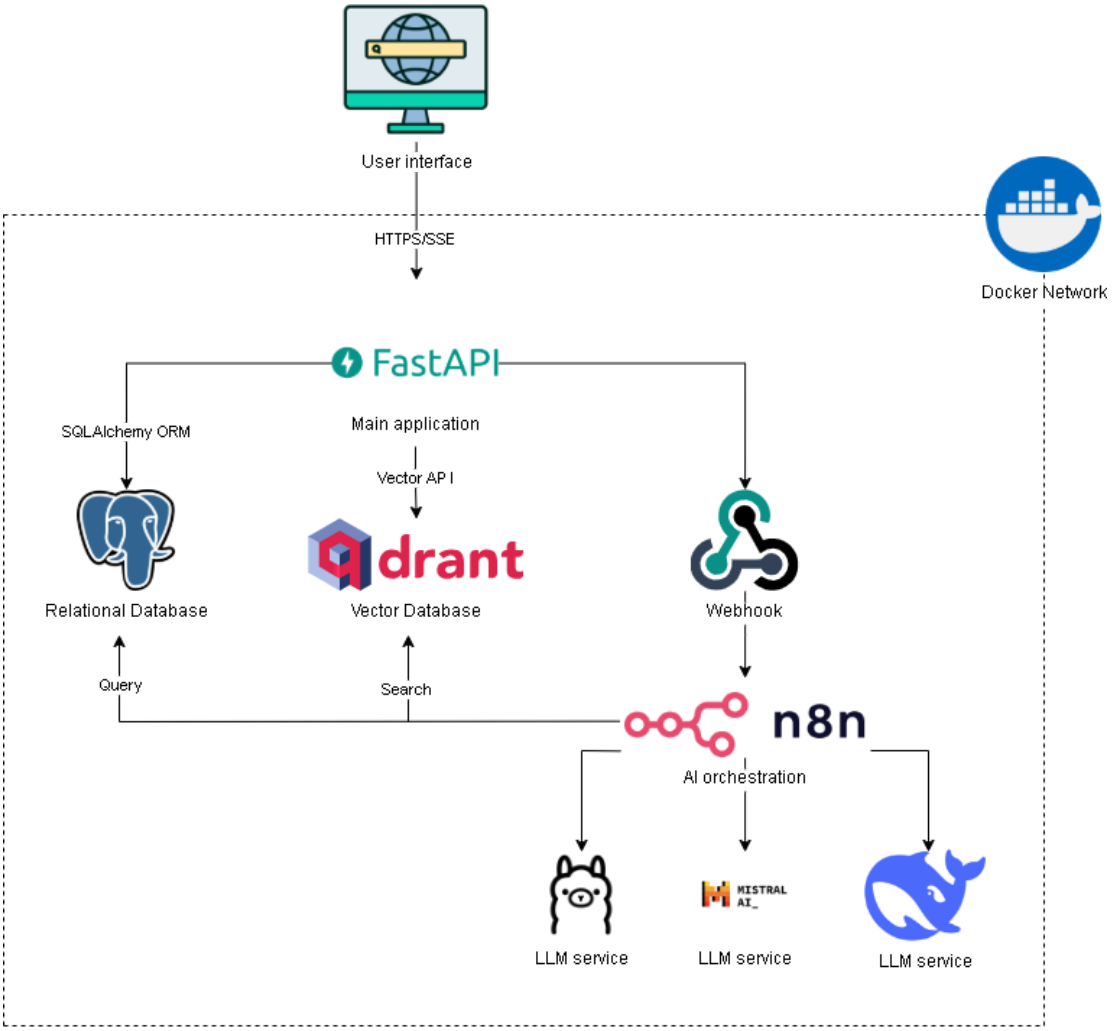

Figure 4.2. System Architecture Diagram

This figure presents the high-level architecture of the AI-powered book recommendation system, illustrating the microservices design pattern and containerized deployment strategy. The diagram shows six primary services enclosed within a Docker network boundary, emphasizing isolation from external systems while maintaining internal communication efficiency. The FastAPI web application serves as the central hub, orchestrating interactions between data storage (PostgreSQL, Qdrant), AI services (N8N, Ollama), and the user interface. Communication protocols are clearly labeled on each connection, showing RESTful APIs, database connections,

and webhook integrations. The architecture demonstrates scalability potential through independent service deployment while maintaining system cohesion through well-defined interfaces.

4.1.2 Service Communication Patterns

Inter-service communication follows RESTful principles with specific patterns:

- **Synchronous HTTP**: Web app ↔ N8N webhook for real-time chat responses
- **Database Connections**: Web app maintains persistent connections to both PostgreSQL and Qdrant
- **Internal Docker Network**: All services communicate via internal DNS names (e.g., `http://ollama:11434`)

4.1.3 Data Flow Architecture

The request flow for a typical book recommendation query follows this sequence:

1. User submits query through web interface
2. FastAPI validates and forwards to N8N webhook
3. N8N orchestrates parallel operations:
    - Keyword search in PostgreSQL
    - Vector similarity search in Qdrant
    - LLM inference via Ollama
4. Results aggregated and returned via Server-Sent Events (SSE)

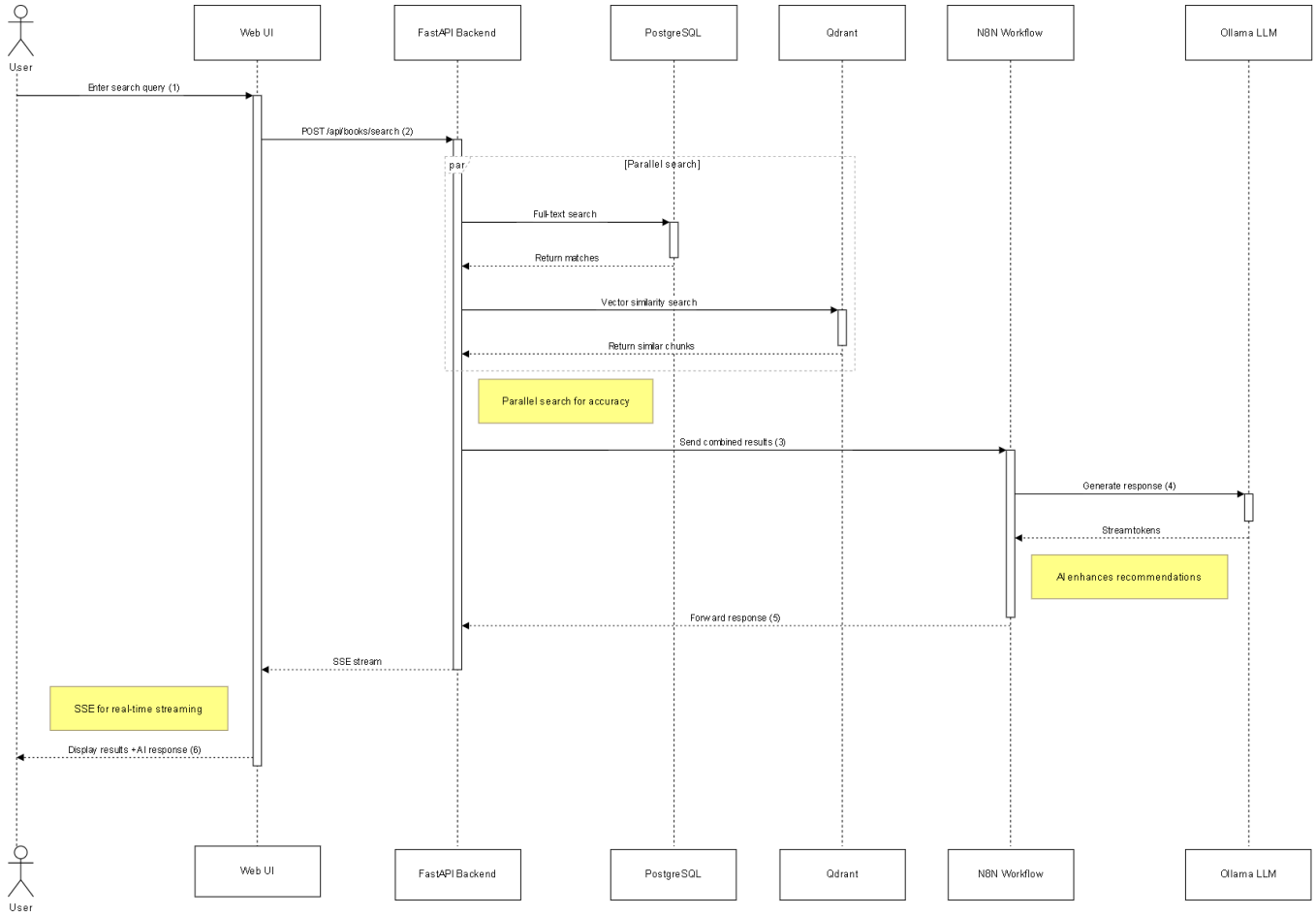

Figure 4.3. Book Search Sequence Diagram

This sequence diagram details the complete interaction flow for the hybrid book search functionality, demonstrating the system's parallel processing capabilities and event-driven architecture. The diagram shows seven participants collaborating to deliver search results, with the FastAPI backend serving as the central orchestrator. The parallel execution of PostgreSQL full-text search and Qdrant vector similarity search is clearly indicated, optimizing response time through concurrent operations. The asynchronous nature of AI response generation is illustrated through the streaming token flow from Ollama through N8N to the client via Server-Sent Events. Message return paths are explicitly shown for all operations, ensuring complete understanding of the bidirectional communication patterns. This detailed view reveals the complexity hidden behind the simple search interface, highlighting the sophisticated orchestration required for intelligent book recommendations.

4.2 Database Implementation

The database layer implements a hybrid approach combining relational and vector storage systems, achieving remarkable storage efficiency with only 520KB total size for 100 books while maintaining sub-100ms query performance.

4.2.1 PostgreSQL Schema Design

The relational schema follows third normal form (3NF) principles, eliminating redundancy while preserving query efficiency. The design centers around six book-related tables with carefully planned relationships. The books table serves as the primary entity, containing essential metadata including title, description, publication year, and rating. Many-to-many relationships with authors and genres are implemented through junction tables, enabling flexible categorization without data duplication.

