Semantic Manga RAG: Retrieval System for Manga Using Natural Language Description

Zhaoyang Zhu
University of Illinois Urbana-Champaign
Urbana, Illinois, USA
zzhu62@illinois.edu

Wei Chen
University of Illinois Urbana-Champaign
Urbana, Illinois, USA
weic6@illinois.edu

Abstract

Manga enthusiasts often struggle to relocate stories they've encountered but can't recall by title or author—a frequent problem when readers reads random panels online. Conventional search tools fail to support these vague, memory-based queries, forcing users into time-consuming manual searches. We present Semantic Manga RAG, a retrieval system that lets users search a manga library with natural-language descriptions. Our approach combines multimodal understanding of manga content with Retrieval-Augmented Generation (RAG) to transform vague memory into accurate search results. The system builds a customized vector database from user-supplied manga, uses Gemini-2.0-flash to summarise each book, page, and panel, and applies vector search, then LLM re-ranking that also explains each match. Evaluation on a test set of memory-fragment queries demonstrated a Top-5 accuracy of 78.95%, confirming the system's effectiveness for memory-based manga retrieval. This paper details the architecture, implementation, and performance of our system, which helps readers relocate manga from memory alone.

Keywords

Retrieval-Augmented Generation, Semantic search, Vector database

ACM Reference Format:
Zhaoyang Zhu and Wei Chen. 2025. Semantic Manga RAG: Retrieval System for Manga Using Natural Language Description. In . ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Manga, Japanese comic books or graphic novels, have gained immense popularity worldwide. Enthusiasts often encounter manga casually online—through viral panels or screenshots—with no record of the title or author. Users may struggle to recall what the manga was called or where they saw it. Most existing manga search tools only support exact or fuzzy search by title or visual-based image search, failing to support vague, memory-based queries. This forces users to rely on time-consuming manual search methods, such as asking others online or browsing forums in hopes of recognition.

Our proposed tool, Semantic Manga RAG, lets users search with short, imperfect text descriptions instead of titles or images. The system is built upon the concept of Retrieval-Augmented Generation (RAG), which combines information retrieval with LLM to provide accurate and contextually relevant results.

The main contributions of this work are:

- An approach that accepts free-form text memories and returns semantically relevant pages
- A pipeline that creates textual summaries of manga at book, page, and panel levels
- A two-stage retrieval process: vector similarity search followed by LLM re-ranking with explanations
- An evaluation framework for memory-based manga retrieval

The source code for our project is available on GitHub at: https://github.com/weic6/Retrieval-System-for-Manga. The repository contains all necessary code, configuration files, and documentation to reproduce our results and deploy the system for personal use.

This paper is organized as follows: Section 2 presents the background and related work. Section 3 describes the system architecture and methodology. Section 4 details implementation specifics. Section 5 presents our evaluation results. Section 6 provides detailed installation and usage instructions. Section 7 discusses limitations and future work. Finally, Section 8 concludes the paper.

2 Background and Related Work

2.1 Manga Search Methods

Traditional approaches to manga search rely primarily on metadata such as titles, authors, genres, and tags. These methods fail when users can only recall content fragments rather than specific metadata. Image-based search approaches like reverse image search require users to have a visual reference, which is often not the case with faded memories.

2.2 Retrieval-Augmented Generation (RAG)

RAG combines retrieval systems with generative models to enhance the quality and accuracy of AI-generated content. In a RAG system, a retrieval component first fetches relevant information from a knowledge base, which is then fed to a language model to generate outputs grounded in that retrieved information [1]. This approach helps overcome limitations of pure generative models by providing them with additional context from external knowledge sources.

2.3 Vector Databases for Semantic Search

Vector databases store high-dimensional vector representations (embeddings) of data, allowing for efficient similarity search. For text data, these embeddings capture semantic meaning, enabling

Conference'17, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

search based on conceptual similarity rather than exact keyword matching. This is particularly valuable for natural language queries where users may describe concepts using different terminology than what appears in the source material.

2.4 Multimodal Content Understanding

Recent advances in multimodal AI models enable joint processing of text and images, allowing for comprehensive understanding of visual content with textual elements—a perfect fit for manga analysis. Models like Gemini can interpret complex visual scenes while also processing text within images, making them suitable for extracting meaningful information from manga pages.

