International Journal of Internet, Broadcasting and Communication Vol.17 No.2 223-231 (2025)
http://dx.doi.org/10.7236/IJIBC.2025.17.2.223

IJIBC 25-2-24

Implementation Details of EPUB Reader using GraphRAG

Jiyoon Ok, Juyeon Soung, Chaewon Park, Kitae Hwang*

Student, Department of Computer Engineering, Hansung University, Korea
** Professor, Department of Computer Engineering, Hansung University, Korea*

*2271522@hansung.ac.kr, tjdwndus1325@gmail.com, caley4@hansung.ac.kr, calafk@hansung.ac.kr**

Abstract

The GraphRag technique has recently been studied meaningfully as a technique for securing high performance in search and inference without hallucination for domain-specific knowledge. GraphRAG builds a graph of the document's core concepts and the relationships between them, resulting in a connected knowledge network. To validate the practicality of GraphRAG, we implemented an EPUB reader using GraphRAG in our previous research and evaluated its retrieval performance, achieving 90% accuracy in both factual retrieval tasks and complex inferential queries, demonstrating superior performance. In this paper, we describe details of the implementation of the EPUB reader capable of retrieval and inference using the GraphRAG technique. Specifically, we detail the processes of building a graph database from EPUB files, conducting retrieval, and visualizing the constructed graphs. Furthermore, we explain how the EPUB reader was designed to run on resource-constrained devices such as the Raspberry Pi 5 single-board computer. We expected that this system serves as a representative implementation example for applications leveraging GraphRAG technology.

Keywords: *GraphRAG, Knowledge Graph, LLM, Epub Reader*

1. Introduction

Recently, large language models (LLMs) have revolutionized retrieval methods by generating context-aware responses to natural language queries. However, LLMs often suffer from hallucination problems, generating inaccurate responses to queries about new information or domain-specific knowledge that are not included in their training data [1]. To address these limitations, Retrieval-Augmented Generation (RAG) techniques have been introduced in recent works [2]. RAG leverages an LLM to embed documents into a vector database, and also segments user queries through the LLM to find similar information based on vector similarity. However, RAG relies on word-level similarity to answer queries, which limits its ability to analyze relationships between concepts within documents.

Manuscript Received: March. 31, 2025 / Revised: April. 14, 2025 / Accepted: April. 27, 2025
Corresponding Author: calafk@hansung.ac.kr
* Professor, Department of Computer Engineering, Hansung University, Korea

To address these limitations of RAG, Microsoft proposed the Graph-based Retrieval-Augmented Generation (GraphRAG) technique [3]. GraphRAG constructs graph structures representing concepts and their interrelationships within documents, thereby building an interconnected knowledge network. Microsoft demonstrated through experiments that GraphRAG achieves superior performance compared to naive RAG in retrieval and inference tasks, providing more comprehensive answers. Recently, some research has explored the application of GraphRAG in specific domains. For instance, Wu et al. [4] proposed MedGraphRAG, a framework based on GraphRAG in the medical domain, and demonstrated improved accuracy in answering queries related to disease diagnosis and treatment. However, their study primarily focused on performance evaluations using benchmark datasets and did not provide details on actual system implementations. Xu et al. [5] applied GraphRAG to Q&A systems in the e-commerce customer service domain and demonstrated improved performance in certain cases, although they provided limited details about the implementation. To date, research on GraphRAG has primarily focused on demonstrating conceptual feasibility and evaluating performance using benchmark datasets, with limited examples of practical application implementations.

To validate the practicality of GraphRAG, our research team implemented an EPUB reader using GraphRAG in a preliminary study and evaluated its retrieval performance, achieving 90% accuracy in both factual retrieval tasks and complex inferential queries, demonstrating superior performance [6]. This paper provides details on the implementation of the EPUB reader developed in the previous study, serving as a meaningful contribution to the GraphRAG research field where practical application implementation examples remain limited. The EPUB reader developed by our team not only enables retrieval and inference based on GraphRAG but also includes voice-based question-answering functionality tailored specifically for digital book learning environments. Additionally, we implemented a video conferencing feature to support collaborative learning. Furthermore, the EPUB reader has been implemented to run not only in desktop environments but also on embedded systems. Specifically, we established an environment with two 7-inch touch displays attached to a Raspberry Pi 5 single-board computer, enabling touch input functionality in addition to all features available in desktop environments.

