HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN

BÀI CÁO KẾT THÚC HỌC PHÀN

MÔN THỰC TẬP CƠ SỞ

ĐỀ TÀI

XÂY DỰNG HỆ THỐNG AI TƯ VẤN LUẬT PHÁP

Giáo viên hướng dẫn: ThS. BÙI VĂN KIÊN

Nhóm học phần: 25

Sinh viên: NGUYỄN QUANG DŨNG

Mã sinh viên: B22DCCN133

Hà Nội, 2025

Mục lục

Danh mục hình ảnh

Danh mục bảng biểu

DANH MỤC CÁC TỪ VIẾT TẮT

Từ viết tắt	**Thuật ngữ tiếng Anh/Giải thích**	**Thuật ngữ tiếng Việt/Giải thích**
GAE	Graph Auto-Encoder	Mã hóa đồ thị tự động
RAG	Retrieval-Augmented Generation	Tạo sinh tăng cường truy xuất
GNN	Graph Neural Network	Mạng nơ-ron đồ thị
LLM	Large Language Model	Mô hình ngôn ngữ lớn

1 Tổng quan dự án

1.1 Giới thiệu bài toán

Trong thời đại chuyển đổi số, nhu cầu tiếp cận thông tin pháp luật một cách nhanh chóng, chính xác và dễ hiểu ngày càng trở nên cấp thiết đối với mọi tầng lớp trong xã hội. Tuy nhiên, hệ thống văn bản pháp luật tại Việt Nam rất đồ sộ, phức tạp và thường xuyên thay đổi, gây khó khăn cho người dân, doanh nghiệp trong việc tra cứu, áp dụng đúng quy định. Các hệ thống hỏi đáp pháp luật truyền thống chủ yếu dựa trên tìm kiếm từ khóa, chưa khai thác được mối quan hệ phức tạp giữa các thực thể pháp lý, dẫn đến kết quả trả lời còn hạn chế về tính chính xác và khả năng suy luận.
Bài toán đặt ra là xây dựng một hệ thống AI có khả năng tiếp nhận câu hỏi tự nhiên của người dùng (input), truy xuất và tổng hợp thông tin từ kho dữ liệu pháp luật (bao gồm văn bản luật, các bài báo về luật pháp,các mối quan hệ giữa các thực thể pháp lý, v.v.), sau đó sinh ra câu trả lời tự động, chính xác, dễ hiểu (output). Dữ liệu đầu vào của hệ thống là các câu hỏi tiếng Việt về pháp luật và kho dữ liệu pháp luật đã được xử lý, chuẩn hóa dưới dạng văn bản và đồ thị tri thức. Đầu ra của hệ thống là các câu trả lời tự động, có thể kèm theo giải thích nguồn gốc, mối liên hệ giữa các điều khoản pháp luật liên quan.

1.2 Mục đích và ý nghĩa

Mục đích: Dự án hướng tới xây dựng một hệ thống AI hỗ trợ hỏi đáp pháp luật thông minh, tận dụng sức mạnh của các mô hình ngôn ngữ lớn (LLM) kết hợp với truy xuất tri thức từ đồ thị (Graph Retrieval-Augmented Generation - GraphRAG). Hệ thống giúp người dùng tra cứu thông tin pháp luật một cách nhanh chóng, chính xác, đồng thời có khả năng suy luận đa bước, giải thích các mối quan hệ giữa các điều khoản, văn bản pháp luật.
Ý nghĩa: Việc phát triển hệ thống này góp phần nâng cao nhận thức pháp luật, hỗ trợ doanh nghiệp và người dân tuân thủ pháp luật, đồng thời giảm tải cho các cơ quan tư vấn pháp lý truyền thống. Hệ thống còn là bước tiến quan trọng trong việc ứng dụng trí tuệ nhân tạo vào lĩnh vực pháp lý, thúc đẩy chuyển đổi số trong ngành luật tại Việt Nam.

1.3 Công nghệ sử dụng

Dự án ứng dụng các công nghệ hiện đại trong lĩnh vực trí tuệ nhân tạo và xử lý ngôn ngữ tự nhiên, bao gồm:

- **Ngôn ngữ lập trình Python:** Ngôn ngữ chính được sử dụng để xây dựng toàn bộ hệ thống nhờ tính linh hoạt, thư viện phong phú và cộng đồng mạnh.
- **PyTorch:** Framework học sâu được sử dụng để xây dựng, huấn luyện và triển khai các mô hình AI như GAE.
- **Large Language Model (LLM):** Sử dụng các mô hình ngôn ngữ lớn để sinh câu trả lời tự nhiên, mạch lạc và phù hợp ngữ cảnh.
- **Graph Retrieval-Augmented Generation (GraphRAG):** Kết hợp truy xuất thông tin từ đồ thị tri thức pháp luật với khả năng sinh ngôn ngữ của LLM, giúp tăng cường độ chính xác và khả năng suy luận.
- **Graph Autoencoder (GAE):** Ứng dụng trong việc học biểu diễn các thực thể và quan hệ trong đồ thị pháp luật, hỗ trợ cho quá trình truy xuất và suy luận.
- **Cơ sở dữ liệu đồ thị (Neo4j):** Lưu trữ và quản lý dữ liệu pháp luật dưới dạng đồ thị, cho phép truy vấn các mối quan hệ phức tạp giữa các thực thể pháp lý.
- **MongoDB:** Lưu trữ dữ liệu văn bản gốc, bài báo, văn bản pháp luật trước khi xử lý và chuyển đổi sang đồ thị.

- **Flask:** Sử dụng Flask để xây dựng các API phục vụ giao tiếp giữa các thành phần của hệ thống và giao diện người dùng.
- **Streamlit:** Xây dựng giao diện web thân thiện, cho phép người dùng nhập câu hỏi và nhận phản hồi trực tiếp từ hệ thống.