3 System Design

Semantic Manga RAG consists of two primary modules: the Preprocessing Module and the Query Module. Figure 1 illustrates the overall system architecture.

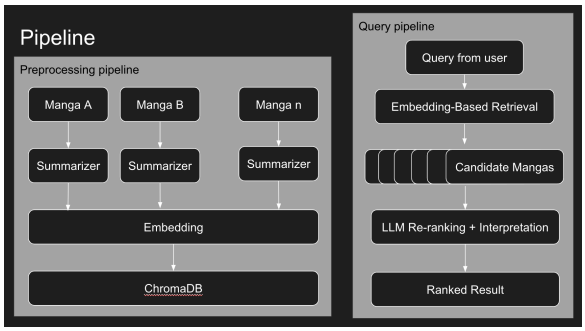


Figure 1: System architecture: the preprocessing pipeline (left) and query pipeline (right). The preprocessing pipeline processes manga content into embeddings stored in ChromaDB, while the query pipeline retrieves and ranks matching manga based on natural language queries.

3.1 Preprocessing Module

The Preprocessing Module is responsible for analyzing manga content and creating searchable representations. It processes manga at three hierarchical levels:

- **Book level:** Overall summary of the entire manga
- **Page level:** Summaries of individual pages
- **Panel level:** Detailed descriptions of individual panels, including characters, settings, actions, dialogue, and visual elements

The preprocessing pipeline follows these steps:

(1) **Manga Input:** Users provide manga images organized by book and page
(2) **Content Analysis:** A multimodal LLM (Gemini-2.0-flash) analyzes each page and generates structured JSON summaries contain summaries, characters, items and other details.
(3) **Vector Embedding:** The structured data are embedded into vector representations using an embedding model
(4) **Database Storage:** Embeddings and metadata are stored in ChromaDB for efficient retrieval

The JSON schema for manga summaries includes rich details about characters (names, expressions, poses), settings (locations, background elements), narrative (actions, dialogue, emotions), and text elements (onomatopoeia, signage). This structured approach ensures comprehensive content understanding suitable for various query types. Figure 2 illustrates an example input and json schema.

3.2 Query Module

The Query Module handles user queries and retrieves relevant manga content through a two-stage process:

(1) **Initial Retrieval:** Vector similarity search identifies candidate manga elements based on semantic similarity to the query
(2) **LLM Re-ranking:** A large language model re-ranks candidate results and provides explanations for why each result matches the query

The LLM re-ranking process considers multiple factors:

- Match specificity (panel-level vs. page-level vs. book-level)
- Character details (names, expressions, poses)
- Setting details (location, background elements)
- Narrative elements (actions, dialogue, emotions)
- Visual text elements

The first stage quickly narrows candidates; the second stage refines them and explains each match.

4 Implementation

4.1 Technologies Used

Our implementation uses the following technologies:

- **Programming Language:** Python 3.11
- **Multimodal Understanding:** Google's Gemini-2.0-flash
- **Vector Database:** ChromaDB
- **Embedding Model:** Default embedding function provided by ChromaDB
- **Additional Libraries:** tqdm (progress tracking), dotenv (environment configuration)

4.2 Preprocessing Pipeline

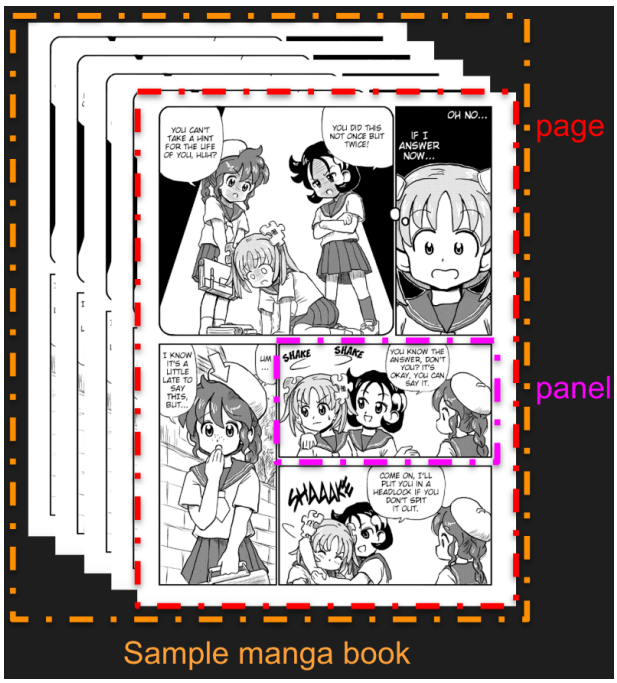
The preprocessing pipeline is implemented in `preprocessing.py` and `vectorize.py`.

