ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

LUẬN VĂN THẠC SĨ

Ứng dụng Graph RAG vào hệ thống Q&A trong lĩnh vực giáo dục

GVHD: PGS. TS. Quản Thành Thơ
HVCH: Lê Nguyên Khang - 2370618

Thành phố Hồ Chí Minh, 05 2025

_____ Ngày: _____

PGS. TS. Quản Thành Thơ (GV hướng dẫn)

Khoa Khoa học và Kỹ thuật Máy tính

Lời cam đoan

Tôi xin cam đoan rằng, bài báo cáo Luận văn Thạc sĩ 'Ứng dụng Graph RAG vào hệ thống Q&A trong lĩnh vực giáo dục' là sản phẩm nghiên cứu của tôi dưới sự hướng dẫn của thầy PGS. TS. Quản Thành Thơ, chú trọng vào việc giải quyết thách thức thực tiễn trong truy vấn dữ liệu cho Chatbot hỏi đáp closed-domain.

Ngoại trừ những thông tin tham khảo rõ ràng từ các công trình nghiên cứu khác, các nội dung trong luận văn là kết quả của quá trình nghiên cứu chủ thể của tôi và chưa từng được công bố trước đây dưới mọi hình thức.

Tôi chấp nhận hoàn toàn trách nhiệm về nội dung và chất lượng của luận văn. Mọi sáng tạo và kết quả đều xuất phát từ công lao và sự cố gắng không ngừng của chính tôi. Trong trường hợp có bất kỳ sự đạo văn hay vi phạm bản quyền nào, tôi xác nhận sẽ chịu trách nhiệm và đảm bảo sửa chữa ngay lập tức.

Tôi cam kết tuân thủ nguyên tắc đạo đức nghiên cứu và tuân thủ các quy định của Trường Đại học Bách khoa - Đại học Quốc gia TP.HCM. Bản cam kết này không làm ảnh hưởng đến uy tín của trường và không tạo ra bất kỳ vấn đề pháp lý nào liên quan đến việc sử dụng thông tin hay kết quả nghiên cứu của tôi.

TP Hồ Chí Minh, Tháng 05/2025

Tác giả

Lê Nguyên Khang

Lời cảm ơn

Tôi xin gửi lời cảm ơn chân thành và tri ân nhất đến thầy PGS.TS Quản Thành Thơ, người đã dành thời gian và tâm huyết hướng dẫn tôi trong quá trình thực hiện đề tài. Sự đồng hành và sự tận tâm chỉ dẫn của thầy không chỉ giúp tôi có một cái nhìn toàn diện hơn về đề tài mà còn nâng cao chất lượng của công trình nghiên cứu.

Tôi muốn bày tỏ lòng biết ơn sâu sắc đến tất cả các thầy, cô và giảng viên Khoa Khoa học và Kỹ thuật Máy tính cũng như Trường Đại học Bách Khoa - Đại học Quốc gia Thành phố Hồ Chí Minh. Kiến thức quý báu mà tôi đã được học từ các thầy, cô đã đóng góp quan trọng vào việc hoàn thành đề tài và phát triển năng lực chuyên môn.

Mặc dù đã cố gắng hết sức để hoàn thiện đề tài, tôi nhận thức rằng vẫn còn những hạn chế và thiếu sót. Tôi mong muốn nhận được những lời nhận xét, góp ý từ thầy cô và bạn bè để bài báo cáo này có thể ngày càng được hoàn thiện và phát triển.

Tóm tắt

Hiện nay, với sự tiến bộ của các kỹ thuật Trí tuệ nhân tạo (Artificial Intelligence), sự phát triển của các hệ thống Chatbot thông minh ngày càng thu hút sự chú ý, đặc biệt là với tính hiệu quả của chúng trong việc thay thế con người ở nhiều lĩnh vực. Trong bối cảnh xu hướng hiện nay, người dùng có xu hướng ưa chuộng sự sử dụng ngôn ngữ tự nhiên bởi tính thân thiện và tính dễ sử dụng của nó. Tuy nhiên, xử lý dữ liệu để tạo câu trả lời là một bài toán cần giải quyết trong các hệ thống Chatbot.

Retrieval-Augmented Generation (RAG) được sinh ra với mục đích kết hợp hai khả năng mạnh mẽ trong xử lý ngôn ngữ tự nhiên: truy vấn thông tin và tạo sinh câu trả lời. Mục tiêu chính của RAG là cải thiện chất lượng và độ chính xác của câu trả lời trong các hệ thống Q&A hoặc các tác vụ tương tự, đặc biệt khi phải làm việc với dữ liệu lớn và phức tạp.

Đề tài này tập trung vào công việc tìm giải pháp để giải quyết một số vấn đề phát sinh khi áp dụng RAG như truy vấn thông tin từ nhiều nguồn, xử lý mối quan hệ giữa các đoạn văn bản hay các từ khóa không liên quan đến các đoạn văn bản cần tìm. Việc tăng tốc độ xử lý, truy vấn cũng là một trong những thách thức quan trọng cần phải được quan tâm.

Mục lục

Danh sách hình vẽ

Danh sách bảng

Chương 1

Giới thiệu

Trong chương này, tác giả sẽ giới thiệu tổng quan về nội dung của đề tài cùng với mục tiêu đề ra trong quá trình thực hiện đề tài.

1.1 Đặt vấn đề

QA là một lĩnh vực nghiên cứu sôi động và nhiều hướng nghiên cứu. Nó là sự giao thoa kết hợp của Xử lý ngôn ngữ tự nhiên (NLP), Truy xuất thông tin (Information Retrieval - IR), Suy luận logic (Logical Reasoning), Biểu diễn tri thức (Knowledge Representation), Học máy máy (Machine Learning), Tìm kiếm Ngữ nghĩa (Semantic search).

CQAS là một hướng nghiên cứu quan trọng trong Xử lý ngôn ngữ tự nhiên (Natural Language Processing - NLP) và là hướng mở rộng hơn của bài toán QAS. Do đặc trưng linh hoạt của ngôn ngữ tự nhiên mà các câu hỏi rất không có cấu trúc đồng thời do sự mơ hồ trong chính câu hỏi có thể dẫn đến các câu trả lời sai, CQAS có thể giảm bớt sự nhập nhằng bằng cách đặt thêm một số câu hỏi phụ cho người hỏi để làm rõ hơn về ngữ cảnh. Cụ thể hơn, nhiệm vụ CQAS là xây dựng một phần mềm có thể tự trả lời một loạt câu hỏi bằng ngôn ngữ tự nhiên có tính liên kết với nhau xuất hiện trong một cuộc hội thoại. Nó cũng có thể duy trì một cuộc đối thoại mạch lạc và phù hợp với người dùng, thay vì chỉ cung cấp các câu trả lời đứt đoạn.

Một trong những công việc cần thiết cho các công tác nghiên cứu QA và CQAS là tăng cường khả năng xử lý các câu hỏi phức tạp và cải tải thiện chất lượng câu trả lời. Từ đó, kỹ thuật RAG được áp dụng nhằm kết hợp khả năng truy vấn thông tin từ các nguồn bên ngoài với khả năng sinh văn bản của các mô hình ngôn ngữ, tạo ra các câu trả lời chính xác và có tính ứng dụng cao hơn, đặc biệt trong các bài toán có dữ liệu lớn và phức tạp, giúp cải thiện hiệu quả và độ chính xác của các hệ thống Q&A hoặc các tác vụ xử lý ngôn ngữ tự nhiên khác.

Tuy nhiên khi sử dụng RAG, sẽ có một số khó khăn như sau:

- Các từ khóa trong câu hỏi chỉ liên quan tới một số đoạn văn bản nhất định. Và

trong mỗi đoạn văn bản, thì lại đề cập tới các đoạn văn bản khác nhau. Do đó, cần truy vấn được hết các tài liệu này mới có thể tạo thành câu trả lời.

- Từ khóa trong câu hỏi là là từ ngữ thông dụng, có thể không khớp với các đoạn văn bản.
- Vấn đề hiểu toàn diện dữ liệu.

Nhận thấy vấn đề trên, tôi quyết định thực hiện đề tài này nhằm tạo ra một giải pháp có khả năng tăng cường khả năng tổng hợp thông tin để tạo thành câu trả lời đầy đủ và chính xác hơn.

1.2 Phạm vi nghiên cứu

Trong bài nghiên cứu này, tôi sẽ thiết kế và xây dựng một giải pháp hướng tới giải quyết vấn đề như sau:

- Tôi đã thu thập và sử dụng dữ liệu cho đề tài từ các nguồn thông tin, các văn bản chính thống từ các trang web, hệ thống lưu trữ thuộc Trường Đại học Bách Khoa - Đại học Quốc gia Thành phố Hồ Chí Minh. Các dữ liệu dùng để thử nghiệm trong bài báo cáo này thuộc các thể loại như thủ tục hành chính, các bài viết giới thiệu, các quy chế, quy định, các văn bản học thuật,...
- Giải pháp cung cấp khả năng cần truy vấn các đoạn văn bản từ nhiều tài liệu khác nhau để tìm ra các câu trả lời phù hợp. Các đoạn văn bản này có thể là từ các nguồn khác nhau như văn bản luật, tài liệu hướng dẫn.
- Các từ khóa trong câu hỏi không nhất thiết phải khớp trực tiếp với các đoạn văn bản, và một đoạn văn bản (ví dụ: A) có thể liên quan đến những đoạn khác (ví dụ: B, C). Hệ thống cần khả năng tìm kiếm các mối quan hệ này để tổng hợp thông tin từ các nguồn khác nhau.
- Các từ khóa trong câu hỏi đôi khi có thể là những từ ngữ thông dụng, không trực tiếp khớp với các đoạn văn bản có chứa thông tin cần thiết. Điều này làm cho việc truy vấn trở nên khó khăn hơn, đặc biệt trong các lĩnh vực chuyên môn như giáo dục, quy chế, quy định,...

1.3 Tổng quan về báo cáo

Có tất cả 5 chương được trình bày trong bài cáo báo đồ án này:

- Chương 1: Giới thiệu tổng quan về nội dung đề tài, mục tiêu và các giai đoạn đặt ra của đề tài.

- Chương 2: Trình bày các kiến thức nền tảng được nghiên cứu và sẽ sử dụng trong đề tài.
- Chương 3: Giới thiệu một số công trình liên quan đến đề tài.
- Chương 4: Trình bày về giải pháp mà tôi đã nghiên cứu và thực hiện đề tài này và các kết quả thực nghiệm
- Chương 5: Tổng kết, đánh giá giải pháp và đề ra hướng phát triển cho các giai đoạn tiếp theo trong tương lai.

Chương 2

Kiến thức nền tảng

Trong chương này, tác giả sẽ trình bày các kiến thức nền tảng cần thiết để hướng đến xây dựng Graph RAG.

2.1 Large Language Model - LLM

"Large language model là một loại mô hình ngôn ngữ được đào tạo bằng cách sử dụng các kỹ thuật học sâu trên tập dữ liệu văn bản khổng lồ. Các mô hình này có khả năng tạo văn bản tương tự như con người và thực hiện các tác vụ xử lý ngôn ngữ tự nhiên khác nhau."

Một mô hình ngôn ngữ có thể có độ phức tạp khác nhau, từ các mô hình n-gram đơn giản đến các mô hình mạng mô phỏng hệ thần kinh của con người vô cùng phức tạp. Tuy nhiên, thuật ngữ Large language model" thường dùng để chỉ các mô hình sử dụng kỹ thuật học sâu và có số lượng tham số lớn, có thể từ hàng tỷ đến hàng nghìn tỷ. Những mô hình này có thể phát hiện các quy luật phức tạp trong ngôn ngữ và tạo ra các văn bản y hệt con người.

Kiến trúc của LLM chủ yếu bao gồm nhiều lớp mạng neural, như recurrent layers, feedforward layers, embedding layers, attention layers. Các lớp này hoạt động cùng nhau để xử lý văn bản đầu vào và tạo dự đoán đầu ra.

- Embedding layer là thành phần quan trọng trong các mạng học sâu, đặc biệt trong xử lý ngôn ngữ tự nhiên (NLP).