2. Implementation of Graph EPUB Reader

2.1 System Architecture

The proposed system is designed based on a client-server model and consists of a web-browser client and two servers, as shown in Figure 1. Server computer A hosts both the EPUB Viewer Server and the WebRTC Signaling Server. The EPUB Viewer Server handles browser connections, renders e-book content to the user's screen, and serves web pages equipped with UI functionality. The WebRTC Signaling Server facilitates real-time video conferencing among users connected via web browsers. Server computer B hosts the EPUB Management Server, which receives natural-language queries from users, analyzes the queries, and provides responses related to the corresponding e-book. Users view EPUB files and utilize question-Answering and video conferencing functionalities by their web browsers

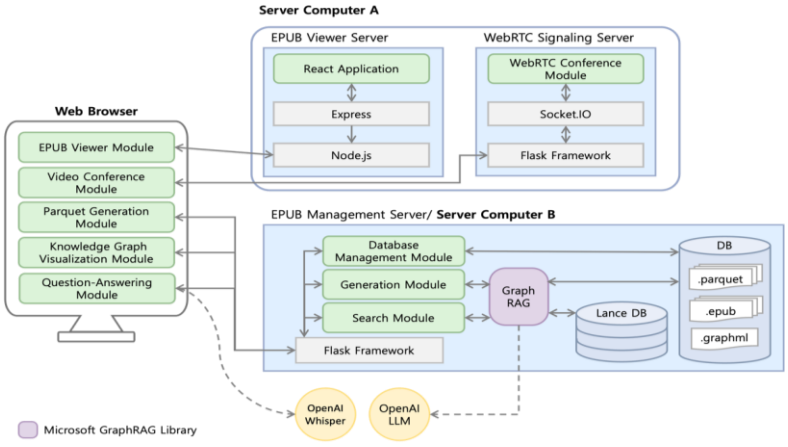

Figure 1. System Architecture of Graph EPUB Reader

The EPUB Viewer Server is built on Node.js, using Express.js as the web server framework. The web server application which handles client requests and responds by delivering web pages is implemented as a React-based application. This application bundles React modules that run in the user's web browser and serves them to the client. The React module comprises six distinct modules. The EPUB Viewer module retrieves EPUB files and renders the EPUB content in the web browser. The Video Conferencing module establishes a connection with the WebRTC Signaling Server to facilitate real-time video conferencing. The Parquet Generation module directs the EPUB Management Server to transform EPUB-formatted e-books into Parquet-formatted knowledge graphs. The Knowledge Graph Visualization module visualizes knowledge graphs in the browser. The Question-Answering module forwards user-generated natural language queries to the EPUB Management Server and presents the received responses in the browser.

The WebRTC Signaling Server is implemented based on the Flask framework and uses the Socket.IO library. It exchanges peer-to-peer connection data for real-time video conferencing, manages session control, and relays signaling messages between users. Two web browser clients perform real-time video conferencing through this server.

The EPUB Management Server which is also based on the Flask framework uses Microsoft's GraphRAG library and OpenAI's GPT-3.5-turbo and text-embedding-3-small models to conduct entity and relationship extraction, embedding generation, knowledge graph construction, and vector storage in LanceDB. Additionally, it provides retrieval and inference results utilizing GraphRAG in response to natural language queries from users.

2.2 Generation of Knowledge Graphs from EPUB Files

The implemented system uses Microsoft's GraphRAG library to transform textual EPUB files into structured knowledge graphs. Table 1 shows the six-stage operation of Microsoft's library using the fairy tale "Peter Pan" as an example.