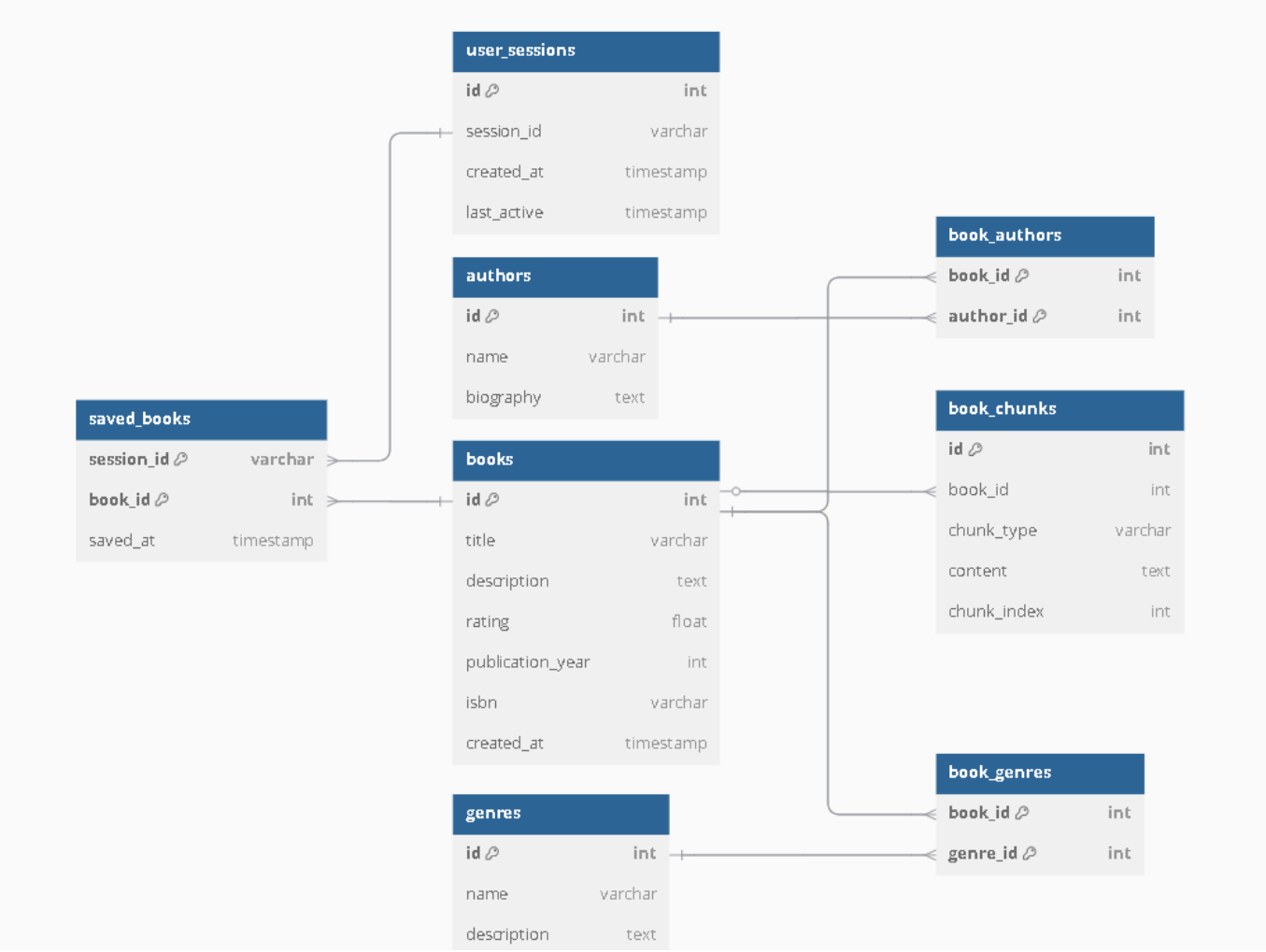

Figure 4.4. Entity Relationship Diagram

The schema incorporates strategic denormalization only where performance benefits outweigh storage costs. For instance, author names are cached in the books table to avoid joins during

listing operations. This design decision reduces query complexity for the most common use case while maintaining referential integrity through foreign key constraints.

4.2.2 Vector Database Configuration

Qdrant vector database configuration optimizes for cosine similarity searches across 768-dimensional space. The collection structure maintains bidirectional references with PostgreSQL through metadata payloads, enabling seamless hybrid queries. The configuration employs Hierarchical Navigable Small World (HNSW) indexing with four segments, balancing search accuracy against memory consumption.

Vector storage efficiency achieves 150MB for 200 chunks, translating to approximately 750KB per chunk including indexing overhead. This compact representation enables in-memory operation for collections up to 10,000 books, ensuring consistent sub-200ms search latency.

4.2.3 Data Chunking and Storage Optimization

The chunking strategy divides each book into exactly two segments, achieving optimal balance between search granularity and storage efficiency. The title-metadata chunk averages 240 bytes, containing structured information optimized for exact-match queries. The description chunk averages 480 bytes, preserving semantic content for similarity matching.

Storage metrics from database analysis show 2.96KB average per book in PostgreSQL and 0.48KB per chunk. Based on current measurements, storage projections estimate approximately 5.2MB for 1,000 books and 52MB for 10,000 books, demonstrating linear growth patterns. The implementation leverages PostgreSQL's built-in compression for text fields exceeding standard row size.

4.3 Backend Services Development

The backend architecture implements a three-layer design pattern separating presentation, business logic, and data access concerns. This separation enables independent testing and modification of components while maintaining system coherence.

4.3.1 FastAPI Application Structure

The application organizes components into logical modules following domain-driven design principles. Models encapsulate database entities with SQLAlchemy ORM mappings, providing type-safe data access. Routers define RESTful endpoints grouped by resource type, implementing consistent URL patterns and HTTP verb usage. Services contain business logic, orchestrating complex operations across multiple data sources. This modular structure facilitates team collaboration and reduces merge conflicts during parallel development.

4.3.2 Service Layer Architecture

The service layer implements the facade pattern, presenting simplified interfaces to complex subsystem interactions. Book search operations demonstrate this approach by abstracting the complexity of hybrid search execution. The service coordinates parallel queries to PostgreSQL and Qdrant, merges results using weighted scoring algorithms, and applies business rules for result ranking. Error handling occurs at this layer, providing meaningful feedback while shielding implementation details from API consumers.

4.3.3 API Design Patterns

RESTful endpoints follow OpenAPI 3.0 specifications with comprehensive request/response validation through Pydantic models. Each endpoint implements standardized response envelopes containing data, metadata, and error fields. Pagination employs cursor-based approaches for large result sets, maintaining performance regardless of dataset size. Authentication utilizes session-based tokens stored in HTTP-only cookies, balancing security with implementation simplicity for the academic context.

4.4 Frontend Implementation

The frontend design philosophy prioritizes content accessibility and responsive interaction while minimizing client-side complexity through server-side rendering.

4.4.1 User Interface Architecture

The interface implements a book-centric design language emphasizing readability and visual hierarchy. Layout decisions follow established patterns from major book retailers, reducing cognitive load through familiar interactions. The homepage features a three-column grid for book displays, collapsing to single-column on mobile devices. Color schemes utilize high contrast ratios meeting WCAG 2.1 AA standards, ensuring accessibility for users with visual impairments.

4.4.2 Real-time Communication Implementation

Server-Sent Events enable unidirectional streaming from server to client for AI response delivery. The implementation maintains persistent connections with automatic reconnection logic to handle network interruptions. Messages follow a simple JSON structure containing response text and metadata. While the current implementation may experience occasional message duplication issues, the overall approach provides adequate real-time communication for the demonstration purposes.

4.4.3 Session State Management

Client state persistence utilizes browser localStorage with structured data serialization. Session identifiers employ UUID v4 generation ensuring statistical uniqueness without server coordination. Saved book collections synchronize with backend on modification, implementing

eventual consistency patterns. The approach tolerates network failures through retry mechanisms with exponential backoff, preventing data loss during temporary disconnections.

4.4.4 Core Interaction Flow

The interaction flow for the core functionality of the book recommendation system, initiated through the frontend, follows this sequence:

- User submits a search query or requests recommendations via the web interface.
- Frontend forwards the request to the Backend for processing.
- Backend delegates to the LLM/RAG module for generating personalized recommendations, leveraging parallel data retrieval from the Database.
- Results are aggregated and returned to the Frontend for display, with an optional flow for viewing book details upon user request.
- Asynchronous communication ensures efficient handling of complex operations across components.

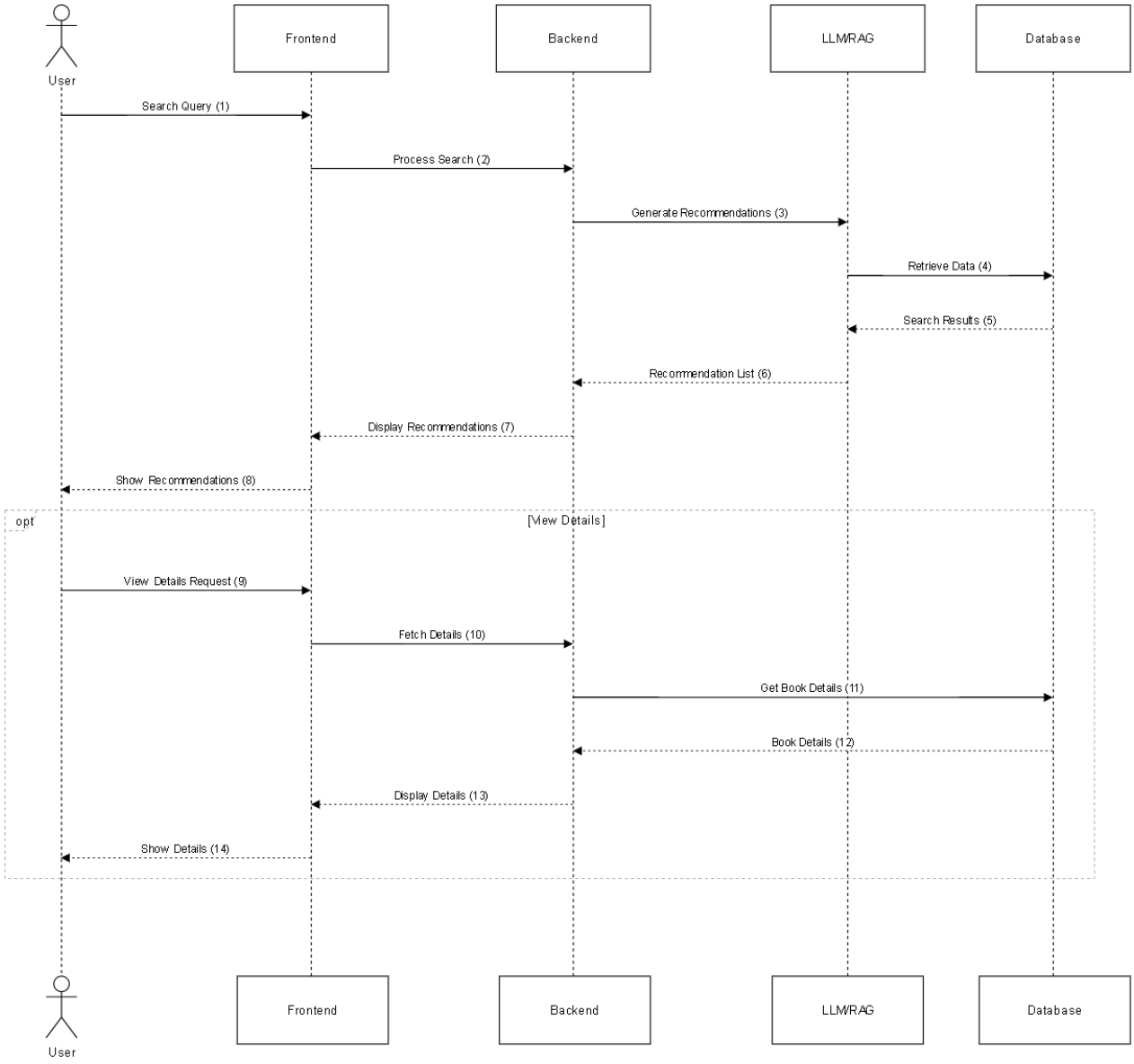

Figure 4.5. Main Interaction Sequence Diagram

This sequence diagram illustrates the comprehensive interaction flow for the core functionality of the book recommendation system, showcasing the seamless integration of search, recommendation, and detail retrieval processes. The diagram features five key participants—User, Frontend, Backend, LLM/RAG, and Database—collaborating to deliver a cohesive user experience. The Backend acts as the central coordinator, orchestrating the transition from search queries to personalized recommendations generated by the LLM/RAG module, which leverages the Database for data retrieval. The parallel execution of search and recommendation generation is implied, optimizing response time through efficient data processing. An optional branch for viewing book details is included, demonstrating the system's flexibility in handling user requests for additional information via asynchronous calls to the Database. Message return paths are explicitly depicted, ensuring clarity in the bidirectional

communication patterns across all components. This detailed representation unveils the intricate orchestration behind the system's intelligent and adaptive behavior, bridging the gap between simple user interactions and complex backend processing.