  "An embedding layer maps discrete tokens (e.g., words or characters) to continuous vectors in a fixed-dimensional space, enabling the model to learn meaningful semantic representations of the input data during training"[1, 2].

  Các vector ánh xạ này có thể học được mối quan hệ ngữ nghĩa giữa các từ và được huấn luyện cùng với các trọng số của mô hình.

  Embedding layer chuyển đổi từng từ trong văn bản đầu vào thành biểu diễn vector nhiều chiều (high-dimensional). Những vector này nắm bắt thông tin ngữ nghĩa và

cú pháp của từng đơn vị cấu tạo nên câu (từ hoặc token) và giúp mô hình hiểu được ngữ cảnh của văn bản.

- Feedforward layer là một thành phần cơ bản trong các mạng học sâu.

  "Feedforward layers are the basic building blocks of deep neural networks, where information flows in a unidirectional manner through successive layers of linear transformations followed by nonlinear activation functions"[3, 4].

  Feedforward layers gồm nhiều lớp được kết nối đầy đủ áp dụng các phép biến đổi phi tuyến tính cho các embedding vector đầu vào. Các lớp này giúp mô hình học các thông tin trừu tượng hơn từ văn bản đầu vào.

- Recurrent layers là thành phần thiết yếu trong mạng học sâu, đặc biệt trong xử lý dữ liệu chuỗi.

  "Recurrent layers are specialized neural network components designed for sequence modeling tasks, enabling the model to retain temporal dependencies by incorporating feedback connections that link the outputs of previous steps to the current computation"[5, 6].

  Recurrent layers của LLM được thiết kế để diễn giải thông tin từ văn bản đầu vào theo trình tự. Các lớp này duy trì trạng thái ẩn được cập nhật ở mỗi bước thời gian, cho phép mô hình nắm bắt được sự phụ thuộc giữa các từ trong câu.

- Attention layers là thành phần quan trọng trong các mạng học sâu, đặc biệt trong các tác vụ xử lý chuỗi dữ liệu như dịch máy và tóm tắt văn bản.

  "Attention layers are designed to allow the model to focus on different parts of the input sequence with varying attention weights, thus enabling it to capture complex dependencies within the data. This mechanism has proven highly effective in tasks like machine translation and text summarization."[7, 8, 9].

  Attention layers là một phần quan trọng khác của LLM, cho phép mô hình tập trung có chọn lọc vào các phần khác nhau của văn bản đầu vào. Cơ chế này giúp mô hình chú ý đến các phần có liên quan nhất của văn bản đầu vào và tạo ra các dự đoán chính xác hơn.

LLM học hỏi từ khối lượng dữ liệu khổng lồ, thường được xây dựng dựa trên những bộ dữ liệu đủ lớn để bao gồm gần như mọi thứ đã được xuất bản trên internet trong một khoảng thời gian dài. LLM được học từ một khối lượng rất lớn văn bản trước khi có thể ghi nhớ các quy luật và cấu trúc ngôn ngữ. Đây là nguyên nhân mấu chốt để LLM có thể hiểu và phản hồi theo ngữ cảnh một cách logic và mạch lạc.

Dưới đây là một số ví dụ về LLM trong thực tế:

- GPT-3 (Generative Pre-training Transformer 3) – Đây là một trong những Mô hình Ngôn ngữ Lớn lớn nhất được phát triển bởi OpenAI. Nó có 175 tỷ tham số và có thể thực hiện nhiều tác vụ, bao gồm tạo văn bản, dịch thuật và tóm tắt.
- BERT (Bidirectional Encoder Representations from Transformers) – Được phát triển bởi Google, BERT là một LLM phổ biến khác đã được đào tạo trên một kho dữ liệu văn bản khổng lồ. Nó có thể hiểu ngữ cảnh của một câu và tạo ra các câu trả lời có ý nghĩa cho các câu hỏi.
- XLNet – LLM này được phát triển bởi Đại học Carnegie Mellon và Google sử dụng một cách tiếp cận mới để lập mô hình ngôn ngữ được gọi là "permutation language modeling". Nó đạt được hiệu suất cao nhất trong các tác vụ ngôn ngữ, bao gồm tạo ngôn ngữ và trả lời câu hỏi.
- T5 (Text-to-Text Transfer Transformer) – T5, do Google phát triển, được đào tạo về nhiều tác vụ ngôn ngữ và có thể thực hiện chuyển đổi văn bản, như dịch văn bản sang ngôn ngữ khác, tạo bản tóm tắt và trả lời câu hỏi.
- RoBERTa (Robustly Optimized BERT Pretraining Approach) – Được phát triển bởi Facebook AI Research, RoBERTa là phiên bản BERT cải tiến, hoạt động tốt hơn trên một số tác vụ ngôn ngữ.

2.2 Retrieval-Augmented Generation - RAG

Retrieval-Augmented Generation - RAG [10] là một phương pháp kết hợp giữa truy vấn thông tin (retrieval) và tạo sinh (generation) trong các mô hình ngôn ngữ. Thay vì chỉ dựa vào các thông tin có sẵn trong mô hình, RAG tích hợp quá trình truy vấn từ một cơ sở dữ liệu bên ngoài (như một bộ dữ liệu văn bản) và sử dụng thông tin đó để hỗ trợ quá trình sinh câu trả lời, giúp cải thiện khả năng tổng hợp và phản hồi chính xác hơn.

Retriever (Truy xuất) trong RAG chịu trách nhiệm tìm kiếm và truy xuất các tài liệu có liên quan từ một kho dữ liệu lớn (thường là tập hợp các văn bản, tài liệu, hoặc câu hỏi và câu trả lời) dựa trên câu hỏi hoặc bối cảnh đầu vào. Mục tiêu là chọn ra một tập hợp con các tài liệu mà có thể giúp mô hình trả lời câu hỏi hoặc tạo nội dung chính xác và có liên quan.

Các kỹ thuật phổ biến cho **retriever** bao gồm:

- **TF-IDF** (Term Frequency-Inverse Document Frequency).
- **BM25**.
- **Embedding-based methods**: Sử dụng mạng nơ-ron để mã hóa tài liệu và câu hỏi thành các vector, sau đó sử dụng khoảng cách giữa các vector để đánh giá độ liên quan.

Generator (Tạo ra văn bản) Sau khi tài liệu có liên quan được truy xuất, phần generator sẽ sử dụng chúng để tạo ra văn bản đáp ứng yêu cầu, ví dụ như câu trả lời cho câu hỏi hoặc một đoạn văn bản mô tả. Các mô hình generator phổ biến hiện nay thường là các mô hình transformer như GPT, BART, hoặc T5. Chúng có khả năng sinh văn bản tự nhiên và mạch lạc từ những thông tin có sẵn.

Mô hình generator sẽ sử dụng thông tin từ phần retriever để bổ sung hoặc làm phong phú thêm các câu trả lời, đồng thời đảm bảo rằng văn bản đầu ra là hợp lý và mạch lạc.

Lợi ích của RAG:

- Cải thiện độ chính xác và tính đầy đủ của câu trả lời: Việc kết hợp thông tin từ cơ sở dữ liệu ngoài giúp RAG không chỉ dựa vào các thông tin được huấn luyện trong mô hình mà còn có thể sử dụng thông tin cập nhật từ môi trường bên ngoài, giúp tăng độ chính xác của kết quả sinh ra.

- Giảm thiểu khả năng lan man: Với việc truy vấn các tài liệu liên quan, mô hình RAG có thể giảm thiểu rủi ro sinh ra thông tin không liên quan hoặc sai lệch.

- Ứng dụng trong các nhiệm vụ yêu cầu tri thức ngoài mô hình (knowledge-intensive tasks): RAG rất hiệu quả cho các nhiệm vụ như trả lời câu hỏi, tóm tắt tài liệu, và tìm kiếm thông tin trong các lĩnh vực như y tế, luật, hoặc khoa học.

2.3 Thuật toán Leiden

Thuật toán Leiden[11] là một thuật toán được sử dụng để phát hiện cộng đồng (community detection) trong mạng (graph). Nó cải tiến thuật toán Louvain nổi tiếng bằng cách đảm bảo các cộng đồng kết quả thỏa mãn các thuộc tính mạnh hơn như tính kết nối.

Các bước cơ bản của thuật toán Leiden:

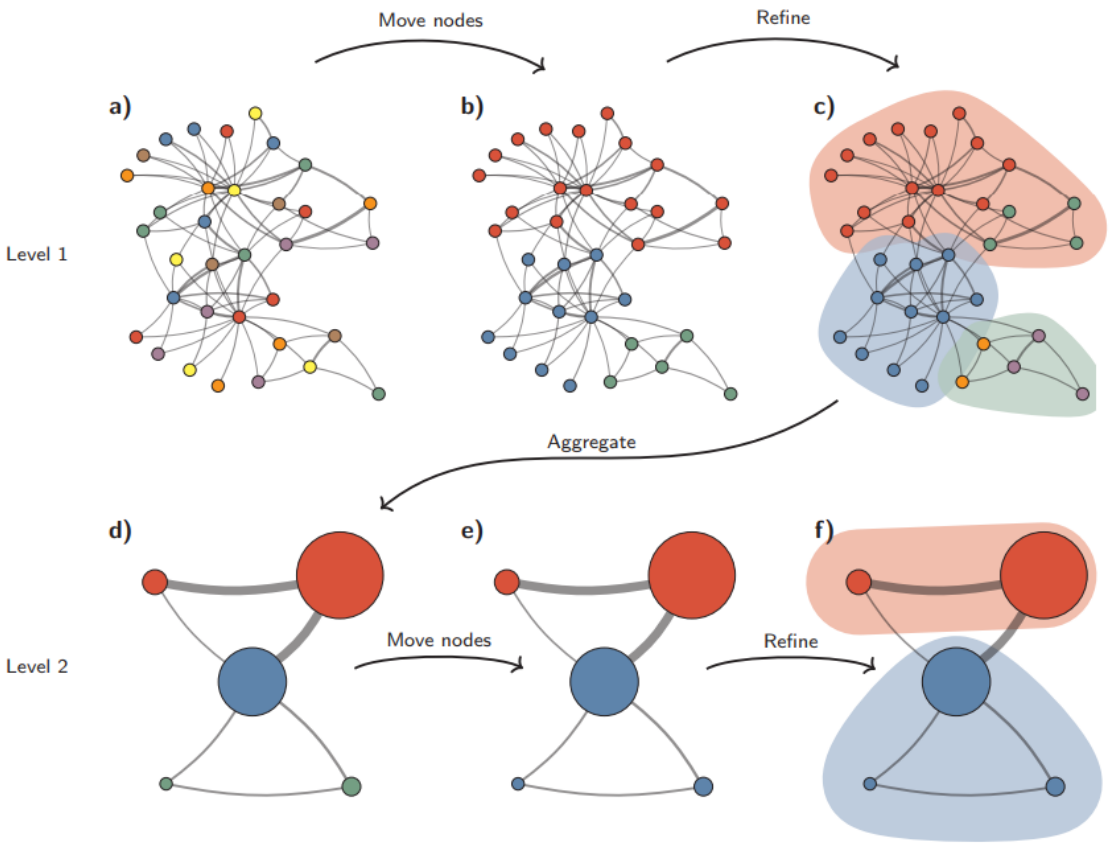
Bước 1: Tối ưu hóa cục bộ trong thuật toán Leiden

Bước đầu tiên của thuật toán Leiden là tối ưu hóa cục bộ các cộng đồng trong mạng để tăng giá trị hàm mục tiêu (thường là *modularity*). Cụ thể, quá trình này tìm cách di chuyển các node giữa các cộng đồng sao cho tổng modularity của mạng được cải thiện.

Chi tiết quá trình

1. **Khởi tạo**: Mỗi node trong mạng được gán vào một cộng đồng ngẫu nhiên hoặc theo một cách nào đó (ví dụ: theo các đặc trưng sẵn có).