Table 1. Generation of Knowledge Graph

Step	Example
Text Extraction and Chunking	"Chapter 1: Journey to Neverland. Once upon a time, Peter Pan lived in Neverland..." (1,200 characters)

Entity and Relationship Extraction	Entity: “Peter Pan”, “Neverland”, Relationship: “resides in”
Embedding Generation and Storage	Entity: “Peter Pan” → Embedding vector: [0.021, …, -0.031] Metadata: {“entity”: “Peter Pan”, “type”: “person”, “source_text”: “Chapter 1...”}
Graph Construction and Clustering	Entities are represented as nodes and relationships as edges to form a single unified graph, which is subsequently divided into multiple communities.
Community Analysis	“Peter Pan, Tinker Bell, and Wendy are companions sharing adventures in Neverland.”
Storage and Utilization	Parquet file

Upon uploading an EPUB file, the EPUB Management Server extracts only textual content. To enhance the contextual accuracy in retrieval and inference tasks, the extracted text is segmented into chunks of 1,200 characters each, with an overlap of 100 characters between consecutive chunks, as recommended by Microsoft. This overlap ensures sentence continuity across chunk boundaries.

Within each chunk, meaningful entities are identified and classified, and relationships among entities are extracted using OpenAI’s GPT-3.5-turbo model. At this stage, configuring custom prompts is crucial. To enhance the accuracy of entity and relationship extraction, the prompts should clearly specify appropriate entity types. In this study, for the sample EPUB file "Peter Pan," we specified four entity types—organization, person, geographic location (geo), and event—and restricted each extraction operation to a single entity type. Extracted entities are represented as words, while relationships are expressed in one or more sentences.

The extracted entities and relationships are embedded into vectors comprising 1,536-dimensional floating-point numbers using OpenAI’s text-embedding-3-small model. These embedded vectors are stored in LanceDB. All extracted entities and relationships are integrated into a single graph using the NetworkX library [7]. In this graph, entities are represented as nodes, relationships as edges, and each edge records a weight that indicates the strength of the relationship. This process is carried out using an LLM. To facilitate efficient and accurate graph construction, the Leiden algorithm was applied to group entities with strong mutual relevance into communities within the graph. The number of entities constituting each community can be determined based on the current graph size and specified by the administrator. In this study, the Leiden algorithm was configured to construct communities containing up to 10 entities. Each community is assigned a unique identifier, and this identifier is recorded as metadata within each node (entity).

Subsequently, community analysis is performed to extract and summarize the core semantic themes of each community. For instance, a community containing "Peter Pan" and "Tinker Bell" may be summarized as "companions sharing adventures in Neverland.". Finally, the generated knowledge graph—including entity information, relationship data, community summaries, and text chunks—is stored as individual Parquet files [8], each corresponding to distinct content. Concurrently, a GraphML file is generated and stored to facilitate knowledge graph visualization within the application.

2.3 Knowledge Graph Visualization

As shown in Figure 2, Knowledge graph visualization is the process of rendering a knowledge graph within the browser interface. Figure 2(a) and Figure 2(b) shows descriptions about Peter Pan and Hook respectively when the user moves the mouse over each entity. The proposed system enables users to interactively explore

entities and their relationships by navigating the visualized knowledge graph using a mouse. The visualization procedure involves loading the stored GraphML file, transforming it into JSON format, and subsequently rendering it on the web page utilizing the D3.js library.