The preprocessing script:

- Traverses manga directories to find image files (.jpg, .jpeg, .png, .webp)
- Sends each page to Gemini-2.0-flash for analysis
- Constructs hierarchical JSON representations with book, page, and panel summaries
- Saves these as schema files in a designated directory

The vectorization script:

- Loads manga schema files
- Creates documents for vectorization at book, page, and panel levels
- Embeds documents using ChromaDB's default embedding function



(a) Sample manga page with page-level (red dashed line) and panel-level (pink dashed line) content. Each level provides different granularity for retrieval.

(b) JSON schema used to represent manga content hierarchically. The schema captures book-level, page-level (red), and panel-level (pink) details, enabling multi-granular search.

Figure 2: Manga content representation: (a) Original manga content with hierarchical structure highlighted and (b) corresponding JSON schema generated by the preprocessing pipeline to enable semantic search.

- Stores vectors and metadata in a ChromaDB collection for retrieval

4.3 Query Pipeline

The query functionality is implemented in `query.py`, with two main components:

4.3.1 Raw Search. The raw search function queries ChromaDB using cosine similarity between the embedded query and stored manga content. It returns candidate results with similarity scores and metadata.

4.3.2 LLM Re-ranking. For re-ranking, candidates are sent to Gemini-2.0-flash along with the original query and instructions to evaluate relevance. The LLM analyzes each candidate, assigns relevance scores (0-100), provides detailed explanations of why each result matches the query, and identifies the match type (character, setting, narrative, text elements, or overall).

4.4 Evaluation Framework

The evaluation system in `eval.py` measures retrieval performance using test queries that simulate memory fragments. For each manga, multiple query variations are tested, with accuracy calculated based on whether the correct manga appears in the top 5 results.

5 Evaluation

To evaluate the effectiveness of our system, we constructed a test set of manga books with corresponding query-document pairs. Each manga had 5 different query variations, simulating realistic user queries based on vague or partial plot recall.

5.1 Metric

We employ the metric **Top K Accuracy @ 5**. This metric measures the probability that relevant documents are retrieved among the top five results. The formula is defined as follows:

$$\text{Top-5 Accuracy} = \frac{1}{N}\sum_{q=1}^{N} I_q$$

where N is the total number of queries or items evaluated, and I_q is an indicator function that equals 1 if the relevant document for query q is among the top 5 results, and 0 otherwise. We chose this as our primary evaluation metric because our system presents 5 results for users in actual use. Consequently, Top K Accuracy @ 5 directly reflects the user's success rate in a real-world scenario and is thus directly correlated with user experience.

5.2 Experimental Setup

The evaluation dataset consisted of manga samples with test queries deliberately formulated to mimic memory fading and variations in user focus. For example, for a manga called "Taking Photos Bang Dream," test queries included variations like:

- "Photography slice of life manga, cute girls, school setting maybe, someone likes taking pictures."
- "Manga about memories and friendship, involves a camera, somewhat comedic, slice of life."

- "Cute girls manga, photography club maybe, one girl is embarrassed about her pictures."

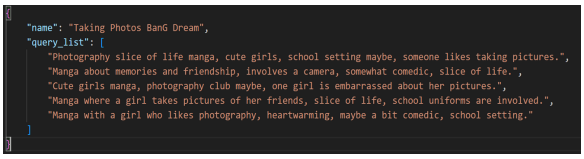


Figure 3: Sample from the test dataset showing a manga with five different query variations. These queries simulate different ways users might remember or describe the same manga content, with varying levels of detail and focus.