2. **Tối ưu hóa modularity**: Mục tiêu là tối ưu hóa modularity (Q), chỉ số thể hiện chất lượng của phân chia cộng đồng. Modulariy đo lường sự phân chia của đồ thị thành các cộng đồng so với phân chia ngẫu nhiên. Cộng đồng có modularity cao thì các node trong cùng cộng đồng có nhiều kết nối với nhau hơn là với các node



Hình 2.1: Minh họa thuật toán Leiden

trong cộng đồng khác.

$$Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j) \tag{2.1}$$

Trong đó:

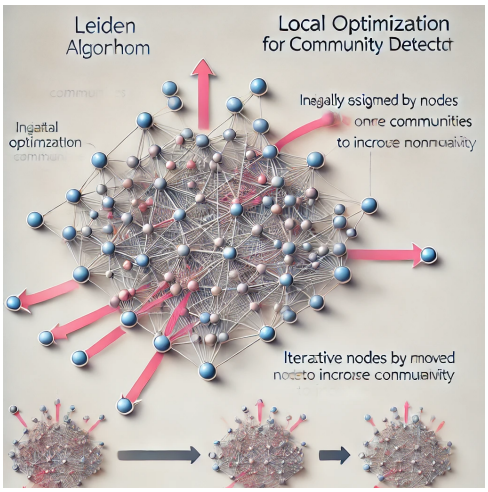
- A_{ij}: Trọng số cạnh giữa node i và j,
- k_i: Tổng trọng số cạnh liên quan đến node i,
- m: Tổng trọng số cạnh trong đồ thị,
- $\delta(c_i, c_j)$: Bằng 1 nếu i và j thuộc cùng một cộng đồng.

Mỗi node có thể được di chuyển giữa các cộng đồng lân cận của nó. Nếu việc di chuyển đó làm tăng giá trị của Q, node sẽ di chuyển vào cộng đồng đó.

3. **Di chuyển các node**: Quá trình này được lặp đi lặp lại cho đến khi không còn node nào có thể di chuyển mà không làm giảm Q.

4. **Lặp lại quá trình**: Sau khi hoàn tất quá trình tối ưu hóa cục bộ cho tất cả các node, thuật toán sẽ tiếp tục với các bước tiếp theo (phân chia tốt hơn, tạo mạng

rút gọn, v.v.), nhưng bước 1 này là trọng tâm trong việc xây dựng cộng đồng ban đầu.



Hình 2.2: Minh họa quá trình tối ưu hóa cục bộ trong thuật toán Leiden

Bước 2: Phân chia tốt hơn trong thuật toán Leiden

Sau khi hoàn tất bước tối ưu hóa cục bộ, thuật toán Leiden thực hiện bước "Phân chia tốt hơn"nhằm đảm bảo rằng các cộng đồng có tính kết nối mạnh mẽ hơn. Quá trình này giúp chia nhỏ các cộng đồng không kết nối hoặc không hoàn chỉnh, từ đó tạo ra các cộng đồng "tốt hơn"về mặt cấu trúc và liên kết.

Chi tiết quá trình

1. **Kiểm tra tính kết nối của cộng đồng**: Mỗi cộng đồng đã được xác định ở bước 1. Tuy nhiên, các cộng đồng có thể không được kết nối hoàn toàn, tức là một cộng đồng có thể chứa nhiều thành phần liên thông rời rạc (tức là các nhóm node trong cộng đồng đó không có đủ kết nối với nhau).

2. **Chia cộng đồng**: Nếu phát hiện một cộng đồng không có tính kết nối mạnh, thuật toán sẽ chia nó thành các nhóm con sao cho mỗi nhóm này là một thành phần liên thông. Quá trình chia này không làm giảm chất lượng phân chia cộng đồng, mà giúp các nhóm con có sự liên kết chặt chẽ hơn.

3. **Quá trình tái cấu trúc**: Sau khi phân chia các cộng đồng, thuật toán tiếp tục quá trình tối ưu hóa cục bộ (như đã mô tả ở bước 1) trên các cộng đồng mới này, nhằm đảm bảo rằng các cộng đồng nhỏ hơn được tối ưu hóa modularity trước khi tiến hành các bước tiếp theo.

4. **Lặp lại**: Quá trình này có thể được lặp lại cho đến khi tất cả các cộng đồng trong mạng đều đạt được tính kết nối tối ưu, và không có cộng đồng nào có thể bị chia nhỏ thêm.

Bước 3: Tạo mạng rút gọn trong thuật toán Leiden

Sau khi tối ưu hóa và phân chia cộng đồng, bước tiếp theo trong thuật toán Leiden là **tạo mạng rút gọn** (coarsening the graph). Mạng rút gọn giúp giảm kích thước của đồ thị gốc và làm cho quá trình phân tích cộng đồng trở nên hiệu quả hơn.

Chi tiết quá trình

1. **Đại diện cộng đồng là các node**: Mỗi cộng đồng đã được xác định trong các bước trước (tối ưu hóa cục bộ và phân chia tốt hơn) sẽ trở thành một node trong mạng mới. Các node trong cộng đồng được thay thế bằng một điểm duy nhất (node), giúp giảm kích thước của đồ thị.

2. **Kết nối giữa cộng đồng**: Các cộng đồng đã được thay thế bằng các node mới sẽ được kết nối với nhau. Trọng số của các cạnh giữa các cộng đồng sẽ được tính bằng tổng trọng số của tất cả các cạnh giữa các node trong cộng đồng đó. Công thức tính trọng số cạnh giữa hai cộng đồng C_1 và C_2 là:

$$w(C_1, C_2) = \sum_{i \in C_1, j \in C_2} A_{ij}$$

   Trong đó, A_{ij} là trọng số của cạnh giữa node i và j trong đồ thị ban đầu.

3. **Tạo mạng rút gọn**: Sau khi xác định các cộng đồng mới và các kết nối giữa chúng, thuật toán sẽ tạo ra một mạng con, trong đó mỗi cộng đồng là một node và các kết nối giữa các cộng đồng là các cạnh. Mạng này có kích thước nhỏ hơn và đơn giản hơn so với mạng gốc, giúp giảm độ phức tạp tính toán trong các bước tiếp theo.

4. **Lặp lại quá trình**: Thuật toán tiếp tục tối ưu hóa cộng đồng trên mạng rút gọn này bằng cách áp dụng lại bước tối ưu hóa cục bộ (như trong Bước 1) và phân chia cộng đồng (như trong Bước 2). Quá trình này có thể được lặp lại nhiều lần cho đến khi không còn cải thiện rõ rệt.

Bước 4. Lặp lại

Bước tối ưu hóa và tạo mạng rút gọn được lặp lại cho đến khi không còn sự thay đổi đáng kể trong cấu trúc cộng đồng.

2.4 Cơ sở dữ liệu Neo4j

Neo4j[12] là một cơ sở dữ liệu đồ thị dựa trên lý thuyết đồ thị, trong đó dữ liệu được lưu trữ bằng các đỉnh (nodes) và các cạnh (edges) [13]. Các nút đại diện cho các thực thể (ví dụ: người, sản phẩm, địa điểm), trong khi các cạnh biểu diễn các mối quan hệ giữa các thực thể đó.

Neo4j cho phép lưu trữ và truy vấn các mối quan hệ phức tạp một cách tự nhiên, khác với các hệ thống cơ sở dữ liệu quan hệ, vốn gặp khó khăn trong việc biểu diễn các quan hệ bậc cao. Điều này làm cho Neo4j đặc biệt phù hợp với các ứng dụng như: hệ thống đề xuất, mạng xã hội, công cụ tìm kiếm, và các hệ thống phân tích mạng.

Neo4j lưu trữ và trình bày dữ liệu dưới dạng biểu đồ, thay vì dạng bảng hoặc JSON như các hệ quản trị cơ sở dữ liệu truyền thống. Trong Neo4j, toàn bộ dữ liệu được biểu diễn bằng các **nút** và các **quan hệ** giữa các nút. Điều này làm cho Neo4j trở nên độc đáo so với các hệ thống quản lý cơ sở dữ liệu khác [13].

Cơ sở dữ liệu đồ thị:

- Các hệ quản trị cơ sở dữ liệu quan hệ như MS Access, SQL Server sử dụng bảng (gồm hàng và cột) để lưu trữ dữ liệu.
- Trong khi đó, Neo4j không sử dụng bảng, hàng hoặc cột mà sử dụng mô hình đồ thị.

Thành phần của cơ sở dữ liệu đồ thị:

- **Nút (Node)**: Đại diện cho các thực thể như con người, doanh nghiệp hoặc đối tượng dữ liệu.
- **Cạnh (Edge hoặc Relationship)**: Kết nối giữa các nút, thể hiện các mối quan hệ giữa các thực thể.
- **Thuộc tính (Property)**: Cung cấp thông tin bổ sung về các nút và mối quan hệ.

Cấu trúc này giúp Neo4j mô hình hóa các tình huống thực tế một cách tự nhiên và trực quan hơn so với cơ sở dữ liệu quan hệ truyền thống.

Tính năng nổi bật của Neo4j:

- **Hiệu suất cao và khả năng mở rộng**: Neo4j xử lý tốt khối lượng dữ liệu lớn và các truy vấn phức tạp. Công cụ lưu trữ và xử lý đồ thị gốc đảm bảo hiệu suất cao ngay cả với hàng tỷ nút và mối quan hệ.
- **Ngôn ngữ truy vấn Cypher**: Cypher là ngôn ngữ truy vấn mạnh mẽ và dễ sử dụng, giúp người dùng tạo, đọc, cập nhật và xóa dữ liệu dễ dàng với cú pháp ngắn gọn, dễ hiểu.
- **Tuân thủ ACID**: Neo4j đảm bảo tính toàn vẹn và độ tin cậy của dữ liệu thông qua việc tuân thủ các nguyên tắc ACID (Tính nguyên tử, Tính nhất quán, Tính cô lập và Tính bền vững).

- **Sơ đồ linh hoạt**: Cho phép thêm hoặc thay đổi mô hình dữ liệu mà không làm gián đoạn hoạt động của hệ thống, lý tưởng cho các môi trường kinh doanh thay đổi nhanh chóng.

Ưu điểm của Neo4j:

- Dễ dàng biểu diễn dữ liệu kết nối.
- Truy xuất, duyệt và điều hướng dữ liệu kết nối rất nhanh chóng.
- Sử dụng mô hình dữ liệu đơn giản nhưng mạnh mẽ.
- Hỗ trợ biểu diễn dữ liệu bán cấu trúc một cách tự nhiên.

Nhược điểm của Neo4j:

- Hỗ trợ cho OLAP (Online Analytical Processing) chưa được tối ưu tốt.
- Lĩnh vực cơ sở dữ liệu đồ thị vẫn đang trong quá trình nghiên cứu và phát triển, nhiều công nghệ liên quan còn chưa hoàn thiện.

2.5 Ngôn ngữ truy vấn Cypher

Cypher [14] là một ngôn ngữ truy vấn cơ sở dữ liệu đồ thị, với ngôn ngữ này chúng ta có thể tương tác như là truy vấn, cập nhật hay quản trị một cách hiệu quả với cơ sở dữ liệu đồ thị[15]. Ngôn ngữ này được thiết kế giúp cho developer cũng như các chuyên gia có thể thuận tiện khi làm việc với Neo4j. Cypher vốn được thiết kế đơn giản, tuy nhiên nó rất mạnh mẽ.

Cypher được lấy cảm hứng từ rất nhiều các cách tiếp cận khác nhau. Một số các từ khóa như `WHERE`, `ORDER BY` được lấy cảm hứng từ ngôn ngữ SQL, trong khi đó pattern matching thì lại được mượn từ SPARQL. Ngoài ra, một vài ngữ nghĩa được mượn từ các ngôn ngữ khác như Haskell và Python. Cấu trúc của Cypher được xây dựng dựa trên ngôn ngữ tiếng Anh với ngữ nghĩa thuận tiện cho người thao tác với ngôn ngữ, điều này giúp cho việc viết và đọc các câu truy vấn cũng dễ dàng hơn.