2 Cơ sở lý thuyết

2.1 Graph Retrieval-Augmented Generation

2.1.1 Giới thiệu về Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) là một kỹ thuật kết hợp truy xuất thông tin và sinh nội dung để cải thiện hiệu suất các tác vụ xử lý ngôn ngữ tự nhiên [6]. Trong RAG truyền thống, bộ truy xuất tìm kiếm thông tin từ nguồn dữ liệu văn bản $\mathcal{D} = \{d_1, d_2, \ldots, d_n\}$ dựa trên truy vấn q, và bộ sinh tạo câu trả lời y bằng cách tích hợp q với thông tin truy xuất $R(q) \subseteq \mathcal{D}$. Quá trình này được mô tả bởi:

$$y = \text{Generator}(q, R(q)), \tag{1}$$

trong đó $R(q)$ là tập hợp các tài liệu liên quan được truy xuất [9].
RAG truyền thống gặp hạn chế khi xử lý dữ liệu có quan hệ phức tạp, do thông tin được biểu diễn độc lập [2]. GraphRAG khắc phục bằng cách sử dụng dữ liệu cấu trúc đồ thị để khai thác thông tin quan hệ, mở rộng khả năng suy luận đa bước và xử lý các tác vụ phức tạp.

2.1.2 Định nghĩa và Đặc điểm của GraphRAG

Định nghĩa 1 (Đồ thị)**.** *Một đồ thị $G = (V, E)$ bao gồm tập hợp các nút $V = \{v_1, v_2, \ldots, v_n\}$ đại diện cho các thực thể và tập hợp các cạnh $E \subseteq V \times V$ biểu thị quan hệ giữa các thực thể.*

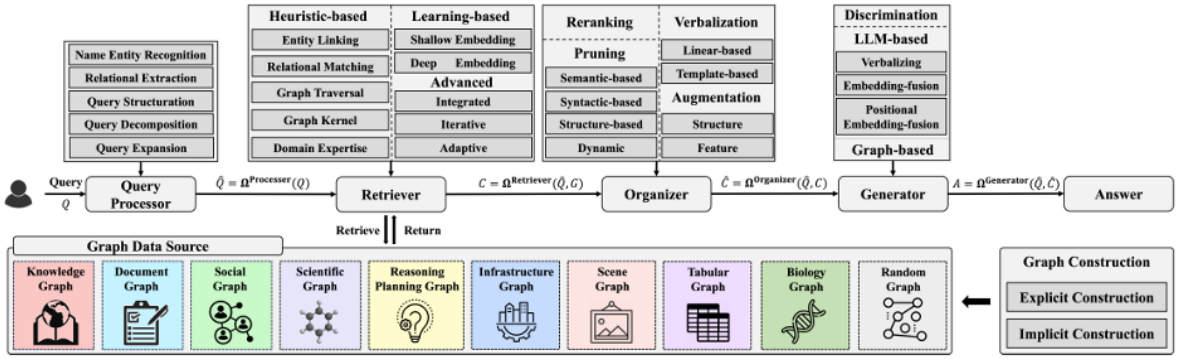
GraphRAG tích hợp dữ liệu đồ thị G vào RAG, trong đó nguồn dữ liệu là $\mathcal{D}_G = \{G_1, G_2, \ldots, G_m\}$, với mỗi G_i là một đồ thị tri thức, đồ thị tài liệu, hoặc đồ thị miền cụ thể [8]. Quá trình GraphRAG được biểu diễn bằng:

$$y = \text{Generator}(q, R_G(q)), \tag{2}$$

trong đó $R_G(q)$ là tập hợp các nút, cạnh, hoặc đồ thị con liên quan được truy xuất từ $\mathcal{D}_G$ [2].

2.1.3 Cấu trúc của GraphRAG

GraphRAG bao gồm năm thành phần chính: Bộ xử lý truy vấn, Nguồn dữ liệu đồ thị, Bộ truy xuất, Bộ tổ chức, và Bộ sinh [2].



Hình 1: Tổng quan về cấu trúc GraphRAG và các công nghệ đi kèm theo thành phần

a. Bộ xử lý truy vấn (Query Processor)
Bộ xử lý truy vấn chuyển đổi truy vấn người dùng q (thường ở dạng văn bản tự nhiên) thành một biểu diễn có cấu trúc q_s, phù hợp với dữ liệu đồ thị. Thành phần này thực hiện các chức năng như nhận diện thực thể (NER), trích xuất quan hệ (RE), cấu trúc hóa truy vấn, phân rã truy vấn, và mở rộng truy vấn [3].
Ví dụ: Với truy vấn q = "Thuốc nào điều trị sarcoma biểu mô và ảnh hưởng đến gen EZH2?", bộ xử lý truy vấn:

- Nhận diện $E_q = \{\text{sarcoma biểu mô}, \text{EZH2}\}$.
- Trích xuất $R_q = \{\text{điều trị}, \text{ảnh hưởng}\}$.
- Cấu trúc hóa thành truy vấn đồ thị q_s với các nút $V_q = \{\text{thuốc}, \text{sarcoma biểu mô}, \text{EZH2}\}$ và các cạnh $E_q = \{(\text{thuốc}, \text{sarcoma biểu mô}, \text{điều trị}), (\text{thuốc}, \text{EZH2}, \text{ảnh hưởng})\}$.

Quá trình được mô tả bằng hàm ánh xạ:

$$q_s = f_{\text{process}}(q; \theta), \tag{3}$$

trong đó θ là tham số của mô hình (ví dụ: BERT cho NER). Xác suất một token t_i thuộc thực thể e_j là:

$$P(e_j|t_i) = \text{softmax}(W \cdot \mathbf{h}_i + b), \tag{4}$$

với $\mathbf{h}_i$ là nhúng token từ BERT [3].

b. Nguồn dữ liệu đồ thị (Graph Data Source)

Nguồn dữ liệu đồ thị $\mathcal{D}_G = \{G_1, G_2, \ldots, G_m\}$ bao gồm các đồ thị $G_i = (V_i, E_i)$, trong đó V_i là tập hợp nút (thực thể) và E_i là tập hợp cạnh (quan hệ). Các loại đồ thị bao gồm đồ thị tri thức, đồ thị tài liệu, và đồ thị sinh học [8].

Ví dụ: Trong y học, G là đồ thị tri thức với $V = \{\text{sarcoma biểu mô}, \text{EZH2}, \text{thuốc A}, \text{thuốc B}\}$ và $E = \{(\text{thuốc A}, \text{sarcoma biểu mô}, \text{điều trị}), (\text{thuốc B}, \text{EZH2}, \text{ảnh hưởng})\}$. Ma trận kề $A \in R^{|V|\times|V|}$ được định nghĩa:

$$A_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Nguồn dữ liệu hỗ trợ truy xuất quan hệ với độ phức tạp lưu trữ $O(|V| + |E|)$ [11].

c. Bộ truy xuất (Retriever)

Bộ truy xuất tìm kiếm các phần tử liên quan $R_G(q_s) \subseteq V \cup E \cup \mathcal{G}$ từ $\mathcal{D}_G$ dựa trên q_s. Các kỹ thuật bao gồm liên kết thực thể, khớp quan hệ, duyệt đồ thị, và nhúng đồ thị bằng Graph Neural Networks (GNNs) [10].