5.3 Results

Table 1: Evaluation Results for Top 5 Accuracy

Total Samples	Relevant Found in Top 5	Top-5 Accuracy
190	150	0.7895

Our system achieved a Top-K Accuracy @ 5 of 0.7895, meaning that for approximately 79% of test queries, the correct manga appeared within the top 5 results. This demonstrates the system's effectiveness in retrieving manga based on vague memory descriptions.

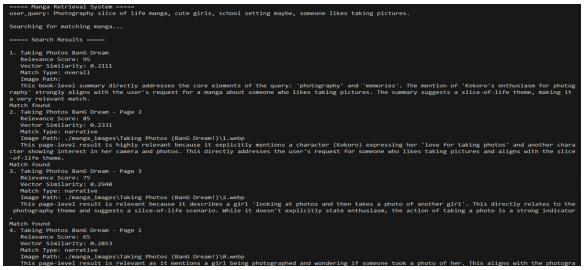


Figure 4: Evaluation results showing a query for "Photography slice of life manga, cute girls, school setting maybe, someone likes taking pictures" and the system's top results. The correct manga (Taking Photos Bang Dream) appears as the top result with a detailed explanation of the match.

Qualitative analysis of the results showed that:

- Panel-level matches typically received higher relevance scores due to their specificity
- The LLM re-ranking significantly improved result ordering compared to raw vector similarity alone
- Explanations provided by the system helped users understand why certain manga matched their queries

6 Installation and Usage Guide

This section provides comprehensive instructions for installing, configuring, and using the Semantic Manga RAG system.

6.1 System Requirements

- Python 3.11
- Google API key with access to Gemini API

6.2 Installation

(1) Clone the repository:

```
git clone https://github.com/weic6/Retrieval-System-for-Manga.git
cd Retrieval-System-for-Manga
```

(2) (Optional) Create and activate a virtual environment:

```
python -m venv manga_env
# On Windows
manga_env\Scripts\activate
# On macOS/Linux
source manga_env/bin/activate
```

(3) Install dependencies:

```
pip install -r requirements.txt
```

(4) Set up API key:

- Rename `.env_sample` to `.env`
- Edit `.env` and add your Google API key: `API_KEY=your_key`

6.3 Data Preparation

Before using the system, you need to prepare your manga files according to the following structure:

```
manga_images/
        manga_name_1/
                page1.png
                page2.png
                ...
        manga_name_2/
                page1.webp
                page2.webp
                ...
        ...
```

Guidelines for best results:

- Create a folder for each manga title inside the `manga_images` directory
- Name image files in numerical order (e.g., page1.png, page2.png)
- Supported image formats: .jpg, .jpeg, .png, .webp
- Ensure images are clean and legible for better content analysis

A sample dataset is available for you to try the application:

```
# Download sample manga images
wget https://drive.google.com/drive/folders/1I0EOQyQLR32NNWqxWrcv7oKKc-OLzy29?usp=sharing -O sample_manga.zip
unzip sample_manga.zip -d manga_images/
```

6.4 Usage Workflow

6.4.1 Step 1: Preprocessing Manga. Run the preprocessing script to analyze manga content, along with summarization:

```
python3.11 preprocessing.py
```

This will:

- Process all manga in the `manga_images` directory
- Generate JSON schema files in `manga_analyses`
- Display progress as each page is analyzed

6.4.2 Step 2: Vectorizing Content. Run the vectorization script to create embeddings:

```
python3.11 vectorize.py
```

This will:

- Load JSON schema files from `manga_analyses`
- Create vector embeddings at book, page, and panel levels
- Store embeddings in ChromaDB for retrieval

6.4.3 Step 3: Querying. Run the query interface to search for manga:

```
python3.11 query.py
```

When prompted:

- Enter a natural language description of what you remember
- Optionally filter by level (book, page, or panel)
- View ranked results with explanations

Figure 5 illustrates a query usage example.