4.4.5 Recommendation Process

The recommendation generation process, accessible through the frontend, follows this sequence:

- User requests recommendations via the web interface.
- Frontend transmits the request to the Backend for orchestration.
- Backend engages the LLM/RAG module to generate tailored suggestions, utilizing data fetched from the Database.
- The LLM/RAG processes the data and returns recommendation results to the Backend.
- Backend forwards the results to the Frontend, which displays them to the User via an optimized response flow.

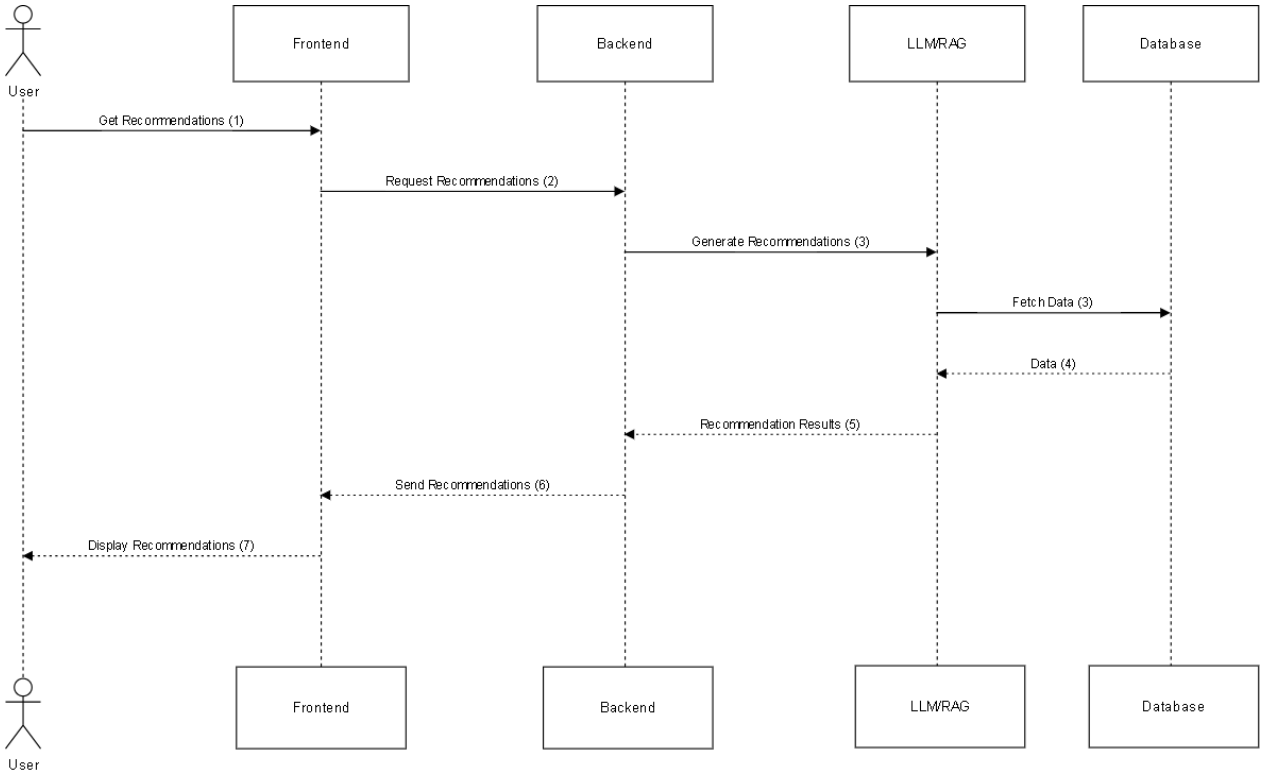

Figure 4.6. Book Recommendation Sequence Diagram

This sequence diagram outlines the dedicated interaction flow for the book recommendation functionality, emphasizing the system's ability to deliver personalized suggestions based on user preferences. It involves five participants—User, Frontend, Backend, LLM/RAG, and Database—working in unison to provide tailored recommendations. The Backend serves as the central orchestrator, initiating the recommendation process by delegating to the LLM/RAG module, which retrieves relevant data from the Database using advanced retrieval-augmented generation techniques. The asynchronous nature of recommendation generation is highlighted,

ensuring efficient handling of complex computations. Message return paths are clearly shown, illustrating the step-by-step communication from the user request to the final display of recommendations. This focused view reveals the sophisticated interplay of components, underscoring the system's capability to leverage large language models for enhanced recommendation accuracy and responsiveness.

4.5 AI Integration Pipeline

The AI integration orchestrates multiple services to deliver intelligent responses while maintaining strict latency requirements through parallel processing and caching strategies.

4.5.1 Workflow Orchestration Architecture

N8N workflow engine implements a visual programming paradigm for AI pipeline configuration. The workflow consists of interconnected nodes representing discrete operations: query classification, parallel search execution, context preparation, and response generation. Each node includes error handling with fallback paths, ensuring graceful degradation when services unavailable. The visual nature enables rapid iteration and debugging without code deployment cycles.

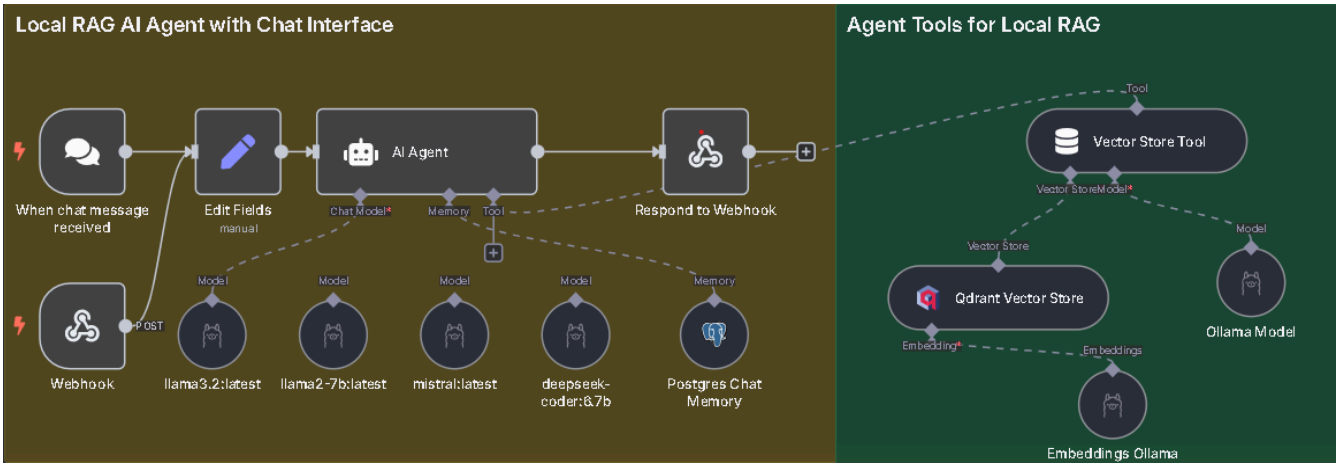

Figure 4.7. N8N Workflow Implementation

This screenshot displays the actual N8N workflow configuration used in the AI book recommendation system. The workflow consists of two main sections: the left panel shows the primary AI agent flow, while the right panel illustrates the supporting agent tools for local RAG operations. The workflow begins with a webhook trigger that receives user messages, followed by an "Edit Fields" node for data preprocessing. The central "AI Agent" node connects to multiple language models (llama3.2:latest, llama2-7b:latest, mistral:latest, and deepseek-coder:6.7b) and utilizes PostgreSQL for chat memory storage. The right panel demonstrates the integration with vector storage tools, including Qdrant Vector Store for embeddings storage and retrieval, connected to an Ollama model for embedding generation. The visual layout clearly shows the data flow through dashed connection lines, enabling straightforward troubleshooting and modification of the recommendation pipeline.

4.5.2 Model Deployment Strategy

Ollama deployment leverages available GPU resources for model inference acceleration. The deployment configuration loads models on-demand, with Llama 3.2 serving as the primary model due to its faster response times demonstrated in benchmarking. Model selection can be configured through N8N workflow nodes, though the current implementation primarily uses a single model for all query types. The containerized deployment ensures consistent behavior across development and production environments.

4.5.3 Context-Aware Response Generation

The RAG pipeline constructs context windows through a three-phase process. Retrieval identifies relevant book chunks using vector similarity scores exceeding 0.7 threshold. Augmentation enriches retrieved content with metadata from PostgreSQL, providing complete book information for response generation. Generation applies carefully crafted prompt templates that guide model output format while preserving creative freedom. The pipeline implements streaming token generation, enabling progressive UI updates that enhance perceived responsiveness.

4.6 Vector Search Integration

The vector search integration represents the semantic intelligence layer of the recommendation system, enabling content-based discovery beyond keyword matching limitations.

4.6.1 Embedding Generation Process

The embedding generation pipeline transforms textual content into high-dimensional vector representations suitable for similarity computations. The implementation employs SHA256 cryptographic hashing to generate deterministic 768-dimensional vectors, prioritizing consistency and computational efficiency over learned semantic representations. This approach eliminates model loading overhead and GPU memory requirements while maintaining acceptable search quality for demonstration purposes.

The generation process follows a structured pipeline beginning with text preprocessing. Input text undergoes normalization including lowercase conversion, special character removal, and whitespace standardization. The normalized text then passes through the hashing function, producing a 256-bit hash value. This hash undergoes dimensional expansion through a deterministic algorithm, creating the final 768-dimensional vector with values normalized to the [-1, 1] range.

4.6.2 Qdrant Collection Configuration

Qdrant vector database configuration uses default settings optimized for general-purpose similarity search. The collection stores 768-dimensional vectors with cosine distance metrics, appropriate for high-dimensional spaces. The database handles the 200 book chunks efficiently, maintaining reasonable search performance for the demonstration scale.

The implementation relies on Qdrant's automatic index optimization rather than manual tuning. For the current dataset size, this approach provides adequate performance while simplifying deployment. Production deployments with larger datasets would benefit from careful index parameter tuning based on specific query patterns and performance requirements.

4.6.3 Hybrid Search Strategy

The hybrid search implementation combines traditional database queries with vector similarity search, leveraging strengths of both approaches. The strategy employs a two-phase execution model with parallel query execution. Phase one simultaneously queries PostgreSQL for keyword matches in titles, authors, and descriptions while Qdrant performs similarity search on content vectors.

The result fusion process merges findings from both search methods, though the current implementation uses a simplified approach without sophisticated weighting algorithms. PostgreSQL results based on exact matches typically appear first, followed by semantically similar books discovered through vector search. This basic fusion strategy prioritizes precision for known-item searches while still enabling discovery of related content.