Ví dụ: Với q_s chứa $E_q = \{\text{sarcoma biểu mô}, \text{EZH2}\}$, bộ truy xuất tìm các nút thuốc có cạnh liên quan. Nhúng nút v_i từ GNN tại tầng $l+1$ là:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{j\in\mathcal{N}(i)} \frac{1}{c_{ij}} \mathbf{h}_j^{(l)} W^{(l)}\right), \tag{6}$$

với $\mathcal{N}(i)$ là tập lân cận, c_{ij} là trọng số chuẩn hóa, và σ là hàm kích hoạt. Độ tương đồng với nhúng truy vấn $\mathbf{h}_{q_s}$ được tính:

$$\text{sim}(\mathbf{h}_{q_s}, \mathbf{h}_i) = \frac{\mathbf{h}_{q_s} \cdot \mathbf{h}_i}{\|\mathbf{h}_{q_s}\|\|\mathbf{h}_i\|}. \tag{7}$$

Tập hợp $R_G(q_s)$ bao gồm các nút có $\text{sim} > \tau$ [10]. **d. Bộ tổ chức (Organizer)**

Bộ tổ chức tinh chỉnh $R_G(q_s)$ để tạo $R'_G(q_s)$, loại bỏ dư thừa và sắp xếp nội dung. Các kỹ thuật bao gồm cắt tỉa đồ thị, sắp xếp lại nội dung, và tóm tắt đồ thị [7].

Ví dụ: Với $R_G(q_s)$ chứa nhiều nút thuốc, bộ tổ chức ưu tiên các nút có cả hai quan hệ "điều trị" và "ảnh hưởng". Điểm số nút v_i là:

$$\text{score}(v_i) = \alpha \cdot \text{sim}(\mathbf{h}_{q_s}, \mathbf{h}_i) + \beta \cdot C_B(v_i), \tag{8}$$

với $C_B(v_i)$ là độ trung tâm giữa. Đồ thị con G' được tối ưu:

$$G' = \arg\max_{G'\subseteq G} \sum_{v_i\in V'} \text{score}(v_i) - \lambda|V'|. \tag{9}$$

e. Bộ sinh (Generator)

Bộ sinh tích hợp q_s và $R'_G(q_s)$ để tạo câu trả lời y. Nhúng đồ thị $R'_G(q_s)$ được tổng hợp:

$$\mathbf{h}_G = \text{READOUT}(\{\mathbf{h}_i \mid v_i \in R'_G(q_s)\}), \tag{10}$$

với READOUT là hàm tổng hợp (ví dụ: trung bình). Câu trả lời được sinh:

$$y = \text{LLM}(q_s, \mathbf{h}_G; \phi), \tag{11}$$

với ϕ là tham số của mô hình ngôn ngữ lớn (LLM) [1].

Ví dụ: Với $R'_G(q_s)$ chứa các nút thuốc được tinh chỉnh, bộ sinh tạo: "Thuốc A điều trị sarcoma biểu mô và ảnh hưởng đến gen EZH2 thông qua cơ chế ức chế".

f. Tích hợp khung

Khung GraphRAG hoạt động tuần tự:

$$y = \text{Generator}(\text{Organizer}(\text{Retriever}(\text{Processor}(q), \mathcal{D}_G))), \tag{12}$$

tối ưu hóa việc khai thác thông tin quan hệ và ngữ nghĩa, phù hợp với các ứng dụng đa dạng từ y học đến mạng xã hội [2].

2.1.4 So sánh với RAG

GraphRAG mở rộng RAG truyền thống bằng cách sử dụng cấu trúc đồ thị thay vì dữ liệu văn bản độc lập, mang lại nhiều ưu điểm nhưng cũng đi kèm với một số thách thức [2]. Dưới đây là so sánh chi tiết giữa GraphRAG và RAG dựa trên các khía cạnh chính:

a. Cấu trúc dữ liệu

Trong RAG, dữ liệu được lưu trữ dưới dạng văn bản $\mathcal{D} = \{d_1, d_2, \ldots, d_n\}$, thường là các tài liệu hoặc đoạn văn, được biểu diễn bằng nhúng ngữ nghĩa (ví dụ: từ BERT hoặc DPR) [**Karpukhin2020DPR**]. Quá trình truy xuất dựa trên độ tương đồng ngữ nghĩa:

$$R(q) = \arg\max_{d_i \in \mathcal{D}} \text{sim}(\mathbf{h}_q, \mathbf{h}_{d_i}), \tag{13}$$

trong đó $\mathbf{h}_q$ và $\mathbf{h}_{d_i}$ là nhúng của truy vấn và tài liệu.

Ngược lại, GraphRAG sử dụng đồ thị $\mathcal{D}_G = \{G_1, G_2, \ldots, G_m\}$, với $G_i = (V_i, E_i)$, cho phép biểu diễn các thực thể và quan hệ [8]. Truy xuất trong GraphRAG dựa trên cấu trúc đồ thị, sử dụng nhúng GNN:

$$R_G(q) = \{v_i \in V \mid \text{sim}(\mathbf{h}_{q_s}, \mathbf{h}_i) > \tau\}, \tag{14}$$

với $\mathbf{h}_i$ được tính từ GNN [10]. Điều này cho phép GraphRAG khai thác các mối quan hệ phức tạp, ví dụ: tìm thuốc vừa "điều trị" sarcoma biểu mô vừa "ảnh hưởng" đến EZH2.

b. Khả năng truy xuất

RAG truyền thống sử dụng các phương pháp truy xuất như DPR hoặc BM25, tập trung vào độ tương đồng ngữ nghĩa nhưng thiếu khả năng suy luận đa bước [6]. Ví dụ, với truy vấn "Thuốc nào điều trị sarcoma biểu mô và ảnh hưởng đến gen EZH2?", RAG có thể trả về các tài liệu chứa từ khóa nhưng không đảm bảo kết nối hai quan hệ.