Cypher có cấu trúc tương tự như SQL. Các câu truy vấn được xây dựng từ nhiều mệnh đề khác nhau. Các mệnh đề là chuỗi liên kết với nhau. Dưới đây là một vài ví dụ sử dụng mệnh đề để đọc dữ liệu từ cơ sở dữ liệu đồ thị:

- **MATCH**: so khớp với các pattern phù hợp. Đây là cách thông dụng nhất để lấy dữ liệu từ graph.
- **WHERE**: không phải là một mệnh đề chính quy, nhưng nó là một phần của `MATCH`, `OPTIONAL MATCH` và `WITH`. `WHERE` sẽ thêm các ràng buộc vào pattern, hoặc sẽ lọc các kết quả có được thông qua `WITH`.

- **RETURN**: trả về kết quả.

Ví dụ:

```
MATCH (p:Person)-[:FRIEND]->(friend)
WHERE p.name = 'Alice'
RETURN friend.name
```

Giải thích:

- `MATCH (p:Person)-[:FRIEND]->(friend)`: Tìm tất cả các nút `friend` có mối quan hệ `FRIEND` đi ra từ một nút `p` mang nhãn `Person`.
- `WHERE p.name = 'Alice'`: Giới hạn chỉ chọn những nút `Person` có thuộc tính `name` bằng `Alice`.
- `RETURN friend.name`: Trả về tên của các nút bạn bè (`friend`) của `Alice`.

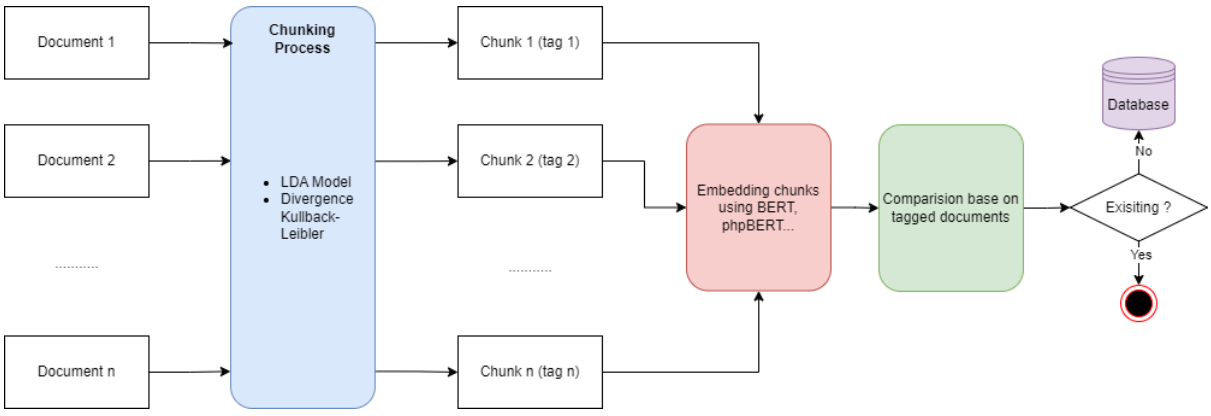
Chương 3

Công trình liên quan

Sau khi tiến hành tìm hiểu và nghiên cứu, tác giả đã tìm được một vài ứng dụng và kỹ thuật có tính ứng dụng và cách hoạt động có thể giúp ích cho hướng giải quyết vấn đề ban đầu.

3.1 Phân đoạn văn bản dựa trên sự thay đổi chủ đề

Để giải quyết tác vụ chia các bài viết dài thành các đoạn ngắn nhưng vẫn phải đảm bảo đầy đủ ngữ nghĩa, đồng thời xác định đoạn văn này có trùng trong cơ sở dữ liệu đã có hay không để đảm bảo việc lưu trữ một bản duy nhất sẽ giúp việc tìm kiếm và trả lời thông tin được hiệu quả, một cách tiếp cận được biểu diễn ở hình bên dưới.[16]



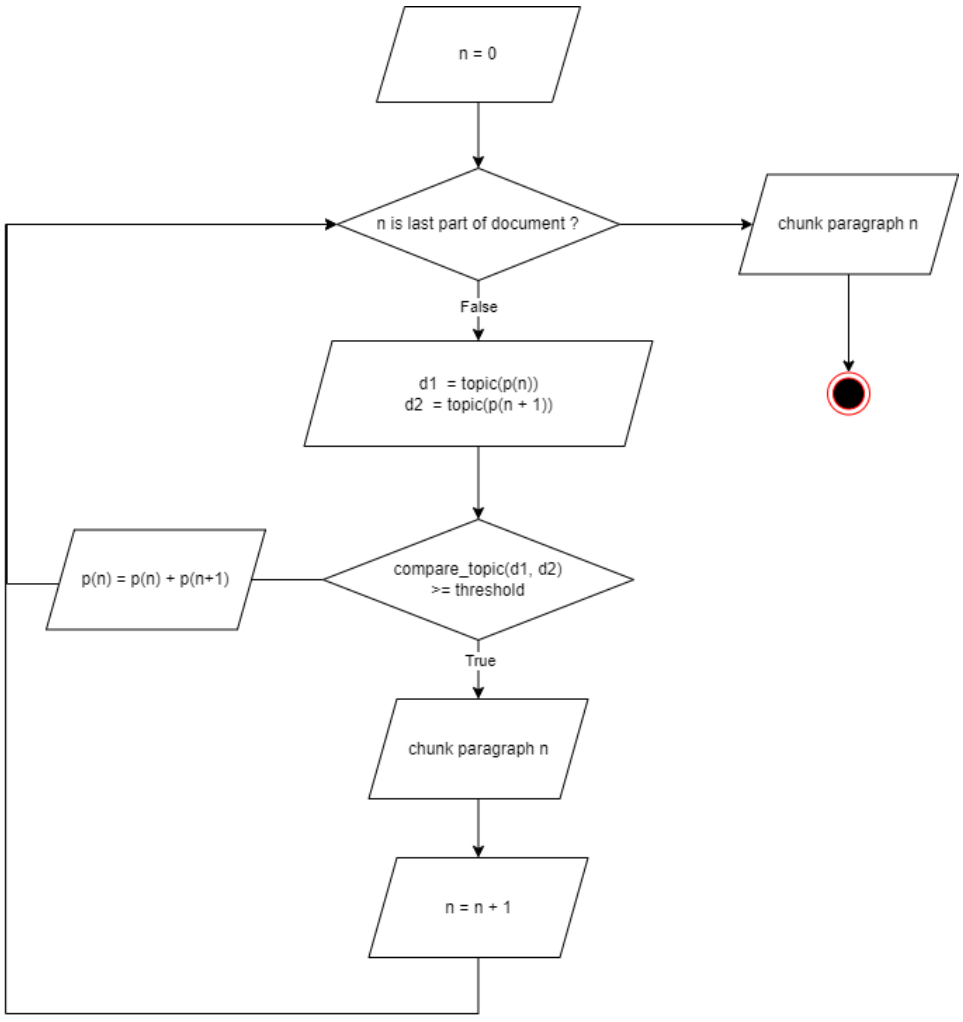
Hình 3.1: Chiến lược theo dõi sự thay đổi chủ đề văn bản

Đầu tiên, với một đoạn văn bản dài, phương pháp phân tách đoạn dựa trên sự thay đổi của chủ đề được sử dụng để chia văn bản thành các phần, các đoạn ngắn (chunk). Sau đó với mỗi đoạn văn thu được, tác giả tiến hành gán nhãn và embed đoạn văn thành vector. Cuối cùng, tác giả thực hiện công tác so sánh các vector vừa thu được với các vector có trong cơ sở dữ liệu dựa trên các nhãn tương ứng, và thực hiện lưu văn bản vào cơ sở dữ liệu nếu đoạn văn chưa tồn tại (về mặt ngữ nghĩa, nội dung).

Để tiến hành phân tách văn bản thành các đoạn văn ngắn dựa trên sự thay đổi chủ đề, tác giả sử dụng một cửa sổ trượt (slide window) để lần lượt trượt qua toàn bộ văn bản. Kích thước của cửa sổ trượt có thể là 2, 3, 4 ,... câu nhưng thường sẽ không có kích thước quá lớn. Thông thường, một đoạn văn tốt sẽ có kích thước từ 3 đến 10 câu[1].

Với mỗi phần văn bản được trượt qua, tác giả sẽ tiến hành đánh giá sự khác nhau giữa chủ đề giữa phần hiện tại và phần trước đó (hoặc các phần trước đó). Nếu chủ đề giữa hai phần có sự khác nhau (vượt ngưỡng chỉ định) thì tiến hành tách đoạn tại điểm hiện tại của cửa sổ trượt. Ngược lại, nếu chủ đề giữa hai phần không có sự khác biệt tương đối, phần văn bản hiện tại ở cửa sổ trượt sẽ được kết hợp với đoạn trước đó để so sánh với đoạn tiếp theo.

Giải thuật phân tách đoạn theo chủ đề được thể hiện ở hình bên dưới.



Hình 3.2: Giải thuật phân tách đoạn theo chủ đề

[1]Paragraphs - Writing Guide `https://www.usu.edu/markdamen/writingguide/15paragr.htm`

Trong đó:

- p(n) là phần văn bản thứ n được theo dõi bởi cửa sổ trượt.
- d(i) là phân phối chủ đề của văn bản thứ i.

Vì đơn vị của cửa sổ trượt là câu nên tác giả đã kết hợp với một số công cụ hỗ trợ để phân tách văn bản thành cách câu trong quá trình xử lý. Một số công cụ hỗ trợ đã được sử dụng như Underthesea Toolkit...

3.2 BERT

BERT (Bidirectional Encoder Representations from Transformers)[17] là một mô hình học sâu được phát triển bởi Google và được công bố vào năm 2018. BERT là một phần của họ mô hình Transformers, và nó đã tạo ra bước tiến lớn trong lĩnh vực xử lý ngôn ngữ tự nhiên (NLP).

Đặc điểm chính của BERT:

- **Bidirectional**: Không giống như các mô hình trước đó chỉ xử lý văn bản từ trái sang phải hoặc từ phải sang trái, BERT xử lý văn bản theo cả hai chiều. Điều này cho phép BERT nắm bắt được ngữ cảnh đầy đủ của một từ dựa trên cả văn bản trước và sau từ đó.
- **Pre-training và Fine-tuning**: BERT được huấn luyện trước (pre-trained) trên một lượng lớn dữ liệu văn bản từ Wikipedia và BooksCorpus, sau đó có thể được tinh chỉnh (fine-tuned) cho các nhiệm vụ cụ thể như phân loại văn bản, trả lời câu hỏi, và nhiều nhiệm vụ khác.
- **Transformers**: BERT dựa trên kiến trúc Transformer, một kiến trúc mạng neural mạnh mẽ cho phép mô hình xử lý mối quan hệ giữa các từ trong một câu mà không cần đến sự phụ thuộc tuần tự như các mô hình Recurrent Neural Network (RNN).

PhoBERT [18] là một mô hình ngôn ngữ dựa trên BERT được phát triển đặc biệt cho tiếng Việt. Giống như BERT, PhoBERT cũng sử dụng kiến trúc Transformer và kỹ thuật học sâu để nắm bắt ngữ cảnh của từ trong câu, nhưng nó được huấn luyện trên một lượng lớn dữ liệu tiếng Việt.