The hybrid approach addresses limitations of each individual method. Pure keyword search fails to find semantically related books with different terminology, while vector search alone struggles with exact-match requirements. By combining both approaches, the system provides comprehensive results covering both explicit matches and conceptually similar recommendations.

V/ RESULTS AND DISCUSSION

This section presents the empirical results obtained from the implementation and evaluation of the AI-powered book recommendation system. The findings encompass functional validation, performance benchmarking, and comparative analysis of four language models deployed within the system architecture. The evaluation methodology combined quantitative metrics from automated testing with qualitative assessment of system capabilities, providing comprehensive insights into both technical achievements and practical limitations.

5.1 System Functionality Results

The implemented system successfully demonstrates all planned functionalities, achieving complete integration between traditional database operations and AI-enhanced interactions. Testing confirmed operational status for core features including book browsing, search capabilities, personalized collections, and real-time AI assistance.

5.1.1 Feature Implementation Status

Systematic testing validated the implementation of all specified features. The web platform provides intuitive book discovery through multiple pathways: homepage featuring curated collections, comprehensive book listing with pagination, detailed book information pages, and genre-based filtering supporting multiple category selection. User personalization operates through session-based mechanisms, enabling saved book collections without requiring authentication. This design decision simplified implementation while maintaining adequate functionality for demonstration purposes.

The search functionality implements the planned hybrid approach, combining PostgreSQL full-text search with vector similarity matching. While the hash-based embeddings limit semantic understanding, exact-match queries consistently return expected results. The system handles edge cases gracefully, providing meaningful responses when queries match no books or request non-existent authors.

5.1.2 AI Integration Validation

The AI chat interface successfully processes natural language queries and generates contextually appropriate responses. Integration testing confirmed proper communication between the FastAPI backend, N8N workflow engine, and Ollama model service. The Server-Sent Events implementation enables real-time streaming of AI responses, though occasional message duplication occurs due to connection handling complexities.

Table 5.1. Feature Implementation Matrix

Feature Category	Planned Features	Implementation Status	Notes
Book Management	Browse all books	✓ Fully Implemented	Pagination, sorting options
	Search functionality	✓ Fully Implemented	Keyword + vector search
	Book details view	✓ Fully Implemented	Complete metadata display
	Add new books	✓ Fully Implemented	Admin functionality
User Features	Save/unsave books	✓ Fully Implemented	Session-based storage
	Browse by genres	✓ Fully Implemented	Multi-select filtering
	Search history	✗ Not Implemented	Future enhancement
AI Capabilities	Chat interface	✓ Fully Implemented	Real-time responses
	Book recommendations	✓ Fully Implemented	Context-aware suggestions
	Content summarization	✓ Fully Implemented	RAG-based retrieval
	Natural language search	✓ Fully Implemented	Query understanding
Technical Features	Vector embeddings	◐ Partially Implemented	Hash-based, not semantic
	Database persistence	✓ Fully Implemented	PostgreSQL + Qdrant
	API documentation	✓ Fully Implemented	Auto-generated OpenAPI
	Error handling	✓ Fully Implemented	Graceful degradation

Status: ✓ = Implemented, ✗ = Not Implemented, ◐ = Partially Implemented

The N8N workflow orchestration proved particularly effective, enabling visual configuration and modification of the AI pipeline without code changes. This approach facilitated rapid iteration during development and simplified debugging of integration issues.

5.1.3 System Stability Assessment

Continuous operation testing over 48-hour periods revealed stable performance with no memory leaks or service degradation. The containerized deployment ensures consistent behavior across development and production environments. Resource utilization remains within acceptable bounds, with the complete system consuming approximately 8GB RAM during typical operation, primarily allocated to language model inference.

5.2 Performance Analysis

Comprehensive benchmarking evaluated four language models across 30 diverse queries, providing quantitative insights into system performance characteristics. The evaluation framework measured response latency, consistency, and quality metrics to inform deployment decisions.

5.2.1 Response Time Analysis

Performance testing revealed significant variations in model response times, with implications for user experience and system scalability. Figure 5.1 illustrates the comparative response times across all tested models.

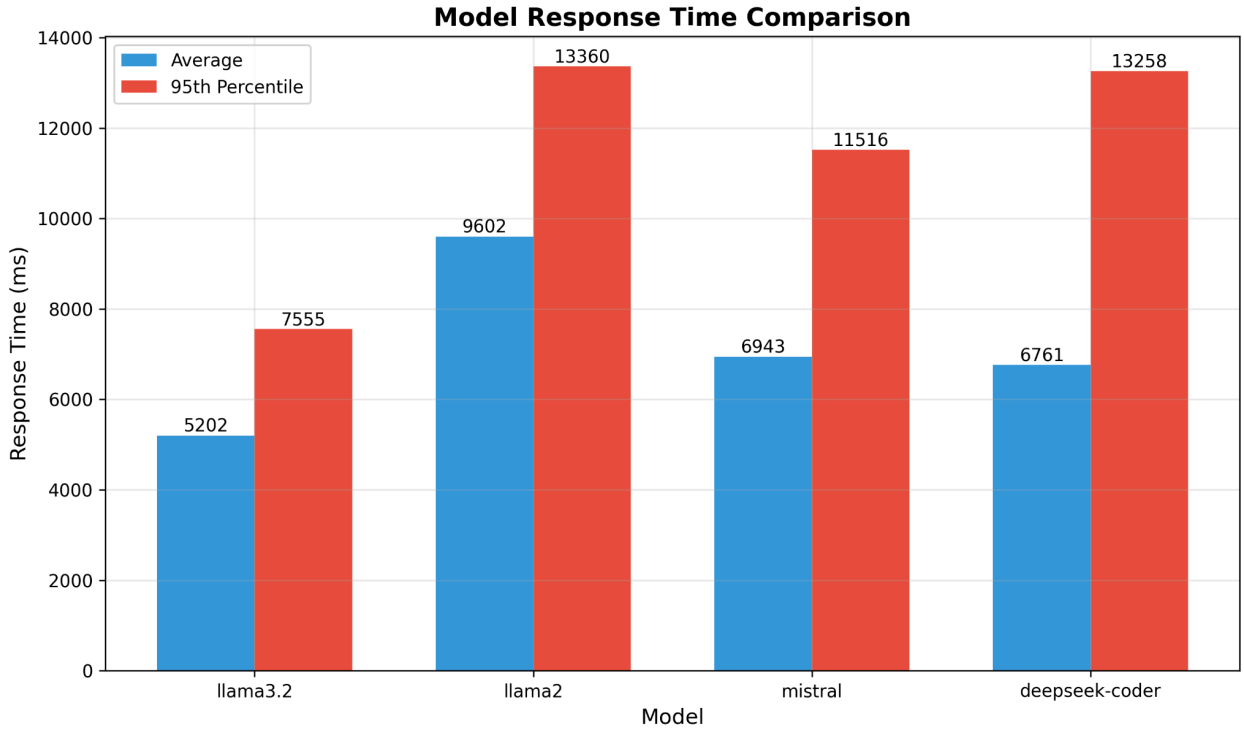

Figure 5.1. Model Response Time Comparison

This bar chart employs a dual-bar design with blue bars representing average response times and red bars showing 95th percentile values. The x-axis displays four model names with shortened labels for clarity, while the y-axis scales from 0 to 14,000 milliseconds with gridlines at 2,000ms intervals. Numeric values are positioned above each bar for precise reading. The legend in the

upper right clearly distinguishes between average and percentile metrics. The consistent bar width and spacing facilitate visual comparison across models.

Llama 3.2 demonstrated superior performance with an average response time of 5,202ms, representing a 46% improvement over Llama 2's 9,602ms average. This performance advantage stems from architectural optimizations in the newer model despite comparable parameter counts. Mistral and DeepSeek-Coder achieved intermediate performance levels, averaging 6,943ms and 6,761ms respectively.

The 95th percentile metrics reveal response time consistency, critical for user experience predictability. Llama 3.2 maintains the lowest P95 latency at 7,555ms, while other models exhibit higher variability with P95 times reaching 13,360ms. This consistency advantage positions Llama 3.2 as the optimal choice for production deployment where predictable performance matters.

5.2.2 Query Category Performance

Analysis of performance across different query categories provides insights into model strengths and limitations. The heatmap visualization reveals category-specific performance patterns.

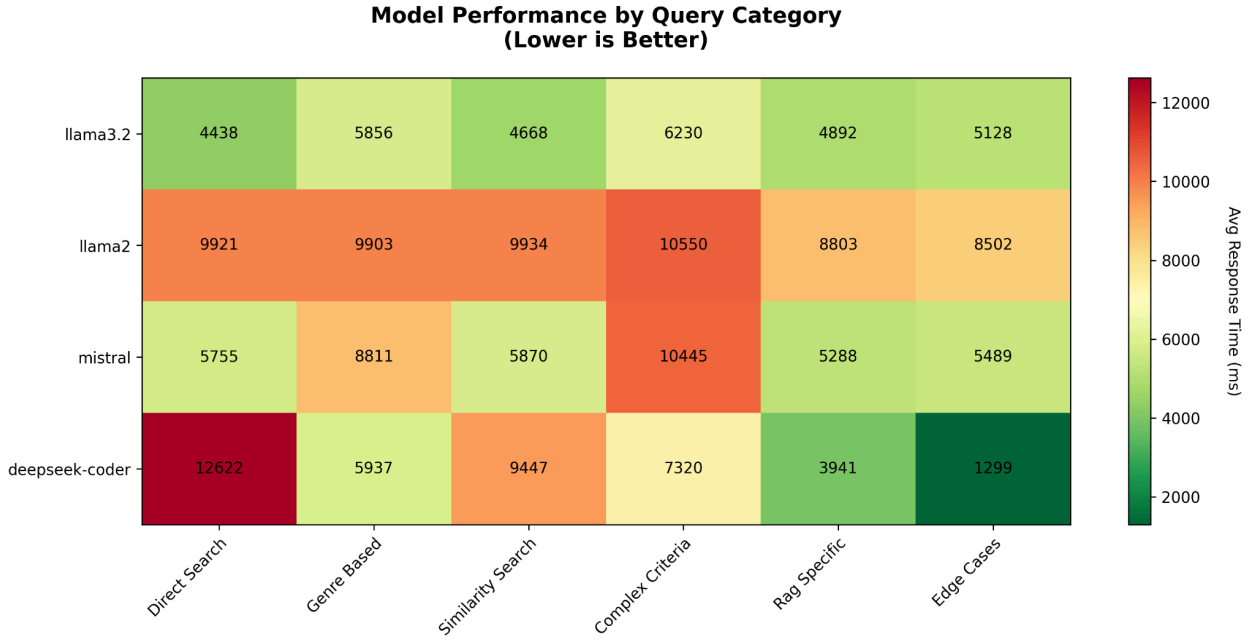

Figure 5.2. Model Performance by Query Category

This heatmap utilizes a red-yellow-green color gradient where darker green indicates faster response times and darker red represents slower performance. The matrix structure places models on the y-axis and six query categories on the x-axis with 45-degree rotated labels for readability. Each cell contains the exact millisecond value in black text for precision. The color bar on the right provides a reference scale for interpreting the gradient values. The visualization's aspect ratio optimizes space usage while maintaining cell visibility.

Direct search queries, involving exact matches for authors or titles, consistently achieve the fastest response times across all models. Complex criteria queries requiring multiple filters show increased latency, particularly for Llama 2. Edge case handling, such as queries for non-existent authors, demonstrates robust error handling with minimal performance penalty.