GraphRAG sử dụng các kỹ thuật duyệt đồ thị (BFS/DFS) và nhúng GNN để truy xuất các nút hoặc đồ thị con liên quan, hỗ trợ suy luận đa bước [2]. Độ phức tạp truy xuất của GraphRAG phụ thuộc vào kích thước đồ thị, thường là $O(|V| + |E|)$ cho BFS, so với $O(n)$ của RAG (với n là số tài liệu) [11].

c Xử lý quan hệ phức tạp

RAG truyền thống không hiệu quả trong việc xử lý các quan hệ phức tạp giữa các thực thể, do thiếu cấu trúc quan hệ [2]. Ví dụ, RAG khó xác định một thuốc thỏa mãn cả hai quan hệ "điều trị" và "ảnh hưởng" trong cùng một truy vấn.

GraphRAG vượt trội nhờ khả năng biểu diễn quan hệ qua cạnh đồ thị và sử dụng GNN để tổng hợp thông tin lân cận [10]. Công thức nhúng GNN:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \frac{1}{c_{ij}} \mathbf{h}_j^{(l)} W^{(l)}\right), \tag{15}$$

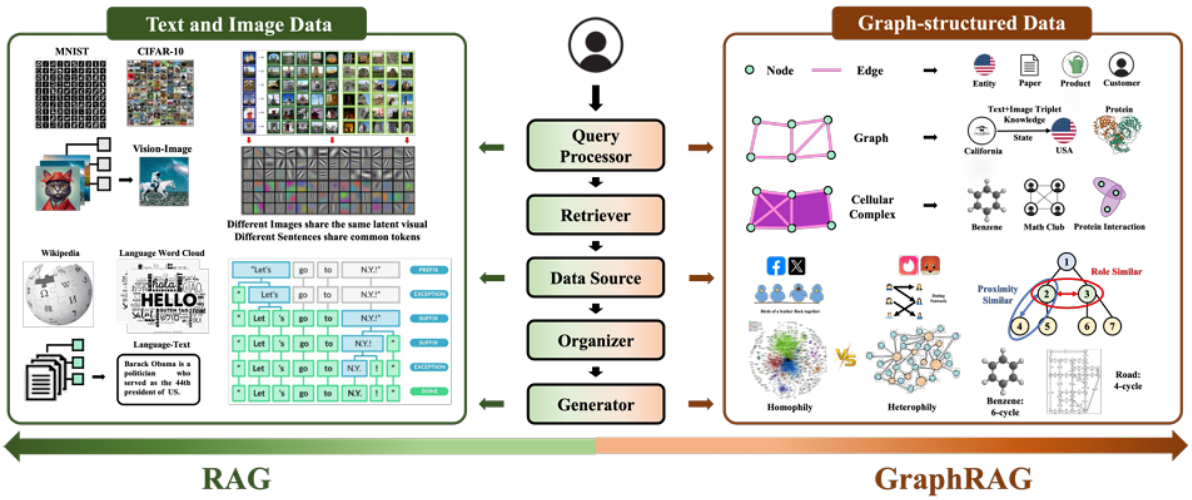
cho phép GraphRAG nắm bắt các mối quan hệ cấp cao, như đường đi giữa các nút trong đồ thị tri thức y học, pháp luật.

d. Hiệu quả tính toán

RAG truyền thống có chi phí tính toán thấp hơn, do chỉ yêu cầu nhúng văn bản và tìm kiếm tuyến tính [**Karpukhin2020DPR**]. Tuy nhiên, GraphRAG đòi hỏi tài nguyên tính toán cao hơn, đặc biệt khi xử lý đồ thị lớn với GNN, với độ phức tạp $O(|V| \cdot d^2)$ cho mỗi tầng GNN (với d là kích thước nhúng) [10]. **Ứng dụng** RAG phù hợp với các tác vụ đơn giản như trả lời câu hỏi dựa trên tài liệu, nhưng kém hiệu quả trong các lĩnh vực yêu cầu quan hệ phức tạp, như phân tích sinh học hoặc mạng xã hội [6]. GraphRAG được thiết kế cho các ứng dụng đòi hỏi suy luận quan hệ, ví dụ: phân tích mạng sinh học để tìm thuốc hoặc dự đoán liên kết xã hội [2]. **Bảng so sánh**:

Tiêu chí	RAG	GraphRAG
Cấu trúc dữ liệu	Văn bản $\mathcal{D}$	Đồ thị $\mathcal{D}_G$
Khả năng truy xuất	Ngữ nghĩa, đơn bước	Quan hệ, đa bước
Xử lý quan hệ	Kém	Tốt
Hiệu quả tính toán	Cao	Thấp hơn
Ứng dụng	Trả lời câu hỏi đơn giản	Phân tích quan hệ phức tạp

Bảng 1: So sánh RAG và GraphRAG



Hình 2: So sánh RAG và GraphRAG

Kết luận: GraphRAG vượt trội so với RAG trong việc xử lý dữ liệu có quan hệ phức tạp nhờ cấu trúc đồ thị và các kỹ thuật như GNN, nhưng yêu cầu tài nguyên tính toán cao hơn. Sự lựa chọn giữa RAG và GraphRAG phụ thuộc vào yêu cầu ứng dụng [2].

2.1.5 Ứng dụng và Thách thức

GraphRAG hỗ trợ nhiều ứng dụng, từ trả lời câu hỏi trên đồ thị tri thức đến phân tích dữ liệu sinh học. Tuy nhiên, các thách thức bao gồm:

- **Định dạng đa dạng**: Xử lý các loại đồ thị khác nhau (G có thể là đồ thị tri thức, đồ thị tài liệu, v.v.).
- **Suy luận đa bước**: Yêu cầu duyệt đồ thị với độ phức tạp $O(|V| + |E|)$.
- **Tính đặc thù theo miền**: Thiết kế riêng cho từng miền ứng dụng.