Đặc điểm chính của PhoBERT:

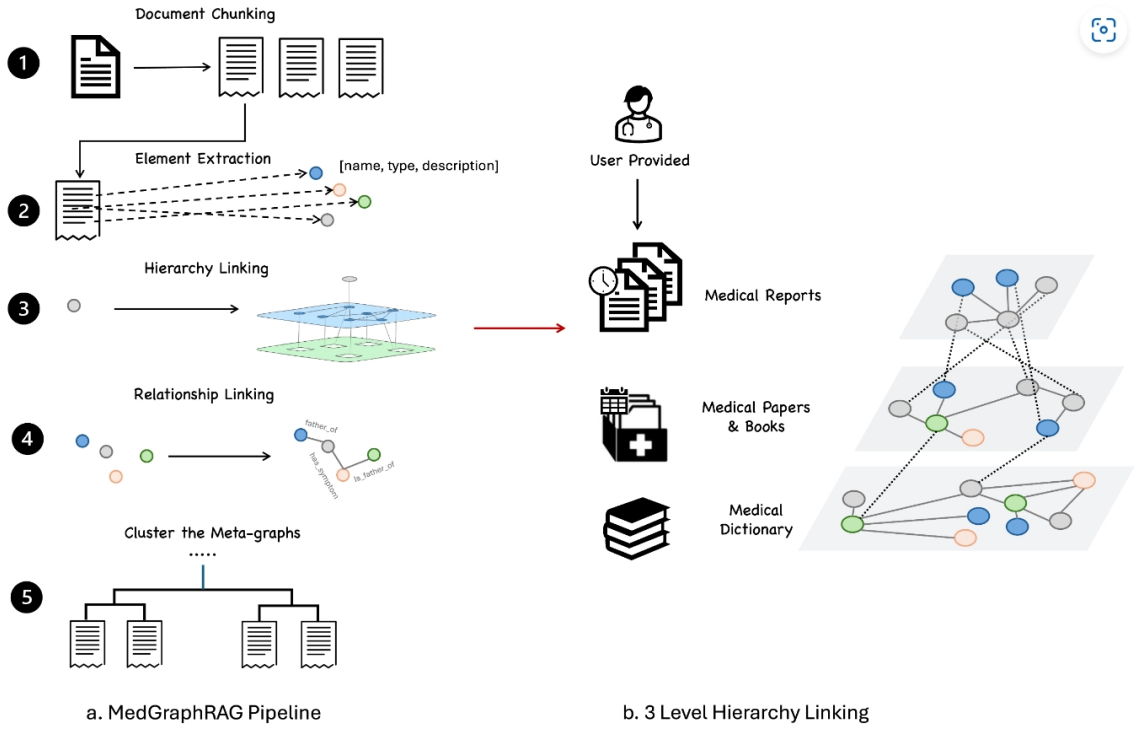
- **Ngôn ngữ đặc thù**: PhoBERT được phát triển riêng cho tiếng Việt, tận dụng các đặc trưng ngữ pháp và từ vựng của tiếng Việt để cải thiện hiệu suất cho các tác vụ NLP liên quan đến tiếng Việt.

- **Corpus huấn luyện**: PhoBERT được huấn luyện trên một lượng lớn dữ liệu văn bản tiếng Việt từ nhiều nguồn khác nhau như báo chí, sách, và các trang web. Điều này giúp mô hình hiểu rõ hơn về ngữ cảnh và ý nghĩa của từ trong tiếng Việt.
- **Kiến trúc**: PhoBERT sử dụng kiến trúc BERT cơ bản, bao gồm các tầng encoder của Transformer, giúp mô hình học được mối quan hệ giữa các từ trong câu theo cả hai chiều (trái sang phải và phải sang trái).

3.3 Medical Graph RAG

Phương pháp dựa trên đồ thị trong mô hình Retrieval-Augmented Generation (RAG) cho lĩnh vực y tế, gọi là **MedGraphRAG** [19].

3.3.1 Cấu trúc đồ thị ba tầng



Hình 3.3: Graph RAG pipeline

MedGraphRAG sử dụng một đồ thị phân cấp gồm ba tầng để liên kết các thực thể y tế với kiến thức cơ bản:

- **Cấp 1:** Tài liệu người dùng cung cấp (ví dụ: báo cáo y tế).
- **Cấp 2:** Kiến thức y học nền tảng từ sách và bài báo khoa học.

- **Cấp 3:** Thuật ngữ và quan hệ y học chuẩn hóa từ hệ thống UMLS (Unified Medical Language System).

Chiến lược U-retrieve kết hợp việc truy xuất thông tin theo cách từ trên xuống dưới và tạo câu trả lời từ dưới lên trên. Mô hình bắt đầu từ truy vấn của người dùng, phân loại theo các nhãn y tế và duyệt qua các đồ thị để tìm ra câu trả lời phù hợp. Thông tin được lấy từ các *meta-graph* — những nút đồ thị và mối quan hệ gần nhất với truy vấn — sau đó được tổng hợp thành một câu trả lời chi tiết.

3.3.2 Xây dựng Đồ thị Y tế

Phân đoạn tài liệu ngữ nghĩa : Các tài liệu y tế lớn thường chứa nhiều chủ đề khác nhau, cần phân chia tài liệu thành các đoạn nhỏ mà vẫn duy trì được ngữ cảnh.

- Kết hợp phương pháp phân đoạn ký tự (tách đoạn theo dấu ngắt dòng) với phân đoạn ngữ nghĩa (dựa trên chủ đề).
- Sử dụng kỹ thuật *proposition transfer* để trích xuất các phát biểu độc lập và quyết định cách nhóm chúng.
- Dùng cửa sổ trượt (*sliding window*) để xử lý 5 đoạn văn một lần, điều chỉnh cho phù hợp với giới hạn ngữ cảnh của LLM.

Trích xuất các phần tử (entities): Trích xuất thực thể từ văn bản đã phân đoạn, gồm tên, loại và mô tả.

- Sử dụng LLM để nhận diện thực thể y tế trong mỗi đoạn.
- Gán mỗi thực thể với một ID duy nhất để truy xuất nguồn gốc.
- Quá trình trích xuất lặp nhiều lần để đảm bảo đầy đủ.

Liên kết phân cấp (Hierarchy Linking): Duy trì tính chính xác và thuật ngữ y học chuẩn hóa.

- Xây dựng cấu trúc đồ thị ba tầng: tài liệu người dùng → kiến thức nền → thuật ngữ UMLS.
- So sánh thực thể với thuật ngữ bằng độ tương đồng ngữ nghĩa để liên kết chính xác.

Liên kết quan hệ (Relationship Linking): Xác định các mối quan hệ giữa thực thể trong đồ thị.

- Dùng LLM để xác định quan hệ dựa trên tên, mô tả, định nghĩa và kiến thức nền.
- Biểu diễn quan hệ dưới dạng đồ thị có hướng có trọng số (*weighted directed graph*), gọi là *meta-graph*.

3.3.3 Gộp đồ thị (Tags Generation and Merge)

- Xây dựng *meta-graph* cho từng phần dữ liệu.
- Dùng LLM tóm tắt các *meta-graph* theo danh mục y tế (triệu chứng, thuốc men, v.v.).
- Hợp nhất các *meta-graph* dựa trên độ tương đồng để tạo thực thể lớn hơn.
- Quá trình lặp lại tối đa 24 lần để tránh mất chi tiết.

3.3.4 Truy xuất từ đồ thị (Retrieve from the Graph)

Chiến lược U-retrieve:

- Sử dụng thẻ mô tả tóm tắt để xác định đồ thị phù hợp.
- Truy xuất thông tin từ các *meta-graph* liên quan.
- Tạo phản hồi trung gian dựa trên thực thể liên quan và kiến thức nền.
- Tổng hợp thêm từ cấp cao hơn trong đồ thị để tạo phản hồi cuối cùng, đảm bảo bao quát và chính xác.

3.3.5 Đánh giá phương pháp

Các kết quả đánh giá phương pháp MedGraphRAG được trình bày rất chi tiết và bao gồm nhiều khía cạnh khác nhau.

Các chỉ số đánh giá (Metrics)

MedGraphRAG được đánh giá bằng các thước đo chính sau:

- **Faithfulness (Tính trung thực):** Kiểm tra mức độ mà câu trả lời của mô hình LLM khớp với thông tin từ tài liệu y khoa thực tế. Các công cụ đánh giá được sử dụng bao gồm: FactScore, BLEU, ROUGE-L, BERTScore, v.v.
- **Correctness (Tính đúng đắn):** Được đánh giá bởi chuyên gia y tế hoặc thông qua các tập dữ liệu gán nhãn (ví dụ: câu trả lời đúng/sai).
- **Completeness (Tính đầy đủ):** Đo lường mức độ mà câu trả lời bao phủ đầy đủ các yếu tố quan trọng trong tài liệu tham khảo.

Bộ dữ liệu đánh giá

Phương pháp được đánh giá trên các tập dữ liệu đa dạng, bao gồm:

- **9 bộ QA y tế:** như *PubMedQA*, *MedMCQA*, *MedicationQA*, *HealthSearchQA*, v.v.
- **2 bộ dữ liệu kiểm chứng sự thật:** như *HealthVer* và *SCI-Fact*.
- **1 tập dữ liệu tạo văn bản dài:** dùng để đánh giá khả năng phản hồi toàn diện của mô hình với các truy vấn y tế phức tạp.

Kết quả nổi bật

MedGraphRAG vượt trội hơn GPT-4 + RAG truyền thống ở tất cả các thước đo.

Đặc biệt tốt trong các truy vấn dài, phức tạp, nơi cần tổng hợp kiến thức từ nhiều phần tài liệu khác nhau.

Khả năng truy vết nguồn tài liệu (source-grounding) cao, giúp tăng độ tin cậy trong các ứng dụng y tế nhạy cảm.

3.4 Underthesea Toolkit

Underthesea[2]là bộ dữ liệu module Python nguồn mở và các hướng dẫn hỗ trợ nghiên cứu và phát triển về Xử lý ngôn ngữ tự nhiên tiếng Việt. Nó cung cấp API cực kỳ dễ dàng để nhanh chóng áp dụng các mô hình NLP đã được huấn luyện trước cho văn bản tiếng Việt, chẳng hạn như phân đoạn từ, gắn thẻ một phần giọng nói (PoS), nhận dạng thực thể được đặt tên (NER), phân loại văn bản và phân tích cú pháp phụ thuộc.

Underthesea được hỗ trợ bởi một trong những thư viên học sâu phổ biến nhất, Pytorch, giúp nó dễ dàng train các mô hình học sâu và thử nghiệp các phương pháp tiếp cận mới bằng cách sử dụng các Module và Class của Underthesea.

Underthesea được công bố theo giấy phép GNU General Public License v3.0. Các quyền của giấy phép này có điều kiện là cung cấp mã nguồn hoàn chỉnh của các tác phẩm được cấp phép và sửa đổi, bao gồm các tác phẩm lớn hơn sử dụng tác phẩm được cấp phép, theo cùng một giấy phép.

3.5 Graphdatascience

Thư viện graphdatascience[3]là một công cụ Python cho phép tương tác với Neo4j Graph Data Science (GDS). Thư viện này hỗ trợ thực hiện các thao tác đồ thị, chạy các thuật toán và xây dựng pipeline học máy trong GDS. API của thư viện cung cấp các phép toán

[2]Underthesea `https://github.com/undertheseanlp/underthesea`

đồ thị mạnh mẽ và dễ sử dụng, giúp người dùng dễ dàng làm việc với dữ liệu đồ thị phức tạp trong môi trường Neo4j.

3.6 Langchain

LangChain[4]là một thư viện mã nguồn mở được phát triển bằng Python và JavaScript, giúp xây dựng các ứng dụng sử dụng mô hình ngôn ngữ lớn (LLM) như GPT-4, GPT-3.5, và các mô hình khác. Thư viện này cung cấp các công cụ để kết nối mô hình ngôn ngữ với dữ liệu bên ngoài, cho phép tạo ra các ứng dụng thông minh và có khả năng xử lý ngữ cảnh phức tạp.

Với LangChain, các nhà phát triển có thể điều chỉnh linh hoạt mô hình ngôn ngữ cho các bối cảnh kinh doanh cụ thể bằng cách chỉ định các bước cần thiết để tạo ra kết quả mong muốn.

Chuỗi là nguyên tắc cơ bản chứa nhiều thành phần AI khác nhau trong LangChain để đưa ra câu trả lời nhận biết ngữ cảnh. Chuỗi là một loạt các hành động tự động từ truy vấn của người dùng đến đầu ra của mô hình. Ví dụ: các nhà phát triển có thể sử dụng chuỗi để:

- Kết nối với các nguồn dữ liệu khác nhau.
- Tạo nội dung độc đáo.
- Dịch nhiều ngôn ngữ.
- Trả lời các truy vấn của người dùng.