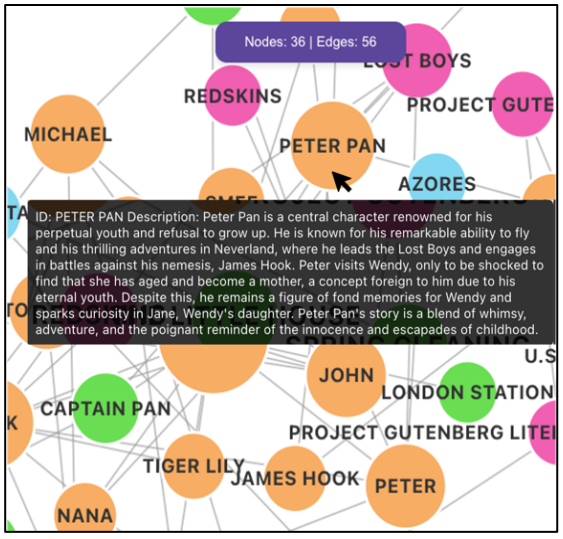

Figure 2 (a). Peter Pan in Knowledge Graph

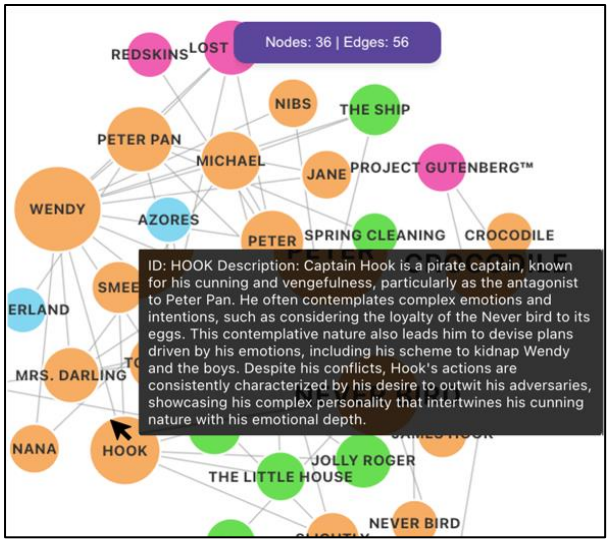

Figure 2 (b). Hook in Knowledge Graph

Table 2 shows examples of GraphML and corresponding JSON representations. In the graph, entities are represented as nodes and relationships as edges, both expressed within JSON arrays labeled "nodes" and "edges," respectively.

Table 2. Comparison of GraphML and JSON representations for the Hook node and the Hook-Wendy relationship edge

GraphML	JSON
<node id="HOOK"> <data key="d0">PERSON</data> <data key="d1">Captain Hook is a pirate captain, known for his cunning and vengefulness, particularly as the...</data> <data key="d2">503485ae5e8...</data> <data key="d3">2</data> <data key="d4">0</data> <data key="d5">8</data> <data key="d6">2</data> </node>	{ "id": "HOOK", "type": "PERSON", "description": "Captain Hook is a pirate captain, known for his cunning and vengefulness, particularly as the…", "source_id": "503485ae5e8674ea5073c507a269f571...", "community": "2", "level": 0, "degree": 8, "human_readable_id": 2, },
<edge source="HOOK" target="WENDY"> <data key="d8">9.0</data> <data key="d9">Hook plans to make Wendy	{ "source": "HOOK", "target": "WENDY", "weight": 9.0,

```
the mother of the pirates by kidnapping her</data>
  <data key="d10">f6a9c779a27d...</data>
  <data key="d11">9851d49183d...</data>
  <data key="d12">4</data>
  <data key="d13">0</data>
</edge>
```

```
  "description": "Hook plans to make Wendy the mother of the pirates by kidnapping her",
  "source_id": "f6a9c779a27d841a3473e845379db1e5",
  "id": "9851d49183d74c97a2c85377b8280663",
  "human_readable_id": 4,
  "level": 0,
},
```

Specifically, our system uses the D3.js library, which uses SVG to visualize knowledge graphs within web pages. Node and edge data are extracted from JSON files and passed to D3.js, which renders nodes as circles and edges as connecting lines. The relative importance of each entity is also provided to dynamically scale the size of the node circles. To ensure a clear layout, D3.js applies a force-directed graph algorithm that distributes nodes evenly and reduces edge overlap. Node colors are distinguished by entity type—such as person, location, organization, and event—enabling users to intuitively identify each entity's category. These types are predefined in the prompts used during entity extraction from EPUB files. Furthermore, D3.js supports interactive features such as zooming, panning, node dragging, and tooltip display, allowing users to easily explore and understand complex relationships within the document.

3. Search

The implemented system utilizes two search modes provided by Microsoft's GraphRAG library: Local Search and Global Search. Local search is a method of providing contextual information to LLM by targeting specific entities and their surrounding relationships that are directly related to the query. In contrast, Global search extracts and analyzes a comprehensive context targeting the entire dataset rather than being limited to a specific entity.

The search process begins when a user enters a natural language query through the web browser. The Flask server in the EPUB Management Server receives the query and launches a separate process to handle it. The query is then converted into a 1,536-dimensional vector using OpenAI's text-embedding-3-small model. This vector is compared with entity, text unit, and community vectors stored in the LanceDB to identify the most relevant items. These selected items are assembled into a single context, combined with a predefined prompt, and passed to the LLM to generate a response. The generated answer is then refined using regular expressions and delivered back to the user's web browser.