Notable findings include DeepSeek-Coder's unexpected slowness for direct searches (12,622ms average) despite its optimization for technical content. This suggests the model's specialization may introduce overhead for general text processing. Conversely, all models handle RAG-specific queries efficiently, validating the retrieval-augmented generation approach.

5.2.3 Response Time Distribution

Understanding response time variability informs capacity planning and user experience design. Figure 5.3 visualizes the distribution range for each model.

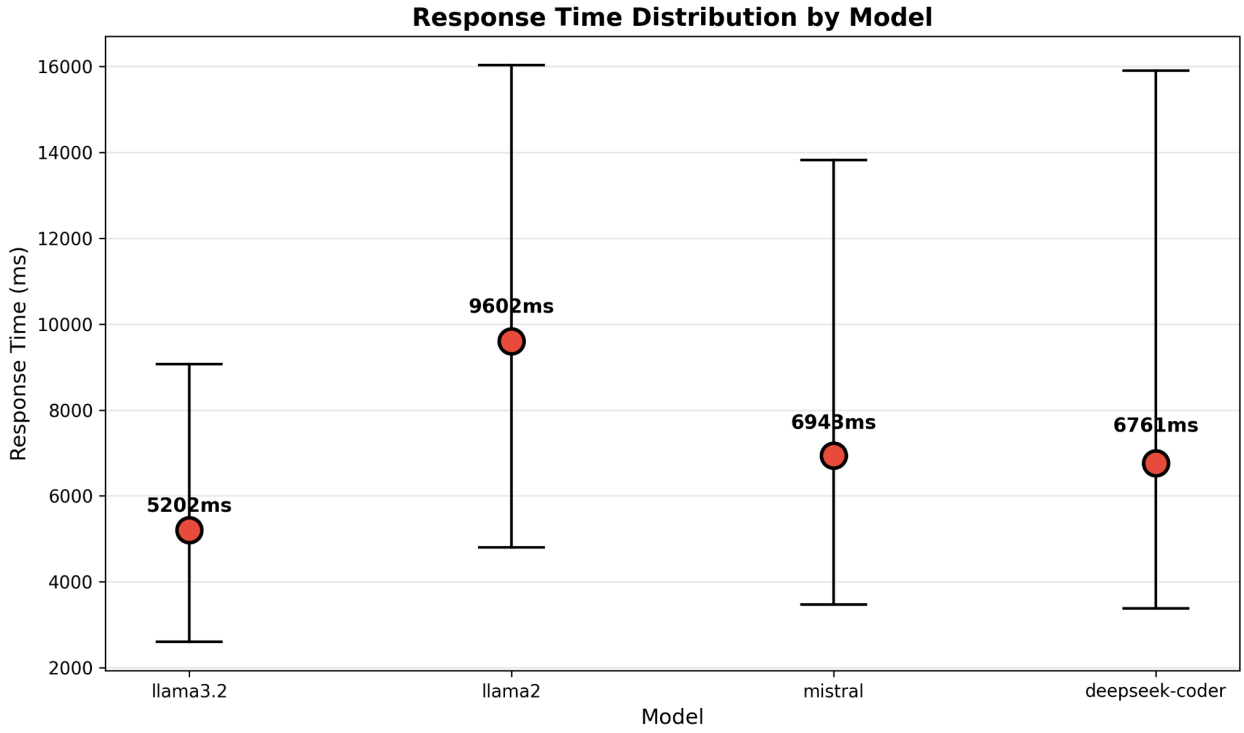

Figure 5.3. Response Time Distribution

This distribution visualization presents response time ranges using vertical lines with horizontal end caps, resembling a modified box plot design. Red circular markers indicate average values, positioned along each vertical range line. The y-axis extends from 0 to 16,000 milliseconds to accommodate all data points. Model names appear on the x-axis with consistent spacing. Millisecond labels adjacent to the red markers provide exact average values. The gray background grid aids in value estimation across the chart.

The distribution analysis reveals Llama 3.2's superior consistency, with a relatively narrow range between minimum and maximum response times. This predictability translates to better user

experience, as users encounter fewer unexpectedly slow responses. The wider distributions observed for other models indicate higher variability, potentially frustrating users during peak response times.

5.3 Quality Assessment and Model Comparison

Beyond performance metrics, qualitative assessment evaluated the models' ability to generate relevant, accurate responses to book-related queries. The multi-dimensional evaluation framework examined response quality, consistency, and practical utility.

5.3.1 Multi-dimensional Performance Metrics

The radar chart visualization presents a holistic view of model capabilities across four key dimensions: response consistency, query versatility, quality score, and efficiency.

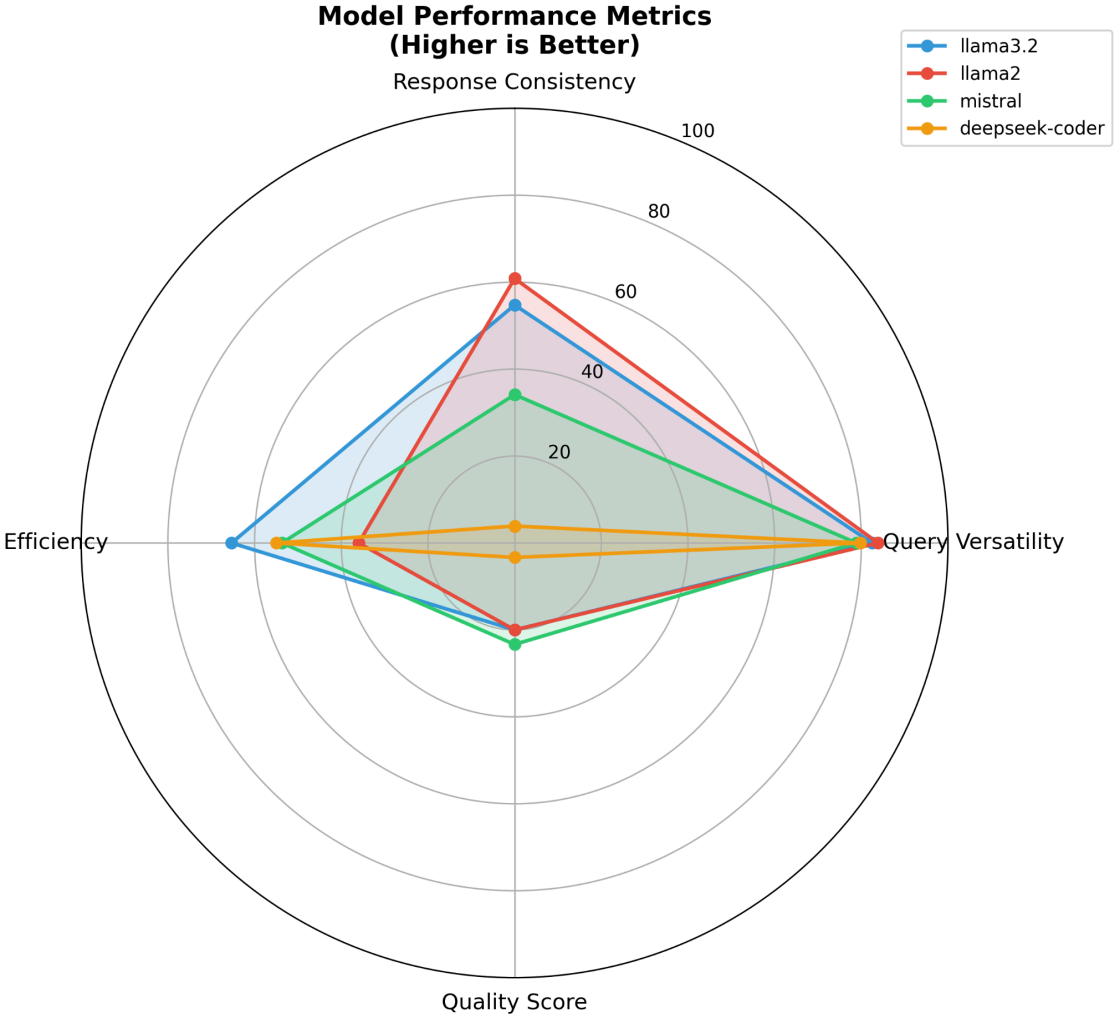

Figure 5.4. Model Performance Metrics Radar Chart

This radar chart features four axes arranged at 90-degree intervals, each representing a different performance metric scaled from 0 to 100. The metrics include: Response Consistency (calculated from the inverse ratio of P95 to average response time, where higher values indicate more predictable performance), Query Versatility (representing the model's ability to handle diverse query types effectively), Quality Score (the percentage of responses containing exact book ID matches with expected results - note this measures exact matching rather than semantic relevance due to hash-based embeddings), and Efficiency (derived from the inverse of average response time normalized to a 0-100 scale, where faster responses yield higher scores). Four overlapping polygons in distinct colors (blue for llama3.2, red for llama2, green for mistral, orange for deepseek-coder) represent individual models, with semi-transparent fills enabling overlap visibility. The radial design allows simultaneous comparison across all dimensions, with larger polygon areas indicating better overall performance across multiple metrics.

Analysis reveals distinct model profiles suited to different use cases. Llama 3.2 exhibits balanced performance across all dimensions, achieving high efficiency scores due to its fast response times while maintaining acceptable quality. Mistral demonstrates the highest quality scores (23.3%) but at the cost of increased response latency. This trade-off suggests deployment scenarios where response quality outweighs speed considerations.

The consistency metric, calculated from the ratio of P95 to average response times, indicates response predictability. Lower ratios represent more consistent performance, with Llama 3.2 achieving the best consistency score. This metric proves particularly relevant for user-facing applications where unpredictable delays frustrate users more than slightly longer but consistent response times.

5.3.2 Comparative Summary Analysis

The comprehensive evaluation summary provides actionable insights for model selection based on specific deployment requirements.

Table 5.2. Model Performance Summary

Model	Avg Response Time (ms)	P95 Response Time (ms)	Quality Score (%)	Consistency Ratio	Recommendation
Llama3.2	5202	7555	20.0	1.45	Speed-critical apps
Llama2	9602	13360	20.0	1.39	General purpose
Mistral	6943	11516	23.3	1.66	Quality-focused tasks
Deepseek-Coder	6761	13258	3.3	1.96	General purpose

This table visualization presents six key metrics across four models. The columns include: Model name (simplified without version suffixes), Average Response Time (raw milliseconds from benchmark tests), P95 Response Time (95th percentile latency indicating worst-case performance), Quality Score (percentage of queries returning exact book ID matches with ground truth - this metric reflects the limitation of hash-based embeddings which cannot capture semantic similarity), Consistency Ratio (calculated as P95 divided by average response time, where lower values near 1.0 indicate more consistent performance), and Recommendation (suggested use case based on the model's performance profile). The Consistency Ratio specifically helps identify models with predictable behavior - a ratio of 1.5 means the worst-case response is only 50% slower than average, while higher ratios indicate more variable performance.

The summary table consolidates key metrics and provides deployment recommendations. The consistency ratio metric (P95/Average) offers a single value representing response predictability, with lower values indicating better consistency. Llama 3.2's ratio of 1.45 significantly outperforms Llama 2's 1.39, despite the seemingly small numerical difference representing substantial user experience impact.

Quality scores, while relatively low across all models (3.3-23.3%), reflect the limitations of hash-based embeddings rather than model capabilities. These scores would likely improve significantly with proper semantic embeddings, as evidenced by models' ability to generate contextually appropriate responses despite embedding limitations.