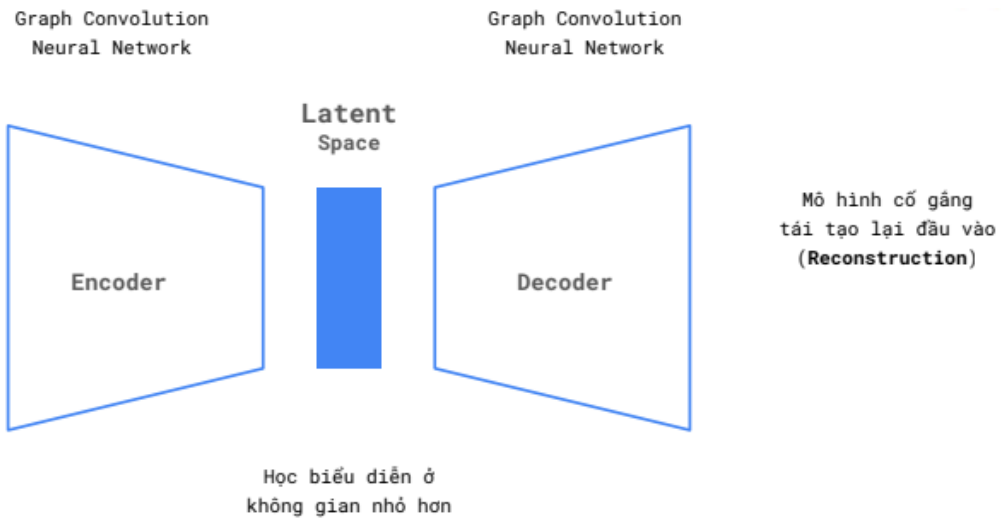
2.2 Graph Autoencoder (GAE)

Graph Autoencoder (GAE) là một mô hình học không giám sát trên đồ thị, được phát triển nhằm học biểu diễn (embedding) của các đỉnh trong đồ thị sao cho có thể khôi phục lại cấu trúc đồ thị đầu vào. Ý tưởng xuất phát từ mô hình autoencoder truyền thống, nhưng được điều chỉnh để làm việc hiệu quả với dữ liệu phi Euclid như đồ thị.

2.2.1 Kiến trúc tổng thể của GAE

Mô hình GAE bao gồm hai thành phần chính:

- **Encoder:** Dùng để ánh xạ đặc trưng đỉnh ban đầu (hoặc đơn vị) và cấu trúc đồ thị thành không gian tiềm ẩn. Thông thường, encoder được xây dựng dựa trên mạng nơ-ron tích chập trên đồ thị (Graph Convolutional Networks - GCN) [4].
- **Decoder:** Nhận đầu vào là biểu diễn tiềm ẩn và tái tạo lại ma trận kề của đồ thị. Một decoder cơ bản dùng tích vô hướng giữa các vector biểu diễn đỉnh



Hình 3: Sơ đồ mô hình GAE

a. Encoder

Encoder có nhiệm vụ biến đổi đặc trưng đầu vào $\mathbf{X} \in R^{n \times d}$ và cấu trúc đồ thị (thông qua ma trận kề $\mathbf{A} \in \{0,1\}^{n \times n}$) thành biểu diễn tiềm ẩn $\mathbf{Z} \in R^{n \times d'}$ cho các đỉnh.

Một encoder phổ biến sử dụng mạng nơ-ron tích chập trên đồ thị (Graph Convolutional Network - GCN) như được đề xuất trong [4]. Với một tầng GCN, công thức lan truyền được định nghĩa như sau:

$$\mathbf{H}^{(l+1)} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right) \tag{16}$$

.

trong đó:

- $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_n$ là ma trận kề với liên kết tự thân,
- $\tilde{\mathbf{D}}$ là ma trận bậc tương ứng với $\tilde{\mathbf{A}}$,
- $\mathbf{W}^{(l)}$ là trọng số học được tại tầng l,
- $\sigma(\cdot)$ là hàm kích hoạt, thường là ReLU,
- $\mathbf{H}^{(0)} = \mathbf{X}$ là đặc trưng đầu vào.

Với hai tầng GCN, biểu diễn tiềm ẩn của đỉnh được tính như sau:

$$\mathbf{Z} = \text{GCN}(\mathbf{X}, \mathbf{A}) = \tilde{\mathbf{A}} \cdot \text{ReLU}(\tilde{\mathbf{A}} \cdot \mathbf{X} \cdot \mathbf{W}^{(0)}) \cdot \mathbf{W}^{(1)} \tag{17}$$

Trong trường hợp của *Variational GAE* (VGAE) [5], encoder được mở rộng để sinh ra hai đầu ra: trung bình $\boldsymbol{\mu}$ và phương sai $\boldsymbol{\sigma}$ để mô hình hóa phân phối xác suất:

$$\mathbf{z}_i = \boldsymbol{\mu}_i + \boldsymbol{\sigma}_i \odot \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}) \tag{18}$$

b. Decoder

Decoder có nhiệm vụ tái tạo ma trận kề $\hat{\mathbf{A}}$ từ biểu diễn tiềm ẩn $\mathbf{Z}$. Cách đơn giản và phổ biến nhất để thực hiện việc này là sử dụng tích vô hướng giữa các embedding:

$$\hat{A}_{ij} = \sigma(\mathbf{z}_i^\top \mathbf{z}_j) \tag{19}$$

Trong đó $\sigma(\cdot)$ là hàm sigmoid, và $\hat{A}_{ij}$ biểu diễn xác suất tồn tại cạnh giữa đỉnh i và đỉnh j.
Ngoài ra, decoder có thể được mở rộng thành mạng neural đa lớp hoặc dùng attention để tăng khả năng mô hình hóa các cấu trúc phức tạp hơn.

2.2.2 Hàm mất mát

GAE sử dụng hàm mất mát là cross-entropy giữa ma trận kề thực tế và ma trận dự đoán:

$$\mathcal{L} = - \sum_{(i,j) \in \Omega^+ \cup \Omega^-} \Big[A_{ij} \log \hat{A}_{ij} + (1 - A_{ij}) \log(1 - \hat{A}_{ij}) \Big] \tag{20}$$

trong đó:

- Ω^+ là tập các cặp đỉnh có cạnh thật,
- Ω^- là tập các cặp đỉnh không có cạnh, được chọn ngẫu nhiên (negative sampling).