4. Speech Recognition and Video conferencing

Figure 3(a) shows that the implemented system incorporates speech recognition capabilities, enabling users to submit natural-language queries without text input. When a user makes a voice query, the user's voice is recorded using the MediaRecorder API, and the recorded audio data is converted into text format using OpenAI's Whisper-1 model. The transcribed text is provided as input to the existing question-answering pipeline.

Figure 3(b) shows that the video conferencing functionality has been implemented using WebRTC technology to support real-time discussions and collaborative learning among users. Peer-to-peer (P2P) connections between users are established with the help of Google's STUN server, ensuring reliable connectivity even in Network Address Translation (NAT) environments. Connection state management is

handled via a signaling server built with the Flask framework, where each user is identified by a unique ID and grouped into virtual "rooms." Furthermore, the system employs the Socket.IO library to ensure stable real-time communication among users.

Figure 2(a). Speech Recognition

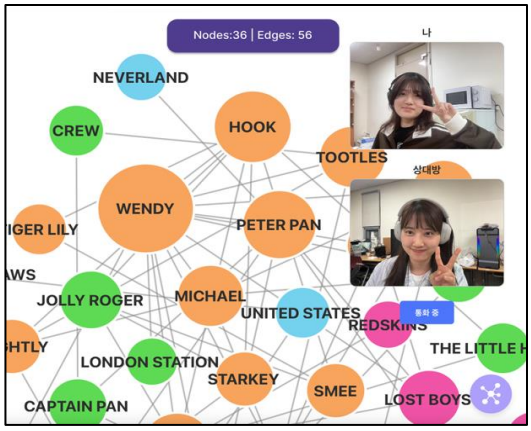

Figure 3(b). Video Conferencing

5. Implementation of EPUB Reader using Raspberry Pi 5

5.1 Hardware Configuration

This section presents the implementation of an EPUB reader using the Raspberry Pi 5 single-board computer [9][10]. As shown in Figure 4, an 8GB Raspberry Pi 5 running the Linux operating system was connected to two 7-inch HD resolution touch displays. The two displays were connected to the Raspberry Pi 5's Micro HDMI ports for video output. We tried connecting two touch displays to each of the Raspberry Pi 5's two USB ports for both touch input and power supply. However, Raspberry Pi 5 is a single-board computer with power limitations that prevent it from powering two external displays. To address this power supply constraint, as shown in Figure 4, a separate USB hub was employed to provide power to both displays. This USB hub was connected to a USB port on the Raspberry Pi 5, thereby resolving the aforementioned power limitation.

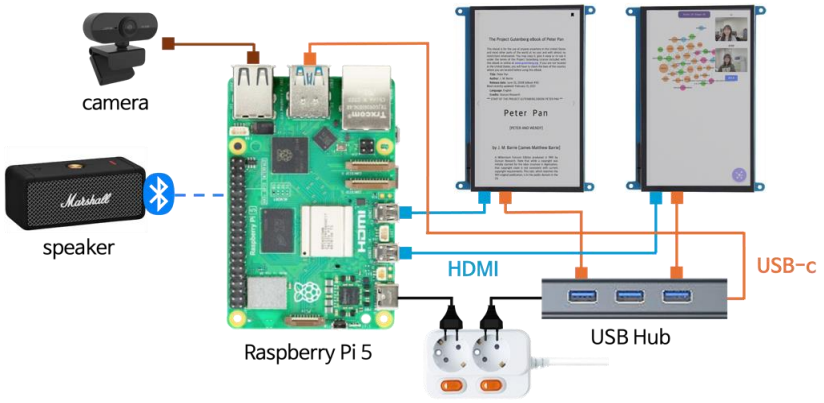

Figure 4. EPUB Reader Hardware

5.2 Display Operation and EPUB Reader Implementation

We implemented the Raspberry Pi OS to recognize two touch displays as one continuous screen, display them in full screen, and process input from each touch display independently. However, the default Linux display server, Wayland, computes touch input coordinates based on the combined resolution of the two displays, which leads to inaccurate input mapping in multi-display configurations. So, we resolved this issue by changing the Raspberry Pi5's configuration from Wayland to Xorg. The Xorg display server can map input devices to specific screens individually, so input from each touch display is handled accurately. Also Wayland cannot recognize two touch displays as one continuous screen, so it cannot implement full screen display, but Xorg recognizes both displays as one continuous screen, while processing each display input independently.