Chuỗi được hình thành từ các liên kết. Mỗi hành động được các nhà phát triển xâu chuỗi lại với nhau để tạo thành chuỗi được kết nối gọi là một liên kết. Với các liên kết, nhà phát triển có thể chia các tác vụ phức tạp thành nhiều tác vụ nhỏ hơn. Ví dụ về các liên kết bao gồm:

- Định dạng đầu vào của người dùng.
- Gửi truy vấn đến LLM.
- Truy xuất dữ liệu từ kho lưu trữ đám mây.
- Dịch từ ngôn ngữ này sang ngôn ngữ khác.

Trong khung LangChain, một liên kết chấp nhận đầu vào từ người dùng và chuyển đầu vào đó đến các thư viện LangChain để xử lý. LangChain cũng cho phép sắp xếp lại liên kết để tạo các quy trình làm việc AI khác nhau.

[3]graphdatascience `https://pypi.org/project/graphdatascience/`
[4]Langchain `https://www.langchain.com/`

Chương 4

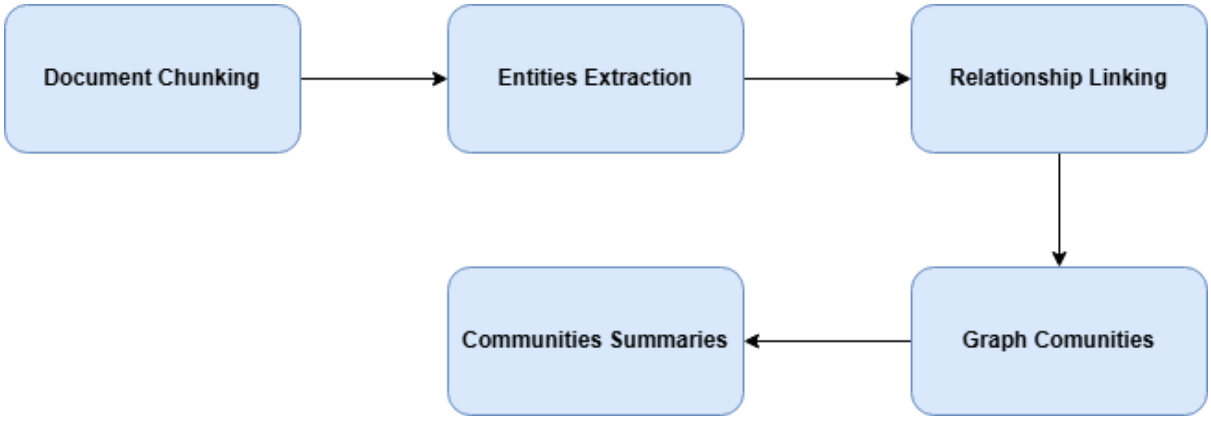
Phương pháp đề xuất

Trong chương này, tác giả sẽ trình hướng giải quyết bày bài toán ban đầu, đồng thời giới thiệu các thách thức và vấn đề cần giải quyết, cũng như các giải pháp được đề xuất để đạt được mục tiêu đề ra.

4.1 Tổng quan về kiến trúc

Các kỹ thuật RAG đã cho thấy triển vọng trong việc giúp các LLM lý luận về các tập dữ liệu riêng tư - dữ liệu mà LLM không được đào tạo và chưa bao giờ thấy trước đó, chẳng hạn như nghiên cứu độc quyền, tài liệu kinh doanh hoặc giao tiếp của một doanh nghiệp. Baseline RAG được tạo ra để giúp giải quyết vấn đề này, nhưng khi triển khai sẽ có những tình huống mà Baseline RAG hoạt động rất kém. Ví dụ Baseline RAG gặp khó khăn khi nối kết các điểm thông tin. Điều này xảy ra khi trả lời một câu hỏi đòi hỏi phải đi qua các mảnh thông tin khác nhau, đôi khi thông tin sẽ khá rắc rối, sau đó được tổng hợp lại. Và đôi khi do sự phức tạp hoặc dư thừa từ dữ liệu đưa vào mà LLM trả lời sai.

Để giải vấn đề này, Graph RAG[20] ra đời, thay vì lưu trữ các chunk text và embedding của nó để tìm kiếm, thì Graph RAG biểu diễn và lưu trữ dưới dạng đồ thị, cùng các kỹ thuật nâng cao về gom nhóm, tìm kiếm để đưa ra câu trả lời tối ưu nhất, khắc phục được nhược điểm nêu trên[21]. Và để áp dụng Graph RAG vào tập dữ liệu giáo dục, tác giả đề xuất một cách tiếp cận được biểu diễn ở hình bên dưới, gọi là **hgRAG**.



Hình 4.1: hgRAG pipeline

Cách hoạt động của hgRAG:

- Bước 1: Xây dựng biểu đồ tri thức từ văn bản (dựa trên các thực thể và mối quan hệ giữa chúng).
- Bước 2: Dùng các thuật toán phát hiện cộng đồng để chia biểu đồ này thành các nhóm nhỏ.
- Bước 3: Mỗi nhóm sẽ được tóm tắt thành các câu trả lời cục bộ.
- Bước 4: Các bản tóm tắt cục bộ này sẽ được kết hợp lại thành một câu trả lời tổng quan cho câu hỏi toàn cục.

4.2 Graph RAG Approach & Pipeline

4.2.1 Phân đoạn văn bản (Document chunking)

Bước quan trọng đầu tiên trong quy trình xây dựng đồ thị tri thức và trả lời câu hỏi trong phương pháp Graph RAG là tách các tài liệu nguồn (source documents) thành các đoạn văn bản nhỏ hơn (text chunks) để dễ dàng xử lý và phân tích trong các bước tiếp theo.

Phương pháp được sử dụng để tách các văn bản thành các đoạn ngắn được sử dụng là dựa trên sự thay đổi chủ đề được trình bày ở mục 3.1.

Các đoạn văn bản sau khi tách được lưu theo từng loại chủ đề riêng biệt và hầu như không có sự trùng lặp để tăng hiệu suất xử lý ở các bước tiếp theo.

4.2.2 Trích xuất thực thể (Entity Extraction)

Sau khi thực hiện quá trình phân đoạn, tác giả sẽ tiến hành xác định và trích xuất các thực thể (entities) và mối quan hệ (relationships) từ mỗi đoạn văn bản (text chunk).

Quy trình thực hiện:

1. **Trích xuất thực thể**:

   - **Thực thể**: Là các đối tượng quan trọng được nhắc đến trong văn bản (ví dụ: người, địa điểm, tổ chức, hoặc các khái niệm).
   - Các thông tin về thực thể bao gồm:
     - **Tên thực thể** (*name*): Ví dụ, "Albert Einstein".
     - **Loại thực thể** (*type*): Ví dụ, "nhà khoa học" hoặc "địa điểm".
     - **Mô tả thực thể** (*description*): Ví dụ, "nhà vật lý lý thuyết nổi tiếng với thuyết tương đối".

2. **Trích xuất mối quan hệ**:

   - Sau khi xác định được các thực thể, tác giả sẽ tìm các mối quan hệ giữa chúng, ví dụ:
     - **Nguồn** (*source*): Thực thể bắt đầu mối quan hệ.
     - **Đích** (*target*): Thực thể kết thúc mối quan hệ.
     - **Mô tả mối quan hệ** (*description*): Ví dụ, "là giáo viên của", "được phát minh bởi".

3. **Trích xuất thông tin bổ sung (covariates)**:

   - Ngoài thực thể và mối quan hệ, có thể trích xuất thêm các thuộc tính bổ sung, bao gồm:
     - **Chủ thể** (*subject*), **đối tượng** (*object*).
     - **Loại thông tin** (*type*), **mô tả** (*description*).
     - **Nguồn gốc thông tin** (*source text span*).
     - **Thời gian bắt đầu và kết thúc** (*start/end dates*).
     - **Embeddings**.

4.2.3 Liên kết thực thể (Relationship linking)

Liên kết thực thể là quá trình rút trích và xây dựng các mệnh đề quan hệ (relation) từ các thực thể, sự mối quan hệ và các thông tin liên quan từ bước để tạo thành một bộ ba (triplet).

Ví dụ cho một câu văn như sau:

"Chương trình đào tạo thạc sĩ công nhận chứng chỉ bồi dưỡng sau đại học (Chứng chỉ có hiệu lực 3 năm kể từ ngày cấp)."

Từ câu trên, tiến hành các bước đã trình bày, ta thu được:

- Entities: Chương trình đào tạo thạc sĩ, Chứng chỉ bồi dưỡng sau đai học.

- Relations: Công nhận.
- Thông tin bổ sung: Chứng chỉ có hiệu lực 3 năm kể từ ngày cấp.

Kết quả quá trình này sẽ tạo ra một chuỗi json biểu diễn cho triple (Sinh viên, có, GPA dưới 2.0) có dạng như sau:


```
{
    "start": {
    "identity": 127,
    "labels": ["__Entity__","Concept"],
    "properties": {
        "wcc": 101,
        "description": "Chứng chỉ có thời hạn hiệu lực 3 năm kể từ ngày
        cấp.",
        "id": "Chứng Chỉ Bồi Dưỡng Sau Đại Học",
        "embedding": [-0.002568683587014675,...],
        "communities": [101,29,34]
    },
    "elementId": "4:0b05b9d5-53ab-4890-a95a-b1d9099fa44f:127"
    },
    "relationship": {
    "identity": 1153045749420785791,
    "start": 127,
    "end": 20,
    "type": "CÔNG_NHẬN",
    "properties": {
        "description": "Chứng chỉ được công nhận cho môn học tương
        ứng thuộc chương trình đào tạo thạc sĩ."
    },
    "elementId": "5:0b05b9d5-53ab-4890-a95a-b1d9099fa44f:1153045749420785791",
    "startNodeElementId": "4:0b05b9d5-53ab-4890-a95a-b1d9099fa44f:127",
    "endNodeElementId": "4:0b05b9d5-53ab-4890-a95a-b1d9099fa44f:20"
    },
    "end": {
    "identity": 20,
    "labels": ["__Entity__","Concept","Program"],
    "properties": {
        "wcc": 0,
        "description": "Chương trình đào tạo thạc sĩ có khối lượng 60
```

```
        tín chỉ.",
        "id": "Chương Trình Đào Tạo Thạc Sĩ",
        "embedding": [-0.019571976736187935,...],
        "communities": 101,29,34]
    },
    "elementId": "4:0b05b9d5-53ab-4890-a95a-b1d9099fa44f:20"
    }
}
```


Trong đó:

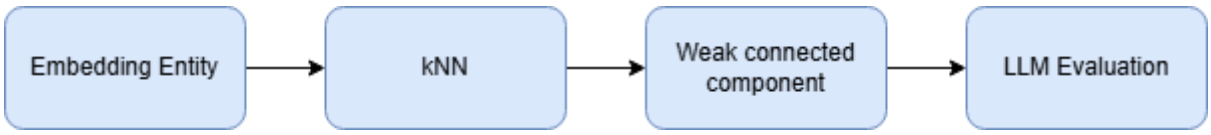
- start: Thực thể bắt đầu với các thông tin như labels (loại thực thể), description (mô tả), elementId (định danh thực thể)...
- end: Thực thể kết thúc.
- relationship: Quan hệ.

4.2.4 Lưu trữ dữ liệu quan hệ

Ở bước 4.2.3, tác giả đã thu được các bộ ba biểu diễn dữ liệu mệnh đề quan hệ dưới dạng các chuỗi json. Dữ liệu từ các mệnh đề quan hệ nêu trên là cơ sở để xây dựng các đồ thị tri thức (Knowledge Graph).

Để lưu các đồ thị tri thức siêu quan hệ (hyper-relational knowledge graphs) vào chuẩn cơ sở dữ liệu Neo4j, với mỗi node là một entity, và các cạnh là các quan hệ tương ứng. Tuy nhiên, để quản lý và truy vấn, cần xử lý hiệu quả các dữ liệu trên.

De-duplication: Đảm bảo rằng mỗi thực thể được biểu diễn duy nhất và chính xác, ngăn ngừa trùng lặp và hợp nhất các bản ghi tham chiếu đến cùng một thực thể trong thế giới thực. Duy trì tính toàn vẹn và nhất quán của dữ liệu trong biểu đồ. Nếu không De-duplication, biểu đồ tri thức sẽ bị phân mảnh và dữ liệu không nhất quán, dẫn đến lỗi và thông tin chi tiết không đáng tin cậy.