Đối với mô hình VGAE, hàm mất mát được mở rộng để bao gồm thêm sai khác KL giữa phân phối ẩn và phân phối chuẩn:

$$\mathcal{L}_{\text{VGAE}} = E_{q(\mathbf{Z}|\mathbf{X},\mathbf{A})}[\log p(\mathbf{A}|\mathbf{Z})] - \text{KL}[q(\mathbf{Z}|\mathbf{X}, \mathbf{A}) \,\|\, p(\mathbf{Z})] \tag{21}$$

Trong đó $p(\mathbf{Z})$ là phân phối chuẩn hoá $\mathcal{N}(0, \mathbf{I})$ và $q(\mathbf{Z}|\mathbf{X}, \mathbf{A})$ là phân phối xác suất sinh bởi encoder.

2.2.3 Quy trình huấn luyện

Mục tiêu của GAE là tối thiểu hóa sai số tái tạo giữa ma trận kề thực A và ma trận được dự đoán $\hat{A}$. Hàm mất mát phổ biến nhất là binary cross-entropy:

$$\mathcal{L} = -\sum_{i,j} [A_{ij} \log \hat{A}_{ij} + (1 - A_{ij}) \log(1 - \hat{A}_{ij})] \tag{22}$$

Trong trường hợp đồ thị lớn và thưa, thường sử dụng sampling âm để giảm chi phí tính toán. Tập dương là các cạnh có thật trong đồ thị, trong khi tập âm là các cặp đỉnh không kết nối được lấy ngẫu nhiên.
Huấn luyện GAE bao gồm các bước:

1. Khởi tạo đặc trưng đỉnh X và ma trận kề A
2. Truyền X qua encoder để thu được biểu diễn Z
3. Tính $\hat{A}$ từ Z qua decoder
4. Tính toán hàm mất mát và cập nhật tham số qua lan truyền ngược

2.2.4 Biến thể của GAE

a. Variational Graph Autoencoder (VGAE) [5]: Thay vì học vector Z cố định, mô hình học phân phối xác suất $q(z|x)$ và lấy mẫu từ phân phối này. Hàm mất mát là ELBO:

$$\mathcal{L}_{\text{VGAE}} = E_{q(Z|X,A)}[\log p(A|Z)] - \text{KL}[q(Z|X,A)\|p(Z)] \tag{23}$$

Trong đó q là phân phối Gaussian có trung bình và phương sai học được từ GCN.
b. Adversarially Regularized GAE (ARGA) [**pan2018arga**]: Kết hợp GAE với huấn luyện đối kháng (GAN) để làm biểu diễn gần với phân phối chuẩn thông qua discriminator. Điều này giúp giảm overfitting và tăng khả năng tổng quát.
c. Marginalized GAE (MGAE) [**wang2017mgae**]: Encoder được xây dựng từ autoencoder tuyến tính, thêm nhiễu vào đầu vào để tăng tính kháng nhiễu, sau đó marginalize qua nhiễu. Phù hợp với bài toán phân cụm.
d. GraphVAE và Graphite: Các mô hình mở rộng nhằm sinh đồ thị mới từ không gian tiềm ẩn, đặc biệt phù hợp với ứng dụng sinh phân tử và thiết kế mạng.

2.2.5 Ứng dụng

Các ứng dụng điển hình của GAE bao gồm:

- **Dự đoán liên kết (Link Prediction)**: Dự đoán các cạnh tiềm năng bằng cách tính xác suất kết nối giữa các cặp đỉnh trong không gian tiềm ẩn.
- **Phân cụm/Phân loại đỉnh**: Biểu diễn Z có thể dùng làm đầu vào cho các mô hình học máy khác để thực hiện phân loại bán giám sát.
- **Sinh đồ thị (Graph Generation)**: Đặc biệt ứng dụng trong sinh phân tử, cấu trúc hóa học.
- **Phát hiện bất thường**: Những đỉnh/cạnh có lỗi tái tạo cao có thể được xem là bất thường.

2.2.6 Hạn chế và hướng phát triển

- GAE không dễ mở rộng với đồ thị lớn do tính phức tạp $O(n^2)$ khi tính tích vô hướng.
- Mô hình decoder đơn giản có thể không đủ mạnh để tái tạo cấu trúc phức tạp.
- Vấn đề "over-smoothing" trong encoder nhiều tầng vẫn là một thách thức.
- Việc mở rộng sang đồ thị dị thể, động, hoặc có thuộc tính cạnh phức tạp đang là hướng nghiên cứu mới.

Một số hướng cải tiến đang được quan tâm bao gồm:

- Sử dụng encoder có attention (GAT), hoặc Transformer cho đồ thị.
- Kết hợp học tương phản (contrastive learning) để tăng cường biểu diễn.
- Tích hợp dữ liệu thuộc tính phức tạp hoặc nhiều loại nút/cạnh.

3 Phân tích và thiết kế dự án

3.1 Phân tích dự án

3.1.1 Phân tích yêu cầu hệ thống

Hệ thống hướng tới mục tiêu hỗ trợ người dùng tra cứu, hỏi đáp pháp luật một cách tự động, nhanh chóng và chính xác. Dưới đây là các yêu cầu chính:
Yêu cầu chức năng:

- Nhận câu hỏi tiếng Việt về pháp luật từ người dùng qua giao diện web.
- Xử lý truy vấn, truy xuất thông tin pháp luật liên quan từ cơ sở dữ liệu.
- Sinh câu trả lời tự động, trả về cho người dùng.
- Chuẩn bị, thu thập, xử lý và xây dựng dữ liệu pháp luật phục vụ cho hệ thống hỏi đáp.

Yêu cầu phi chức năng:

- Thời gian phản hồi nhanh, đảm bảo trải nghiệm người dùng.
- Độ chính xác cao trong việc truy xuất và sinh câu trả lời.
- Giao diện đơn giản, dễ sử dụng, phù hợp với nhiều đối tượng.
- Hệ thống dễ dàng mở rộng, bảo trì và tích hợp thêm dữ liệu mới.

3.1.2 Phân tích luồng hoạt động chính

Quy trình hoạt động của hệ thống được mô tả như sau:

1. Giai đoạn chuẩn bị dữ liệu: Thu thập, chuẩn hóa, trích xuất thực thể/quan hệ từ văn bản pháp luật, xây dựng đồ thị tri thức (thực hiện bởi các script như make_data.py, insert_graph_db.py).
2. Giai đoạn hỏi đáp: Người dùng nhập câu hỏi về pháp luật vào giao diện web (Streamlit).
3. Giao diện gửi câu hỏi đến API backend (Flask).
4. Backend tiếp nhận câu hỏi, chuyển tới module AI để xử lý.
5. Module AI thực hiện truy xuất thông tin liên quan từ đồ thị tri thức (Neo4j) và dữ liệu văn bản (MongoDB).
6. Mô hình ngôn ngữ lớn (LLM) sinh câu trả lời dựa trên thông tin truy xuất được.
7. Backend trả về câu trả lời cho giao diện web để hiển thị cho người dùng.