6.4.4 Evaluation Mode. To evaluate system performance with test queries:

```
python3.11 eval.py
```

This will:

- Load test queries from `testset/test_query.json`
- Run each query against the system
- Calculate accuracy metrics
- Display detailed results

6.5 Troubleshooting

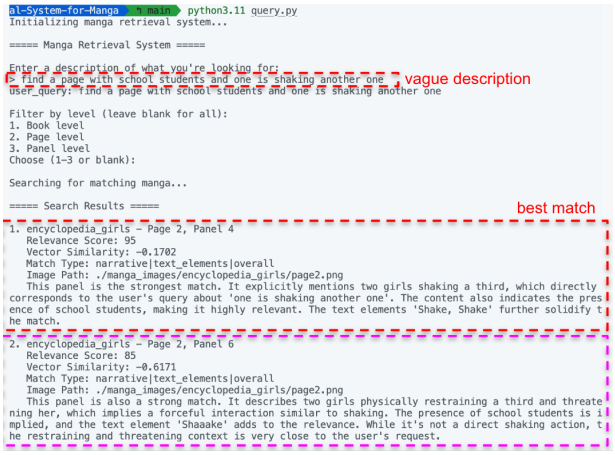
- **API Key Issues**: Ensure your API key is valid and has access to Gemini API. Check the `.env` file format.
- **Image Format Errors**: If preprocessing fails for certain images, convert them to PNG or JPEG format and try again.
- **Memory Errors**: For large manga collections, process them in smaller batches by temporarily moving some manga folders out of the `manga_images` directory.
- **ChromaDB Errors**: If you encounter database errors, try deleting the `_chroma` directory and re-running vectorization.

7 Limitations and Future Work

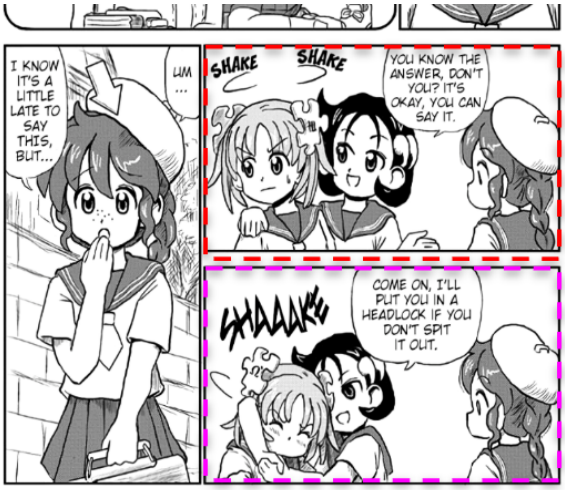
7.1 Limitations

Several limitations were identified during system development and evaluation:

- **Long-context Processing:** The system struggles with longer manga (exceeding 100 pages) due to LLM context window limitations
- **Preprocessing Efficiency:** Content analysis is time-consuming, limiting scalability for large collections
- **Summary Quality:** The quality of retrieval is directly dependent on summary quality, which can vary based on manga complexity



(a) Query interface showing user input and system output with ranked results (two out of five are shown). Each result includes manga title, page/panel information, relevance score, and explanation.



(b) The matching manga panel corresponding to the result.

Figure 5: Query process and results: (a) shows the search interface with query and ranked results, and (b) shows the actual manga panel that was retrieved as the top match.

- **API Dependency:** The system requires an internet connection and API key for Gemini-2.0-flash, which may incur costs for large collections

7.2 Future Work

To address these limitations and enhance the system, we plan to:

- Implement sliding window techniques for processing extremely long manga
- Develop parallel processing capabilities for more efficient page summarization

- Improve data structures and prompting strategies to enhance summary quality
- Explore visual-textual joint embeddings to better capture the unique aspects of manga content
- Add personalization capabilities to adapt to individual users' memory and query patterns
- Implement a web-based user interface for easier interaction

8 Conclusion

Semantic Manga RAG adds a search mode not handled by existing manga tools by enabling retrieval based on natural language descriptions of vague memories. By combining multimodal content understanding with vector-based retrieval and LLM re-ranking, the system often matches vague user memory with relevant manga content.

Our evaluation demonstrates promising results, with a Top-K accuracy of nearly 80%, confirming the viability of this approach for memory-based manga retrieval. The system not only helps users rediscover manga from faint memories but also provides explanations that connect their queries to the results, enhancing the search experience.

The approach presented in this paper could be extended to other visual-textual media types, such as comics, illustrated books, or even video key-frames.

Acknowledgments

We thank the CS510 professor and TAs for support and resources that made this project possible.

References

[1] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS '20). Curran Associates Inc., Red Hook, NY, USA, Article 793, 9459–9474.