To provide a consistent user experience within an embedded system environment, we utilized the Electron framework. Electron which is a desktop application development framework based on Chromium and Node.js provides the functionality to transform web-based React applications into standalone desktop applications. Considering the limited hardware resources of the Raspberry Pi, the application was designed to use the entire screen as a GUI without running a separate web browser. Also, by connecting to the existing server using the HTTP protocol, the server-client communication structure of the desktop is maintained as is.

6. Conclusion

In this paper, we presented the detailed implementation of an EPUB reader using GraphRAG to verify the practicality of GraphRAG. It provided in-depth descriptions of the processes involved in constructing a graph database from EPUB files, performing search operations, and visualizing the knowledge graph. Furthermore, it outlines the implementation of voice-based querying and a video conferencing feature that enables collaborative e-book reading between users. In particular, we described how the EPUB reader was implemented on an embedded system environment using a Raspberry Pi 5 single-board computer equipped with two 7-inch touch displays. This implementation preserves all functionalities available in desktop environments while additionally supporting touch input. Finally, we conclude that the system described in this paper is expected to serve as a valuable reference for researchers aiming to develop systems with retrieval and reasoning capabilities using GraphRAG technology.

Acknowledgement

This research was financially supported by the Hansung University

References

[1] G. Perković, A. Drobnjak, and I. Botički, “Hallucinations in LLMs: Understanding and Addressing Challenges,” in Proc. 2024 47th MIPRO ICT and Electronics Convention (MIPRO), Opatija, Croatia, pp. 2084–2088, 2024. DOI: https://doi.org/10.1109/MIPRO60963.2024.10569238

[2] Patrick Lewis, et al., “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” arXiv preprint arXiv:2005.11401, 2020. DOI: https://doi.org/10.48550/arXiv.2005.11401

[3] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, et al., “From local to global: A graph RAG approach to query-focused summarization,” arXiv preprint arXiv:2404.16130, 2024.

DOI: https://doi.org/10.48550/arXiv.2404.16130

[4] J. Wu, J. Zhu, Y. Qi, J. Chen, M. Xu, F. Menolascina, and V. Grau, “Medical Graph RAG: Towards safe medical large language model via graph retrieval-augmented generation,” arXiv preprint arXiv:2408.04187, 2024.
DOI: https://doi.org/10.48550/arXiv.2408.04187

[5] Zhentao Xu, Mark Jerome Cruz, Matthew Guevara, Tie Wang, Manasi Deshpande, Xiaofeng Wang, Zheng Li, "Retrieval-Augmented Generation with Knowledge Graphs for Customer Service Question Answering," Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 2905-2909, July 2024.
DOI: https://doi.org/10.1145/3626772.3661370

[6] C. Park, J. Ok, J. Han, J. Soung, and K. Hwang, “EPUB Reader with Advanced Search and Inference Functions using GraphRAG,” The Journal of The Institute of Internet, Broadcasting and Communication (JIIBC), Vol. 24, No. 6, pp. 29–35, 2024.
DOI: https://doi.org/10.7236/JIIBC.2024.24.6.29

[7] NetworkX. https://networkx.org/.

[8] Apache Parquet. https://parquet.apache.org/

[9] Kitae Hwang, et al., “Implementation of CoMirror System with Video Call and Messaging Function between Smart Mirrors“, The Journal of The Institute of Internet, Broadcasting and Communication (JIIBC), Vol.22, No. 6, pp.121-127, 2022.
DOI: https://doi.org/10.7236/JIIBC.2022.22.6.121

[10] K. Hwang et al., “Performance Evaluation of CoMirror System with Video Call and Messaging Function between Smart Mirrors,” The Journal of The Institute of Internet, Broadcasting and Communication, vol. 23, no. 3, pp. 51–57, 2023.
DOI: https://doi.org/10.7236/JIIBC.2023.23.3.51