3.1.3 Phân tích dữ liệu đầu vào, đầu ra và mô hình hóa dữ liệu

Dữ liệu là nền tảng quan trọng của hệ thống hỏi đáp pháp luật. Việc phân tích và mô hình hóa dữ liệu giúp đảm bảo hệ thống hoạt động hiệu quả, truy xuất và sinh câu trả lời chính xác.
Dữ liệu đầu vào:

- Câu hỏi tiếng Việt do người dùng nhập vào giao diện web, thường là các thắc mắc về quy định pháp luật, điều khoản, quyền và nghĩa vụ.
- Kho dữ liệu pháp luật gồm: văn bản luật, các bài báo liên quan, dữ liệu đã được trích xuất thực thể và quan hệ từ các nguồn này.

Dữ liệu đầu ra:

- Câu trả lời tự động, ngắn gọn, chính xác, có thể kèm theo giải thích nguồn gốc hoặc mối liên hệ giữa các điều khoản pháp luật liên quan.

Mô hình hóa dữ liệu:

- **Các thực thể chính:** Luật, Điều, Khoản, Thực thể pháp lý (ví dụ: cá nhân, tổ chức, cơ quan ban hành), Quan hệ pháp lý (ví dụ: ban hành, quy định, áp dụng).
- **Lưu trữ dữ liệu văn bản:** Các văn bản luật, bài báo, dữ liệu gốc được lưu trữ trong MongoDB dưới dạng các collection, thuận tiện cho việc truy xuất và cập nhật.
- **Đồ thị tri thức pháp luật:** Dữ liệu đã được trích xuất thực thể và quan hệ sẽ được xây dựng thành đồ thị tri thức và lưu trữ trong Neo4j. Mỗi node đại diện cho một thực thể pháp lý, mỗi edge đại diện cho một quan hệ pháp lý giữa các thực thể.

Việc mô hình hóa dữ liệu như trên giúp hệ thống dễ dàng truy xuất thông tin liên quan, hỗ trợ quá trình suy luận và sinh câu trả lời chính xác cho người dùng.

3.2 Thiết kế dự án

3.2.1 Thiết kế kiến trúc hệ thống

Hệ thống được thiết kế theo mô hình client-server với các thành phần chính như sau:

- **Module xử lý và chuẩn bị dữ liệu:** Bao gồm các script như make_data.py (thu thập, chuẩn hóa, trích xuất thực thể/quan hệ, lưu vào MongoDB) và insert_graph_db.py (xây dựng, cập nhật đồ thị tri thức trong Neo4j).
- **Giao diện web (Streamlit):** Cho phép người dùng nhập câu hỏi và hiển thị câu trả lời.
- **API backend (Flask):** Nhận câu hỏi từ giao diện, chuyển tiếp truy vấn đến các thành phần xử lý và trả về kết quả.
- **LLM1 (trích xuất thực thể/quan hệ):** Mô hình ngôn ngữ lớn đầu tiên, đảm nhiệm vai trò trích xuất thực thể và quan hệ từ câu hỏi của người dùng để hỗ trợ truy xuất thông tin chính xác trong đồ thị tri thức.
- **Mô hình Graph AutoEncoder (GAE):** Học biểu diễn (embedding) cho các đỉnh trong đồ thị tri thức pháp luật, hỗ trợ truy xuất và suy luận các quan hệ pháp lý.
- **LLM2 (sinh câu trả lời):** Mô hình ngôn ngữ lớn thứ hai, đảm nhiệm vai trò tổng hợp, diễn giải và sinh ra câu trả lời tự động, tự nhiên, dễ hiểu cho người dùng dựa trên thông tin đã truy xuất được.
- **Cơ sở dữ liệu:** Lưu trữ văn bản pháp luật, dữ liệu đã trích xuất (MongoDB) và đồ thị tri thức (Neo4j).

Mô tả luồng dữ liệu: Khi người dùng gửi câu hỏi, dữ liệu được truyền từ giao diện web đến backend, LLM1 sẽ trích xuất thực thể/quan hệ từ câu hỏi, truy xuất thông tin từ cơ sở dữ liệu, embedding và suy luận bằng GAE, sau đó LLM2 sinh câu trả lời và trả về cho người dùng. Dữ liệu nền tảng được chuẩn bị, cập nhật định kỳ qua các script xử lý dữ liệu.

3.2.2 Thiết kế chi tiết các module

Module xử lý và chuẩn bị dữ liệu:

- **make_data.py:** Thu thập dữ liệu pháp luật từ các nguồn (web, văn bản, bài báo), chuẩn hóa lưu trữ vào MongoDB hoặc các file JSON ở local.
- **insert_graph_db.py:** Đọc dữ liệu từ MongoDB hoặc các file JSON ở local, xây dựng và cập nhật đồ thị tri thức pháp luật trong Neo4j, đảm bảo dữ liệu sẵn sàng cho truy xuất AI.

Giao diện người dùng (Streamlit):

- Giao diện đơn giản với một ô nhập câu hỏi và khu vực hiển thị câu trả lời.
- Khi người dùng nhập câu hỏi và gửi, hệ thống sẽ hiển thị câu trả lời trả về từ backend.

Hình 4: Giao diện người dùng

API backend (Flask):

- Cung cấp endpoint nhận câu hỏi từ giao diện web (POST /api/chatbot).
- Nhận dữ liệu JSON chứa câu hỏi, chuyển tiếp tới LLM1, GAE, LLM2 và trả về dữ liệu JSON chứa câu trả lời.

LLM1 (trích xuất thực thể/quan hệ):

- **Vai trò:** Nhận câu hỏi từ người dùng, sử dụng mô hình ngôn ngữ lớn để trích xuất các thực thể và quan hệ quan trọng, phục vụ cho việc truy xuất thông tin chính xác trong đồ thị tri thức.
- **Luồng dữ liệu:** Đầu vào là câu hỏi của người dùng, đầu ra là danh sách thực thể và quan hệ được trích xuất.