Hình 4.2: De-duplication

Quy trình loại bỏ trùng lặp:

1. **Embedding Entity** — Bắt đầu với tất cả các thực thể trong biểu đồ, thêm thuộc tính embedding chứa thông tin vector của các thực thể.
2. **kNN** — Xây dựng đồ thị kNN, kết nối các thực thể tương tự dựa trên *embedding*.

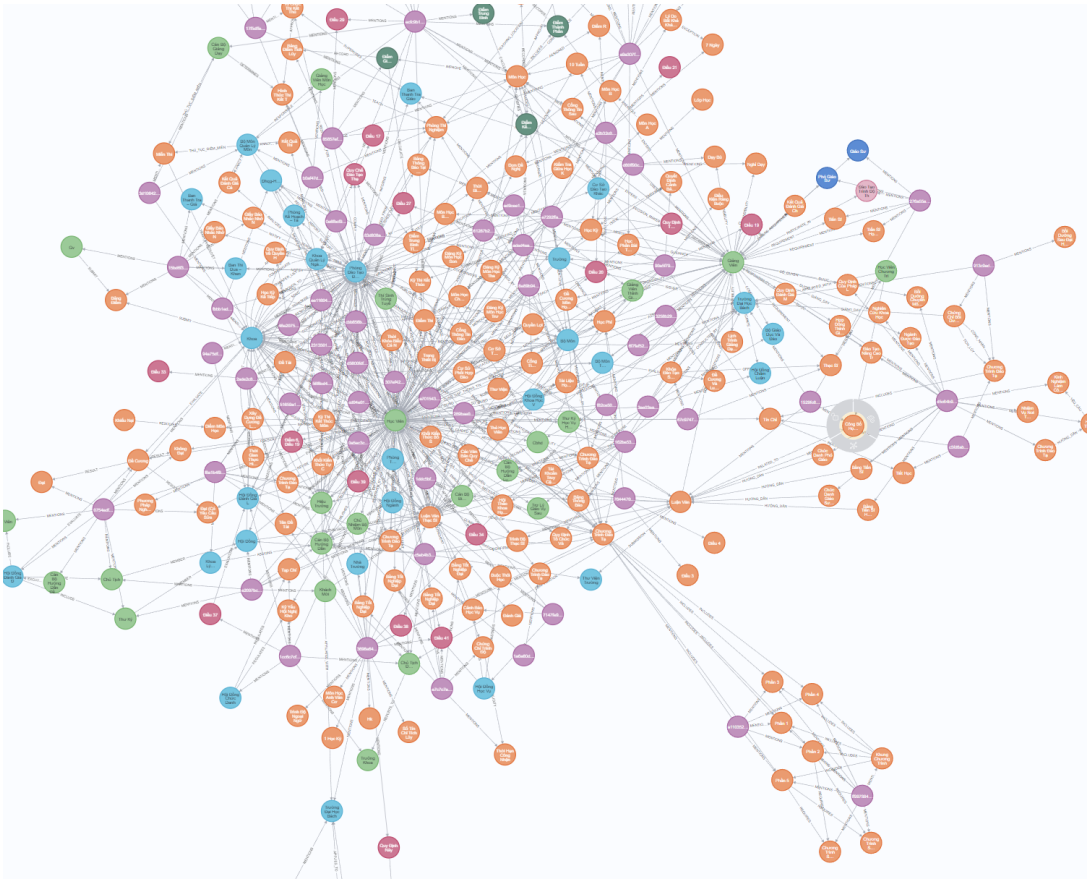
3. **Thành phần liên thông yếu (Weak Connected Component)** — Xác định các thành phần kết nối yếu trong đồ thị kNN, nhóm các thực thể có khả năng tương tự nhau. Thêm bước lọc khoảng cách sau khi các thành phần này đã được xác định.

4. **Đánh giá bằng LLM** — Sử dụng LLM để đánh giá các thành phần này và quyết định xem các thực thể trong mỗi thành phần có nên được sáp nhập hay không, từ đó đưa ra quyết định cuối cùng về việc giải quyết thực thể.

Sau khi xây dựng hệ thống lưu trữ, việc truy vấn và biểu diễn các đồ thị tri thức (*knowledge graph*) sẽ được quan tâm tiếp theo. Ngôn ngữ truy vấn Cypher được sử dụng để truy xuất thông tin trên các đồ thị này.

Ví dụ: Truy vấn tất cả các node có nhãn `Organization` và giới hạn kết quả ở 25 node đầu tiên:

```
MATCH (n:Organization) RETURN n LIMIT 25
```



Hình 4.3: Cơ sở dữ liệu đồ thị

4.2.5 Xây dựng đồ thị cộng đồng (Graph Communities)

Community detection (phát hiện cộng đồng) là quá trình phân tách hoặc phân cụm các đỉnh trong một mạng hoặc đồ thị thành các nhóm hoặc cộng đồng dựa trên mối liên kết mạng giữa chúng. Mục tiêu của community detection là tìm hiểu cấu trúc tổ chức và mối quan hệ trong mạng, tìm ra các nhóm tương đồng hoặc có chức năng tương tự. Các cộng đồng trong một mạng có thể được xác định dựa trên nhiều tiêu chí khác nhau, bao gồm mối quan hệ tương đồng, gần gũi, hay tương tác giữa các thành viên trong cộng đồng.

Tác giả sử dụng thuật toán phát hiện cộng đồng (community detection) để nhóm các thực thể và mối quan hệ liên quan thành các cộng đồng, dựa trên độ kết nối mạnh giữa chúng.

Nghiên cứu sẽ sử dụng thuật toán Leiden. cùng với Graphdatascience là công cụ được sử dụng để hỗ trợ hiện thực thuật toán.

Thuật toán Leiden cho phép phân chia đồ thị thành nhiều cấp độ cộng đồng, từ cấp thấp (chi tiết) đến cấp cao (bao quát).

Cộng đồng cấp thấp (Leaf-level communities): Đây là các cộng đồng ban đầu, thường chứa các nhóm nút liên kết mạnh với nhau.

Cộng đồng cấp cao (Higher-level communities): Sau khi các cộng đồng cấp thấp được hợp nhất, thuật toán sẽ tạo ra các cộng đồng lớn hơn, với các node tương ứng là các cộng đồng cấp thấp đã được hợp nhất.

Hình 4.4: Minh họa một cộng đồng được xây dựng dựa trên thuật toán Leiden

4.2.6 Tóm tắt cộng đồng (Communities Summaries)

Mục đích của bước này nhằm tạo các tóm tắt chi tiết cho từng cộng đồng, cung cấp cái nhìn toàn diện về từng phần của dữ liệu và đảm bảo rằng các tóm tắt này có thể được sử dụng để trả lời câu hỏi hoặc làm cơ sở cho các phân tích toàn cục.

Quy trình thực hiện:

- Cộng đồng cấp thấp nhất (leaf-level communities): Đối với các cộng đồng ở cấp thấp nhất (chi tiết nhất), tất cả các thực thể, mối quan hệ, và thuộc tính liên quan được đưa vào một cửa sổ ngữ cảnh cho LLM, sau đó tiến hành tạo ra một bản tóm tắt cô đọng dựa trên các thông tin này.
- Cộng đồng cấp cao hơn (higher-level communities): Đối với các cộng đồng lớn hơn, tóm tắt của các cộng đồng con được sử dụng thay vì thông tin chi tiết từng thực thể, để đảm bảo không vượt quá giới hạn token.

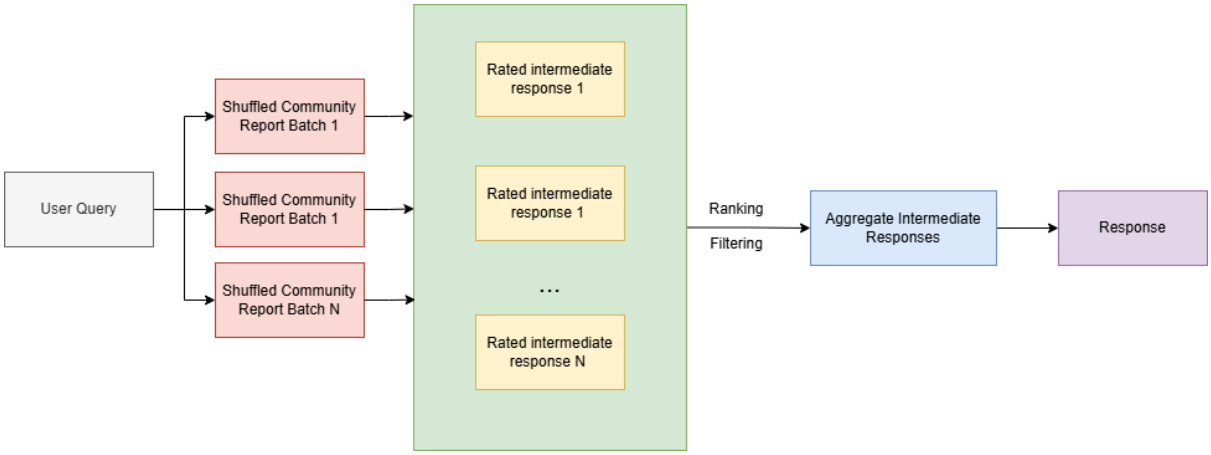
Khi tổng hợp thông tin, các thực thể và mối quan hệ có mức độ quan trọng cao (ví dụ: số lượng liên kết lớn) sẽ được ưu tiên. Thông tin được thêm vào tóm tắt theo thứ tự ưu tiên cho đến khi đạt giới hạn token của mô hình.

4.2.7 Truy vấn dữ liệu

Việc truy vấn dữ liệu trong Graph RAG bao gồm 2 phần: truy vấn toàn cục và truy vấn cục bộ.

Truy vấn toàn cục

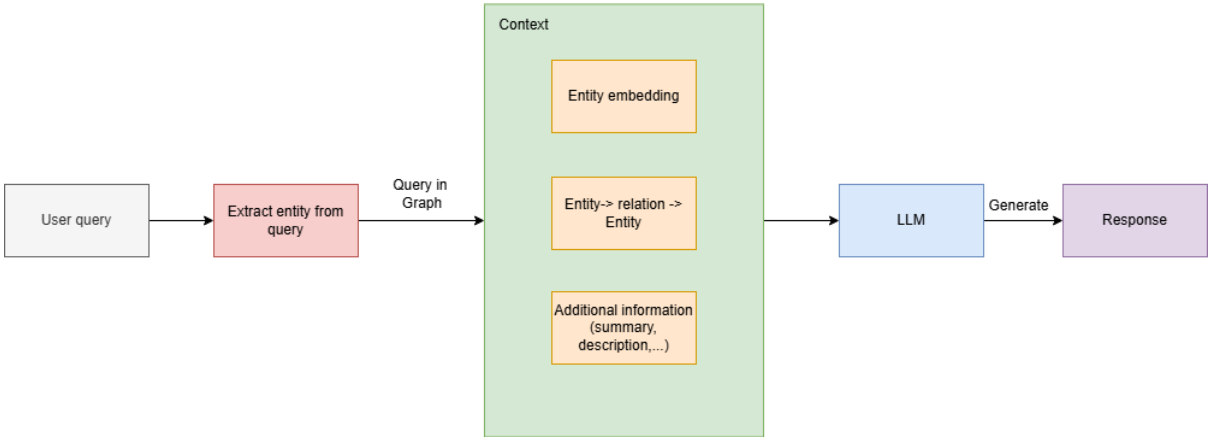
- Sử dụng với các câu hỏi mang tính toàn diện, tổng quan về dữ liệu, những câu hỏi mang tính khái quát chung mà RAG truyền thống có khả năng thất bại cao.
- Công việc truy vấn sẽ dựa trên các Communities Summaries đã được xây dựng trước đó:
  - Truy vấn ra các bản tóm tắt (Community Summaries) có liên quan nhất đến câu hỏi ban đầu.
  - Các Communities Summaries được chia thành các đoạn nhỏ hơn (chunks) với kích thước cố định phù hợp với giới hạn ngữ cảnh (context window) của mô hình LLM.
  - Các đoạn được xáo trộn ngẫu nhiên để phân tán thông tin, tránh việc tất cả thông tin liên quan bị cô lập trong một đoạn mà có thể bị bỏ sót khi xử lý.
  - Dùng LLM để sinh câu trả lời cho từng chunk và tiến hành đánh giá.
  - Các câu trả lời được đánh giá tốt sẽ được đưa vào LLM để sinh ra các phản hồi cho đến khi đạt giới hạn token.