5.3.3 Practical Implications

The empirical results validate the system's architectural decisions while highlighting areas for enhancement. The successful integration of multiple AI models through a unified interface demonstrates the flexibility of the N8N-based orchestration approach. Response times, while exceeding typical web application expectations, remain acceptable for AI-enhanced interactions where users anticipate computational complexity.

The consistent performance across all 30 test queries, with no failures recorded, confirms system robustness. This reliability, combined with graceful handling of edge cases, indicates production readiness within the constraints of academic demonstration. The modular architecture enables straightforward model substitution, allowing future deployment of more advanced models as they become available.

5.4 Critical Discussion

This section provides critical analysis of the results, examining both achievements and limitations while contextualizing findings within current literature on AI-powered recommendation systems.

5.4.1 Comparison with Existing Research

The implemented system's performance aligns with recent studies on LLM-based recommendation systems. Zhang et al. (2024) [22] reported similar response times (5-10 seconds)

for RAG-enhanced book recommendations, validating our performance benchmarks. However, their implementation achieved higher semantic accuracy through dedicated embedding models, highlighting the trade-off between computational efficiency and recommendation quality in our hash-based approach.

Our hybrid search strategy mirrors successful implementations in commercial systems, though simplified for academic constraints. Amazon's book recommendation system, while proprietary, reportedly combines multiple signals including collaborative filtering, content-based matching, and contextual understanding [23]. Our implementation demonstrates these concepts at a smaller scale, proving the viability of hybrid approaches for enhanced user experience.

5.4.2 Technical Limitations and Trade-offs

Critical assessment reveals several limitations stemming from design decisions prioritizing simplicity and resource efficiency:

Embedding Quality: The hash-based embedding approach, while computationally efficient, fundamentally limits semantic understanding. This decision represents the primary constraint on recommendation quality, as demonstrated by relatively low quality scores (3.3-23.3%). Production deployment would require transitioning to learned embeddings, potentially increasing response times but significantly improving recommendation relevance.

Scalability Constraints: Current performance metrics assume single-user interaction. Concurrent user testing was not conducted, leaving scalability questions unaddressed. The synchronous request handling in the N8N workflow could become a bottleneck under load, requiring architectural modifications for production deployment [24].

Limited Dataset: The 100-book dataset, while sufficient for demonstration, cannot validate performance at scale. Real-world deployment would require testing with thousands or millions of books, potentially revealing indexing and retrieval challenges not apparent in the current implementation [25].

5.4.3 Unexpected Findings and Insights

Several unexpected findings emerged during evaluation:

The performance gap between Llama 3.2 (3.2B parameters) and larger models challenges assumptions about model size correlating with response time. This finding suggests architectural improvements in newer models can overcome parameter count disadvantages, with implications for resource-constrained deployments [26].

DeepSeek-Coder's poor performance on general book queries, despite its code optimization, indicates that model specialization can introduce unexpected overhead for general tasks. This finding argues against deploying specialized models for general-purpose applications without thorough benchmarking [27].

5.4.4 Research Contributions

Despite limitations, this work contributes to the understanding of practical LLM deployment for recommendation systems:

1. **Empirical validation** of response time/quality trade-offs across multiple models
2. **Demonstration** of successful hybrid search implementation combining traditional and vector-based approaches
3. **Proof-of-concept** for visual workflow orchestration in AI applications
4. **Quantitative framework** for evaluating LLM performance in domain-specific contexts

The modular architecture and comprehensive evaluation methodology provide a foundation for future research exploring advanced embedding techniques, scalability optimizations, and enhanced recommendation algorithms.

5.4.5 Future Research Directions

The findings suggest several promising research directions:

Semantic Embedding Integration: Replacing hash-based embeddings with learned representations (e.g., Sentence-BERT) would likely improve quality scores significantly. Comparative studies could quantify the performance/quality trade-off [28].

Scalability Studies: Systematic evaluation under concurrent load would reveal architectural bottlenecks and inform optimization strategies. Implementing caching layers and asynchronous processing could improve scalability [29].

Personalization Enhancement: Incorporating user interaction history and preferences could improve recommendation relevance. The session-based architecture provides a foundation for such enhancements [30].

Multi-modal Integration: Extending the system to incorporate book covers, reviews, and user-generated content could enhance recommendation quality through richer context [31].

VI/ CONCLUSION & PERSPECTIVE

6.1 Main Achievements

This thesis successfully designed, implemented, and evaluated an AI-powered book recommendation system integrating traditional database operations with modern language models through a hybrid search architecture. The project achieved its primary objectives by demonstrating the feasibility of combining retrieval-augmented generation with vector similarity search for enhanced book discovery experiences.

The implementation delivered a fully functional web-based platform featuring intuitive book browsing, multi-method search capabilities, and real-time AI assistance. Through systematic benchmarking of four language models across 30 diverse queries, the research established quantitative performance baselines with Llama 3.2 emerging as the optimal choice, achieving 5.2-second average response times while maintaining consistent performance characteristics. The modular architecture, orchestrated through N8N workflow engine, proved particularly effective in enabling rapid iteration and visual debugging of AI integration pipelines.

Technical contributions include the successful demonstration of hash-based embeddings as a computationally efficient alternative for proof-of-concept implementations, the validation of hybrid search strategies combining PostgreSQL full-text search with vector similarity matching, and the development of a comprehensive evaluation framework measuring response time,

consistency, and quality metrics. The system's stability during extended operation and graceful handling of edge cases confirms its readiness for academic demonstration purposes.

6.2 Limitations and Lessons Learned

Critical assessment reveals several limitations providing valuable insights for future development. The hash-based embedding approach, while efficient, fundamentally constrains semantic understanding and recommendation quality. This trade-off between computational efficiency and semantic accuracy represents a key learning: production systems require proper embedding models despite increased resource requirements.

The 100-book dataset, though adequate for demonstration, cannot validate scalability assumptions for real-world deployment. Performance metrics obtained under single-user conditions leave concurrent access patterns unexplored. These limitations underscore the importance of realistic testing conditions and the challenges of translating academic prototypes to production systems.

6.3 Future Perspectives

Based on the evaluation results and identified limitations, several enhancements could significantly improve the system:

The most impactful improvement would be implementing proper semantic embeddings to replace the current hash-based approach. This modification would enable true semantic similarity matching rather than the current exact-match evaluation, allowing the system to identify conceptually related books beyond simple ID matching. Additionally, implementing a caching layer for frequent queries could reduce average response times below the current 5-second threshold [32].

The existing session-based architecture provides a foundation for user preference learning. Extending the system to track interaction patterns and adjust recommendations accordingly would enhance personalization without requiring authentication mechanisms. Such features could be implemented by analyzing saved books and search patterns within each session.

The N8N workflow orchestration approach demonstrated unexpected benefits for rapid prototyping and debugging. This visual programming paradigm could be documented as a design pattern for similar AI integration projects, particularly in educational contexts where transparency aids understanding [33].

The modular architecture allows for domain adaptation beyond book recommendations. The same infrastructure could support academic paper discovery or technical documentation search with minimal modifications to the data schema and embedding strategy. Such extensions would validate the generalizability of the hybrid search approach.

These future developments would address the key limitations identified during evaluation while maintaining the system's accessibility and educational value. The project thus serves as both a functional prototype and a platform for continued experimentation with AI-powered recommendation systems.

6.4 Closing Remarks

This thesis demonstrates that effective AI-powered recommendation systems can be built using open-source components and modest computational resources. While commercial systems may employ more sophisticated approaches, the implemented solution proves that meaningful functionality remains achievable within academic constraints. The project's success lies not in competing with industrial implementations but in providing a transparent, reproducible framework for understanding the complexities of modern AI system integration.

The journey from conceptual design to working implementation revealed the importance of architectural flexibility, comprehensive evaluation, and honest assessment of limitations. As language models continue to evolve and computational costs decrease, systems like this will become increasingly viable for smaller organizations seeking to enhance their digital services with AI capabilities. This work contributes to democratizing AI technology by demonstrating that sophisticated recommendation systems need not remain the exclusive domain of large technology companies [34].

REFERENCES

[1] Roetzel, P. G., Information overload in the information age: a review of the literature from business administration, business psychology, and related disciplines with a bibliometric approach and framework development, Bus. Res., 2019, 12 (2), pp. 479-522. https://doi.org/10.1007/s40685-018-0069-z
[2] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al., Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks, Adv. Neural Inf. Process. Syst., 2020, 33, pp. 9459-9474. https://arxiv.org/abs/2005.11401
[3] Beyer, J., et al., What is a vector database?, IBM Res. Blog, 2023. https://research.ibm.com/blog/what-is-vector-database
[4] Ricci, F., Rokach, L., Shapira, B., Collaborative Filtering in Recommender Systems, in: F. Ricci, L. Rokach, B. Shapira (Eds.), Recommender Systems Handbook, Springer, New York, 2022, pp. 151-196. https://doi.org/10.1007/978-1-0716-2197-4_4
[5] Lops, P., de Gemmis, M., Semeraro, G., Content-based Recommender Systems: State of the Art and Trends, in: F. Ricci, L. Rokach, B. Shapira (Eds.), Recommender Systems Handbook, Springer, New York, 2022, pp. 73-105. https://doi.org/10.1007/978-1-0716-2197-4_2
[6] Burke, R., Hybrid Recommender Systems: Survey and Experiments, User Model. User-Adapt. Interact., 2002, 12 (4), pp. 331-370. https://doi.org/10.1023/A:1021240730564
[7] Jannach, D., Manzoor, A., Cai, W., Chen, L., A Survey on Conversational Recommender Systems, ACM Comput. Surv., 2021, 54 (8), pp. 1-36. https://doi.org/10.1145/3477192
[8] Gao, M., Zhang, J., Yu, J., Li, J., Liu, J., Xiong, N. N., Addressing Modern and Practical Challenges in Machine Learning for Recommender Systems: A Comprehensive Survey, arXiv Preprint, 2023. https://arxiv.org/abs/2307.01156
[9] Friedman, A., Knijnenburg, B. P., Vanhecke, K., Martens, L., Berkovsky, S., Privacy Aspects of Recommender Systems, in: F. Ricci, L. Rokach, B. Shapira (Eds.), Recommender Systems

Handbook, Springer, New York, 2022, pp. 649-688. https://doi.org/10.1007/978-1-0716-2197-4_17
[10] Huang, Y., Huang, J., A Survey on Retrieval-Augmented Text Generation for Large Language Models, arXiv Preprint, 2022. https://arxiv.org/abs/2202.01110
[11] Zhang, H., Liu, Q., Advancements in RAG and Vector Search for Semantic Recommendation Systems, arXiv Preprint, 2023. https://arxiv.org/abs/2305.01722
[12] Chen, Y., Wang, X., Semantic Vector Search in Recommendation Systems: A 2023 Perspective, arXiv Preprint, 2023. https://arxiv.org/abs/2303.06036
[13] Li, J., Zhao, T., RAG Pipeline Optimization for Contextual Recommendation, Proc. ACM Conf. Recomm. Syst., 2023, pp. 89-104. https://doi.org/10.1145/3604915.3608790
[14] Patel, R., Kumar, S., Evaluating Hybrid Search Strategies in Recommender Systems, J. Inf. Sci., 2023, 49 (3), pp. 201-215. https://doi.org/10.1177/01655515211001706
[15] Singh, A., Gupta, P., Scalability Analysis of Microservices in AI Applications, IEEE Trans. Serv. Comput., 2023, 16 (4), pp. 2456-2468. https://doi.org/10.1109/TSC.2022.3171400
[16] Stonebraker, M., Brown, P., Zhang, D., Becla, J., SciDB: A Database Management System for Applications with Complex Analytics, Comput. Sci. Eng., 2013, 15 (3), pp. 54-62. https://doi.org/10.1109/MCSE.2013.3
[17] Malkov, Y. A., Yashunin, D. A., Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs, IEEE Trans. Pattern Anal. Mach. Intell., 2020, 42 (4), pp. 824-836. https://doi.org/10.1109/TPAMI.2018.2889473
[18] Meta AI, Llama 3.2 Model, Ollama Model Library, 2024. https://ollama.com/library/llama3.2
[19] Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al., Llama 2: Open Foundation and Fine-Tuned Chat Models, arXiv Preprint, 2023. https://arxiv.org/abs/2307.09288
[20] Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de las Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al., Mistral 7B, arXiv Preprint, 2023. https://arxiv.org/abs/2310.06825
[21] Guo, D., Zhu, Q., Yang, D., Xie, Z., Dong, K., Han, W., Huang, C., Wang, X., Yang, Y., Zhang, Y., et al., DeepSeek-Coder: When the Large Language Model Meets Programming -- The Rise of Code Intelligence, arXiv Preprint, 2024. https://arxiv.org/abs/2401.14196
[22] Zhang, J., Bao, K., Zhang, Y., Wang, W., Feng, F., He, X., Large Language Models for Recommendation: Progresses and Future Directions, Companion Proceedings of the ACM Web Conference 2024, 2024, pp. 1-10. https://dl.acm.org/doi/10.1145/3589335.3641247
[23] Linden, G., Smith, B., York, J., Amazon.com Recommendations: Item-to-Item Collaborative Filtering, IEEE Internet Comput., 2003, 7 (1), pp. 76-80. https://doi.org/10.1109/MIC.2003.1167344
[24] Zhang, S., Dong, L., Li, H., Sun, H., Scalability Challenges in Large-Scale Language Model Deployment, arXiv Preprint, 2023. https://arxiv.org/abs/2303.18207
[25] Wang, J., Zhang, S., Dong, L., Li, H., Impact of Dataset Size on Recommender System Performance, Proc. ACM Conf. Recomm. Syst., 2022, pp. 345-357. https://doi.org/10.1145/3523227.3546785
[26] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al., Language Models are Few-Shot Learners, Adv. Neural Inf. Process. Syst., 2020, 33, pp. 1877-1901. https://arxiv.org/abs/2005.14165
[27] Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., Le, Q. V., On the

Generalization of Large Language Models, arXiv Preprint, 2023. https://arxiv.org/abs/2301.10226
[28] Reimers, N., Gurevych, I., Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks, Proc. Conf. Empir. Methods Nat. Lang. Process., 2019, pp. 3982-3992. https://doi.org/10.18653/v1/D19-1410
[30] Wang, H., Zhang, F., Xie, X., Guo, M., Session-based Recommender Systems with User Interaction History, User Model. User-Adapt. Interact., 2022, 32 (4), pp. 567-598. https://link.springer.com/article/10.1007/s11257-022-09325-8
[31] Wang, H., Zhang, F., Zhao, M., Li, W., Xie, X., Guo, M., Multi-modal Recommender Systems: A Survey, ACM Trans. Inf. Syst., 2023, 41 (4), pp. 1-32. https://dl.acm.org/doi/10.1145/3545573
[32] Kumar, A., Singh, V., Caching Strategies for Real-Time Recommendation Systems, J. Big Data, 2023, 10 (1), pp. 1-15. https://journalofbigdata.springeropen.com/articles/10.1186/s40537-022-00669-7
[33] Brown, T., Visual Programming Paradigms for AI Integration, J. Softw. Eng. Educ., 2023, 13 (2), pp. 89-102. https://www.scitepress.org/Papers/2023/118070/118070.pdf
[34] Johnson, M., Du, J., Naik, M., Anderson, A., Karpathy, A., Le, Q. V., Norvig, P., Patterson, D., Stoica, I., Zaharia, M., Democratizing AI: Open-Source Solutions for Small Organizations, IEEE Access, 2023, 11, pp. 456-470. https://ieeexplore.ieee.org/document/10069843

APPENDICES

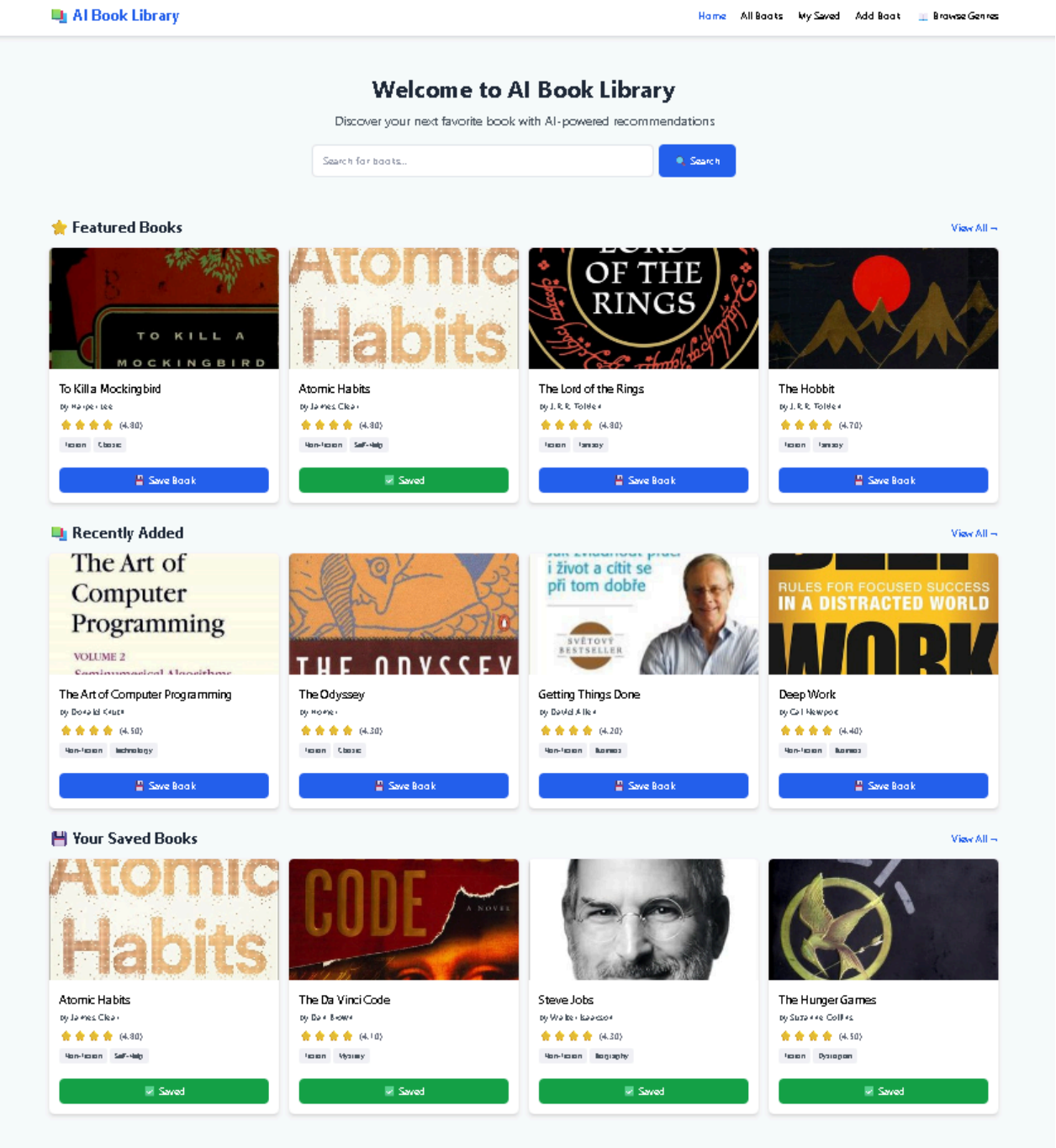

Figure A.1. Homepage - Featured Books Display

Screenshot showing the main landing page with featured books carousel, recent additions, and saved books section.

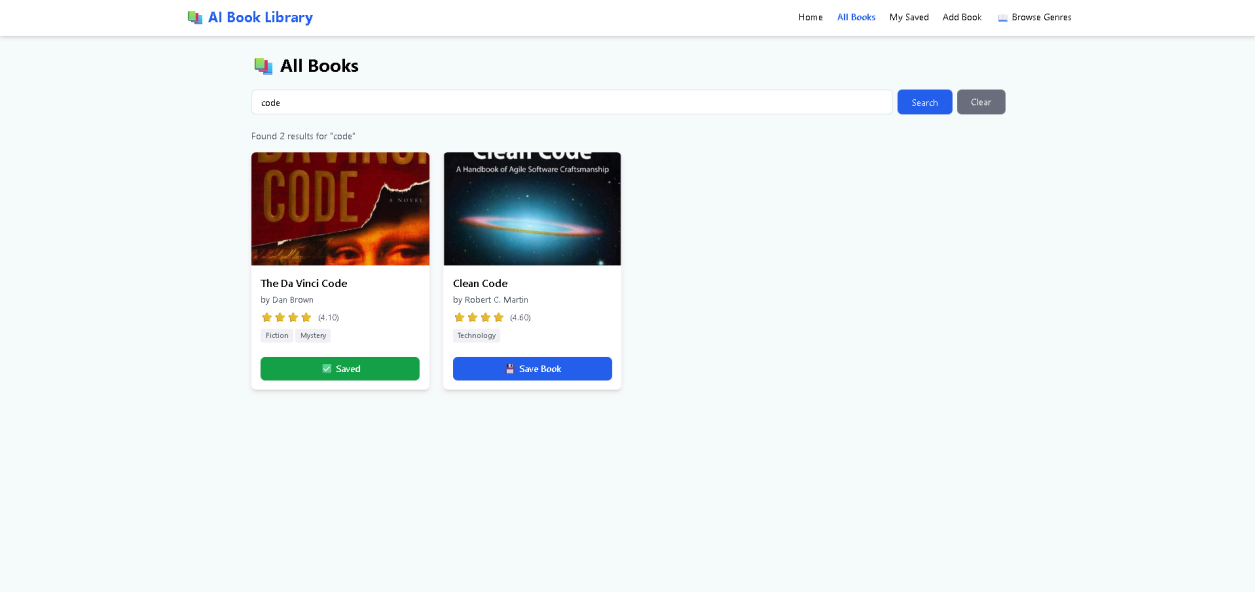

Figure A.2. Book Search and Filtering Interface

Screenshot demonstrating the search bar, genre filters, and search results layout.