Hình 4.5: Minh họa luồng truy vấn toàn cục

Truy vấn cục bộ

- Phù hợp với các câu hỏi sâu về một chủ đề nào đó, phù hợp để suy luận sâu hơn về các thực thể và các mối quan hệ.
  - Xác định một tập các thực thể từ các knowledge graph có liên quan từ truy vấn ban đầu. Các thực thể này là các node trong graph, cho phép truy xuất thêm các thông tin liên quan để tạo thành câu trả lời như các mối quan hệ, các thực thể được liên kết, các thông tin bổ sung.
  - Ngoài ra, có thể trích xuất các đoạn văn bản được liên kết với các thực thể (nếu có) để làm giàu cho câu trả lời.
  - Các kết quả từ các truy vấn này sẽ được sử dụng phù hợp với LLM để sinh ra câu trả lời cuối cùng.



Hình 4.6: Luồng truy vấn cục bộ

4.3 Kết quả hiện thực

4.3.1 Dữ liệu thực nghiệm

Nguồn dữ liệu đầu vào cho pipeline Graph RAG là các văn bản giáo dục được lấy từ kho dữ liệu, từ các website chính thống của trường Đại học Bách Khoa Thành phố Hồ Chí Minh. Ở đây, 25 văn bản, với nội dung chủ yếu liên quan đến các quy chế và quy định được sử dụng.

Một tập dataset gồm 3000 câu hỏi và câu trả lời tương ứng với các văn bản ở trên cũng được xây dựng nhằm phục vụ công việc đánh giá kết quả.

4.3.2 Phương pháp đánh giá

BERTScore [22] là một phương pháp đánh giá độ giống nhau ngữ nghĩa giữa hai đoạn văn bản, dựa trên các vector embedding sinh ra bởi các mô hình ngôn ngữ như BERT, RoBERTa, v.v.

Khác với các chỉ số truyền thống như BLEU, ROUGE (so sánh theo từ hoặc n-gram), BERTScore so sánh **ý nghĩa** của các từ, thay vì chỉ dựa trên hình thức.

Quy trình tính BERTScore:

Giả sử:

- **Candidate**: Câu trả lời từ mô hình (Prediction).
- **Reference**: Đáp án chuẩn (Ground Truth).

Các bước tính toán:

1. **Tokenize**: Candidate và Reference được token hóa thành các từ/token nhỏ.

   Ví dụ:

   - Candidate tokens: [sinh, viên, cần, 135, tín, chỉ]
   - Reference tokens: [sinh, viên, phải, hoàn, thành, 135, tín, chỉ]

2. **Embedding**: Mỗi token được ánh xạ thành một vector embedding bằng mô hình ngôn ngữ như BERT. Vector thể hiện ý nghĩa ngữ cảnh của token.

3. **Tính toán độ tương đồng**: Với mỗi token trong Candidate, tìm token trong Reference có độ tương đồng cosine cao nhất. Ngược lại, cũng thực hiện từ Reference đến Candidate.

4. **Tính Precision, Recall và F1**:

- **Precision**: Trung bình độ tương đồng cao nhất của mỗi token trong Candidate so với Reference, có giá trị trong khoảng [0;1].
- **Recall**: Trung bình độ tương đồng cao nhất của mỗi token trong Reference so với Candidate, có giá trị trong khoảng [0;1].
- **F1**: Trung bình điều hòa giữa Precision và Recall, có giá trị trong khoảng [0;1].

Công thức toán học:

Gọi:

- C là tập token trong Candidate,
- R là tập token trong Reference,

ta có:

$$\text{Precision} = \frac{1}{|C|} \sum_{c \in C} \max_{r \in R} \text{cosine\_similarity}(c, r)$$

$$\text{Recall} = \frac{1}{|R|} \sum_{r \in R} \max_{c \in C} \text{cosine\_similarity}(r, c)$$

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Ưu điểm của BERTScore:

- Hiểu ý nghĩa ngữ cảnh: Không yêu cầu phải khớp chính xác từ ngữ.
- Ngôn ngữ độc lập: Áp dụng tốt cho nhiều ngôn ngữ khác nhau.
- Phát hiện tốt lỗi nội dung: Phát hiện lỗi thêm hoặc thiếu ý.

4.3.3 Kết quả

Khi chạy thực nghiệm trên tập dữ liệu quan đầu, kết quả chỉ số F1 thu được là 0.776. Nếu một mô hình có F1 Score cao, nó có nghĩa là nó đang đưa ra nhiều dự đoán chính xác và có độ phủ tốt.

Bảng 4.1: Bảng kết quả đánh giá BERTscore

Phương pháp	Precision	Recall	F1
Graph RAG	0.819	0.739	0.776
RAG	0.813	0.702	0.753
GPT-4o	0.698	0.570	0.628

- Với mô hình Gpt-4o, không được train trên tập dữ liệu thực nghiệp, các câu trả lời chủ yếu được *"bịa"* ra, do đó chỉ số F1 chỉ là 0.628, độ tin cậy thấp.
- Khi sử dụng RAG và Graph RAG trên tập dữ liệu ban đầu, kết quả thu được có độ tin cậy cao hơn, và Graph RAG đạt kết quả cao nhất.

Chương 5

Kết luận

Nội dung chương này sẽ khái quát lại những điều đã làm được, nhận xét và định hướng tương lai.

5.1 Nhận xét

Bài báo cáo Luận văn Thạc sĩ này đã trình bày về các mô hình, các công trình nghiên cứu liên quan để đề xuất một phương pháp cho phép triển khai pipeline Graph RAG - hgRAG trong lĩnh vực giáo dục để giải quyết các khó khăn khi nối kết các điểm thông tin, các tác vụ tóm tắt, khái quát hóa và đem lại cái nhìn tổng quan về dữ liệu, đồng thời cũng đã tiến hành triển khai và đánh giá kết quả theo phương pháp được đề xuất. Các kết quả ban đầu thu được mang hướng tích cực, pipeline đề xuất đem lại khả năng cải thiện chất lượng câu trả lời dựa trên cái knowledge graph, đem lại hiệu quả tốt hơn so với RAG trong nhiều trường hợp.

Tuy nhiên, trong quá trình nghiên cứu vẫn còn gặp nhiều thách thức cần được giải quyết trong thời gian tới:

- Tốn nhiều không gian lưu trữ hơn so với RAG.
- Việc triển khai phức tạp. Thay vì chỉ cần lưu trữ các chunk text và embedding của chúng như trong RAG truyền thống, hgRAG cần thực hiện nhiều công đoạn, lưu trữ dữ liệu dưới dạng knowledge graph và embedding. Ngoài ra, liên kết thực thể là một bước vô cùng quan trọng trong pipeline Graph RAG. Tuy nhiên, việc xử lý dữ liệu để thu được các bộ ba (triples) còn gặp nhiều khó khăn. Các mô hình như Transformer hay phoBERT được sử dụng để hỗ trợ quá trình này, tuy nhiên, các kết quả thu được tính đến thời điểm hiện tại chưa thực sự tốt. Do đó, các công việc trong giai đoạn này thương phải được xử lý nhiều lần.
- Kết quả trả lời tùy thuộc vào lượng thông tin trích xuất được từ quá trình truy vấn dữ liệu. Do đó, trong một số trường hợp, database chứa các knowledge graph

không mang đủ thông tin cần thiết, cần tiến hành bổ sung dữ liệu vào database để làm giàu cho ngữ cảnh của câu trả lời.

- Phương pháp đã giải quyết các vấn đề liên quan đến tóm tắt thông tin, khái quát về dữ liệu so với baseline RAG, tuy nhiên, tốc độ truy vấn cho các trường hợp này cũng là một vấn đề cần được quan tâm.

5.2 Hướng phát triển trong tương lai

Mặc dù đã đạt được những kết quả khả quan, pipeline Graph RAG vẫn có thể được cải thiện và phát triển thêm để nâng cao hiệu suất. Dưới đây là một số hướng phát triển tiềm năng:

- Cải thiện khả năng của module trích xuất thực thể, mối quan hệ.
- Xây dựng cấu trúc dữ liệu liên kết vào tập dữ liệu chuẩn cho các từ ngữ chuyên ngành, từ ngữ học thuật...
- Tạo một pipeline Graph RAG cho dữ liệu có cấu trúc [23].

Bằng cách tập trung vào những hướng phát triển này, nghiên cứu có thể được mở rộng và hoàn thiện hơn. Nghiên cứu này đã khẳng định tiềm năng to lớn của Graph RAG vào việc truy vấn thông tin từ các nguồn bên ngoài nhằm cải thiện hiệu quả, độ chính xác của các hệ thống Q&A, hứa hẹn sẽ mở ra thêm những hướng đi mới cho lĩnh vực xử lý ngôn ngữ tự nhiên.

Danh sách tham khảo

[1] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, pp. 1137–1155, 2003.

[2] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, pp. 3111–3119, 2013.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[5] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[6] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[7] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[9] Z. Lin, Y. Deng, J. Kuo, R. Zhang, and H. Liu, "Structured attention networks," *arXiv preprint arXiv:1702.00887*, 2017.

[10] M. Lewis, Y. Liu, N. Goyal, S. Ruder, D. Gromann, S. K. Moosavi, N. Bassil, Y. Jernite, , *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Proceedings of NeurIPS 2020*, 2020.

[11] V. Traag, L. Waltman, and N. J. van Eck, "From louvain to leiden: Guaranteeing well-connected communities," *arXiv preprint arXiv:1810.08473*, 2018.

[12] Neo4j, "What's neo4j?." `https://neo4j.com/docs/getting-started/whats-neo4j/`, n.d. Truy cập lần cuối: tháng 4, 2025.

[13] GeeksforGeeks, "Neo4j introduction." `https://www.geeksforgeeks.org/neo4j-introduction/`, n.d. Truy cập lần cuối: tháng 4, 2025.

[14] Neo4j, "Cypher query language manual." `https://neo4j.com/docs/cypher-manual/current/introduction/`, n.d. Truy cập lần cuối: tháng 4, 2025.

[15] Viblo, "Tìm hiểu về ngôn ngữ truy vấn cypher." `https://viblo.asia/p/tim-hieu-ve-ngon-ngu-truy-van-cypher-gDVK2BmAKLj`, n.d. Truy cập lần cuối: tháng 4, 2025.

[16] L. N. Khang, "Bài toán xác định nội dung trùng lặp," June 2024. Accessed: 2024-12-11.

[17] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *In: Google AI Language*, p. 1.

[18] N. Q. Dat and N. T. Anh, "Phobert: Pre-trained language models for vietnamese," *In: Findings of the Association for Computational Linguistics: EMNLP 2020*, p. 1037–1042, 2020.

[19] 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. Version 2, revised on 15 Oct 2024.

[20] N. C. J. B. A. C. A. M. S. T. J. L. Darren Edge, Ha Trinh, "From local to global: A graph rag approach to query-focused summarization," *arXiv preprint arXiv:2404.16130*, 2023.

[21] J. Wu, J. Zhu, and Y. Qi, "Medical graph rag: Towards safe medical large language model via graph retrieval-augmented generation." `https://arxiv.org/abs/2408.04187`, 2024. Truy cập lần cuối: tháng 4, 2025.

[22] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, "Bertscore: Evaluating text generation with bert," *arXiv preprint arXiv:1904.09675*, 2019. Version 3, revised on 24 Feb 2020.

[23] J. Zou, D. Fu, S. Chen, X. He, Z. Li, Y. Zhu, J. Han, and J. He, "Gtr: Graph-table-rag for cross-table question answering," *arXiv preprint arXiv:2504.01346*, 2025. Version 2, revised on 3 Apr 2025.