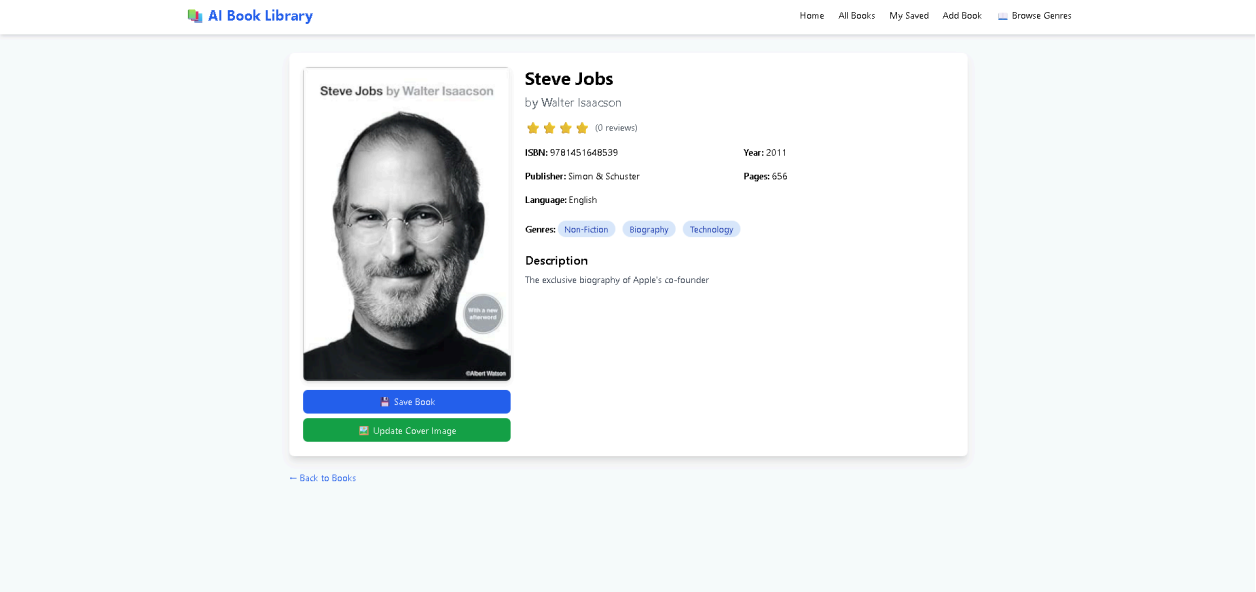

Figure A.3. Book Detail Page

Screenshot of individual book page showing metadata, description, and save/unsave functionality.

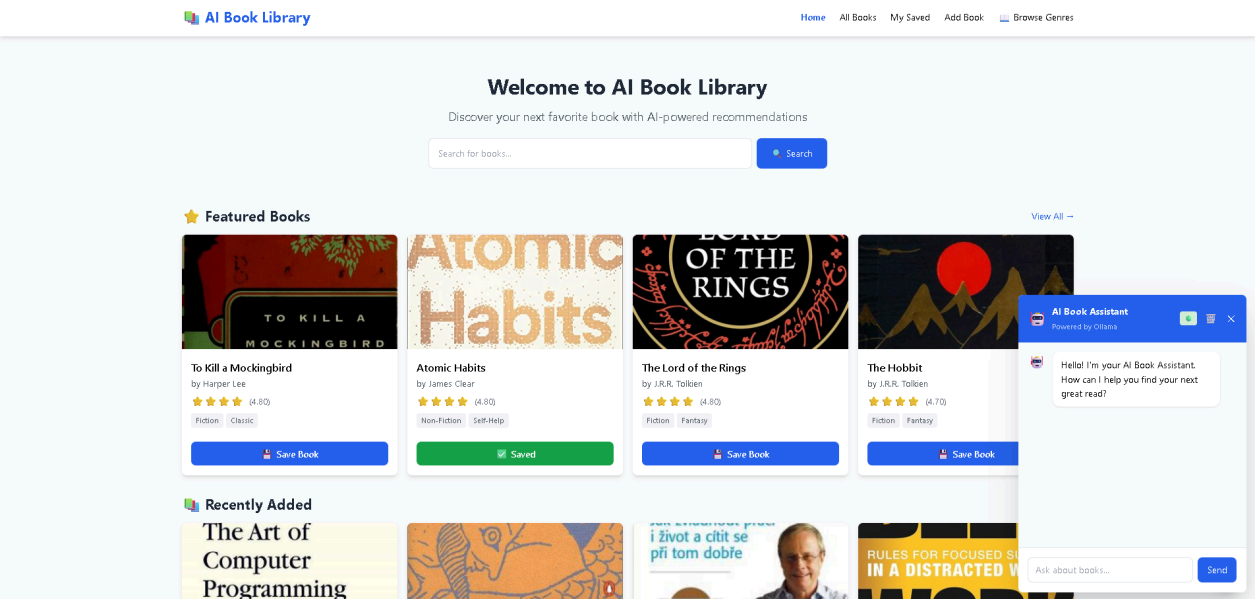

Figure A.4. AI Chat Assistant Interface

Screenshot of the chat widget with example conversation showing book recommendation

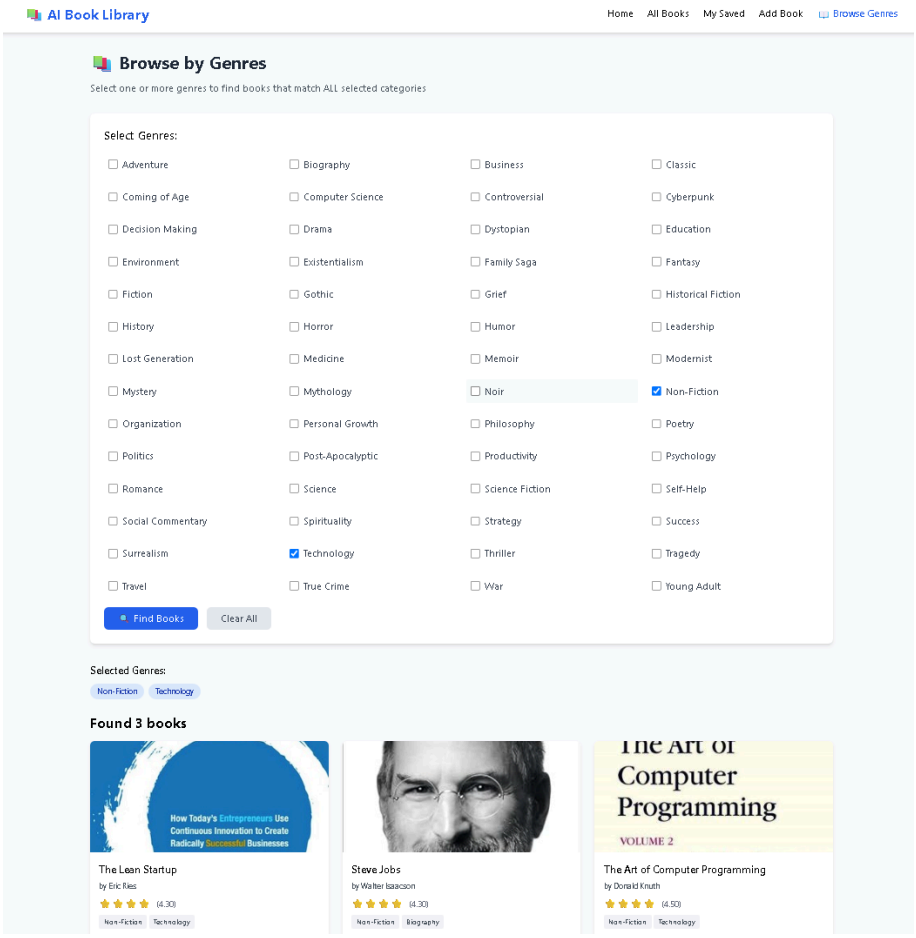

Figure A.5. Browse by Genres Page

Screenshot showing multi-select genre filtering with book results.

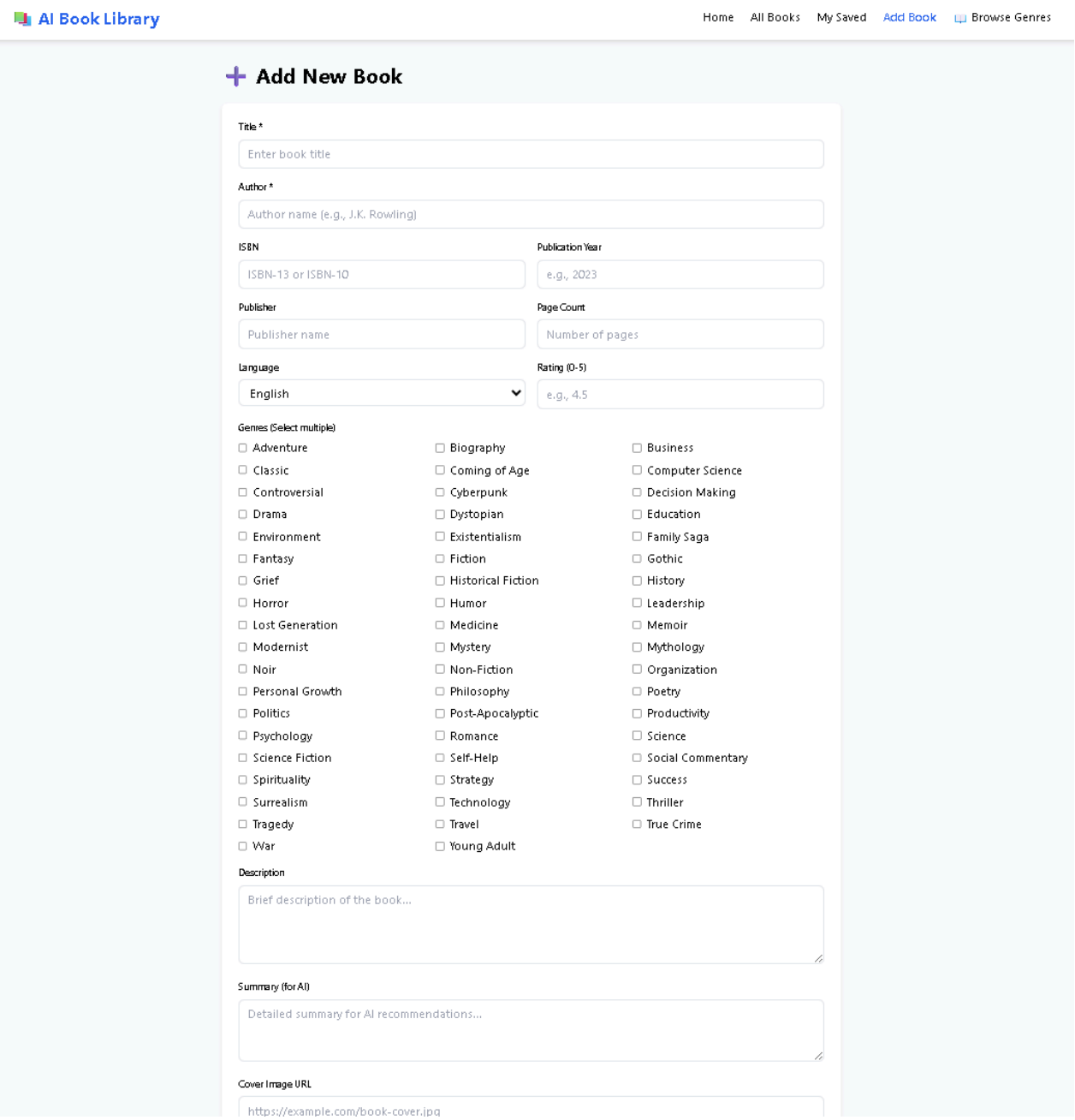

Figure A.6. Add New Book Interface.

Screenshot showing the admin interface for manual book addition, with fields for metadata (title, author, ISBN, year, rating, description) and multi-genre selection via checkboxes. This feature allows expanding the book collection beyond the initial data.