Mô hình Graph AutoEncoder (GAE):

- **Vai trò:** Học biểu diễn (embedding) cho các đỉnh trong đồ thị tri thức pháp luật, giúp mô hình hóa quan hệ giữa các thực thể pháp lý và hỗ trợ quá trình truy xuất, suy luận.
- **Kiến trúc:** Mô hình gồm hai lớp GCN (Graph Convolutional Network) liên tiếp.
- **Luồng dữ liệu:** Đầu vào là đặc trưng các đỉnh và danh sách cạnh, đầu ra là embedding các đỉnh và dự đoán tồn tại cạnh.

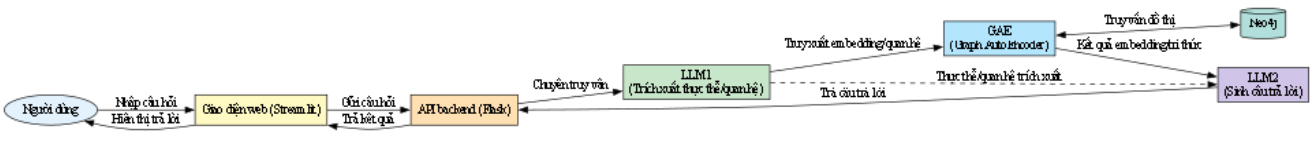
LLM2 (sinh câu trả lời):

- **Vai trò:** Sau khi truy xuất được thông tin liên quan, LLM2 tổng hợp, diễn giải và sinh ra câu trả lời tự động, tự nhiên, dễ hiểu cho người dùng.
- **Luồng dữ liệu:** Đầu vào là thông tin pháp luật đã truy xuất và câu hỏi gốc, đầu ra là câu trả lời hoàn chỉnh, mạch lạc, có thể kèm giải thích nguồn gốc hoặc mối liên hệ giữa các điều khoản pháp luật.

3.2.3 Sơ đồ luồng dữ liệu và hoạt động

Luồng dữ liệu của hệ thống được mô tả như sau:

- Giai đoạn chuẩn bị dữ liệu: Các script make_data.py và insert_graph_db.py thu thập, xử lý, xây dựng dữ liệu nền tảng cho hệ thống.
- Giai đoạn hỏi đáp: Người dùng nhập câu hỏi trên giao diện web (Streamlit), câu hỏi được gửi tới API backend (Flask), backend chuyển câu hỏi tới module AI để truy xuất thông tin và sinh câu trả lời, kết quả trả về giao diện web.



Hình 5: Sơ đồ luồng dữ liệu của hệ thống hỏi đáp pháp luật

3.2.4 Các quyết định thiết kế và lý do lựa chọn

- Sử dụng Python vì tính linh hoạt, thư viện AI mạnh và cộng đồng hỗ trợ lớn.
- PyTorch được lựa chọn để xây dựng và huấn luyện các mô hình AI như GAE, LLM nhờ khả năng tùy biến và hiệu năng tốt.
- Flask phù hợp để xây dựng API backend đơn giản, dễ tích hợp với các thành phần khác.
- Streamlit giúp phát triển giao diện web nhanh chóng, thân thiện với người dùng.
- MongoDB lưu trữ dữ liệu phi cấu trúc (văn bản, bài báo), còn Neo4j phù hợp cho dữ liệu đồ thị tri thức pháp luật.
- Việc kết hợp LLM, GraphRAG, GAE giúp hệ thống tăng khả năng hiểu ngữ nghĩa, truy xuất và suy luận pháp lý, nâng cao chất lượng câu trả lời.
- Các script xử lý dữ liệu giúp hệ thống luôn có dữ liệu nền tảng sạch, có cấu trúc, sẵn sàng cho truy xuất và hỏi đáp.

4 Cài đặt và kiểm thử

4.1 Cấu trúc mã nguồn

```
thuc_tap_co_so/
 baocao.tex
 requirements.txt //các thư viện cần thiết
 README.md //tổng quan về dự án
 make_data/
  make_data.py //thu thập, chuẩn hóa, trích xuất thực thể/quan hệ, lưu vào MongoDB
  insert_graph_db.py //đọc dữ liệu từ MongoDB hoặc các file JSON ở local, xây dựng và cập nhật đồ thi
  data/ //dữ liệu đã được chuẩn hóa và trích xuất
 src/
  ai_model/
   gae/
    train.py //train model gae
    test.py //test model gae
    model.py //model gae
    gae.torch //model gae đã được train
   llm/
    model.py //model llm call api gemini về
   retrieve.py //khối truy xuất thông tin từ graphdb
  web_app/
  stremlit_web.py //giao diện web
  back_end/
  chatbot_api.py //api backend
```

4.2 Cài đặt và train model GAE

4.2.1 Cài đặt model GAE

Mô hình Graph AutoEncoder (GAE) được xây dựng bằng thư viện PyTorch Geometric, sử dụng hai lớp GCN để học biểu diễn (embedding) cho các đỉnh trong đồ thị tri thức pháp luật. **Kiến trúc mô hình:**

- **Lớp encoder1:** GCNConv đầu vào, ánh xạ từ số chiều đặc trưng ban đầu (`input_dim`) sang không gian ẩn (`hidden_dim`).
- **Lớp encoder2:** GCNConv tiếp theo, ánh xạ từ không gian ẩn sang không gian embedding (`embedding_dim`).
- **Hàm encode:** Truyền dữ liệu qua hai lớp GCN, sử dụng hàm kích hoạt ReLU ở lớp đầu.
- **Hàm decode:** Dự đoán khả năng tồn tại cạnh giữa hai đỉnh dựa trên tích vô hướng của embedding.
- **Hàm forward:** Kết hợp encode và decode để huấn luyện và đánh giá mô hình.

Tham số cấu hình:

- `input_dim`: Số chiều đặc trưng đầu vào của mỗi node. Trong hệ thống này, sử dụng one-hot encoding nên `input_dim` = số lượng node trong đồ thị.
- `hidden_dim`: Số chiều không gian ẩn, thường đặt là 16.
- `embedding_dim`: Số chiều vector embedding đầu ra, thường đặt là 8.
- `epochs`: Số vòng lặp huấn luyện, thường đặt là 200.
- `optimizer`: Adam, learning rate 0.01.

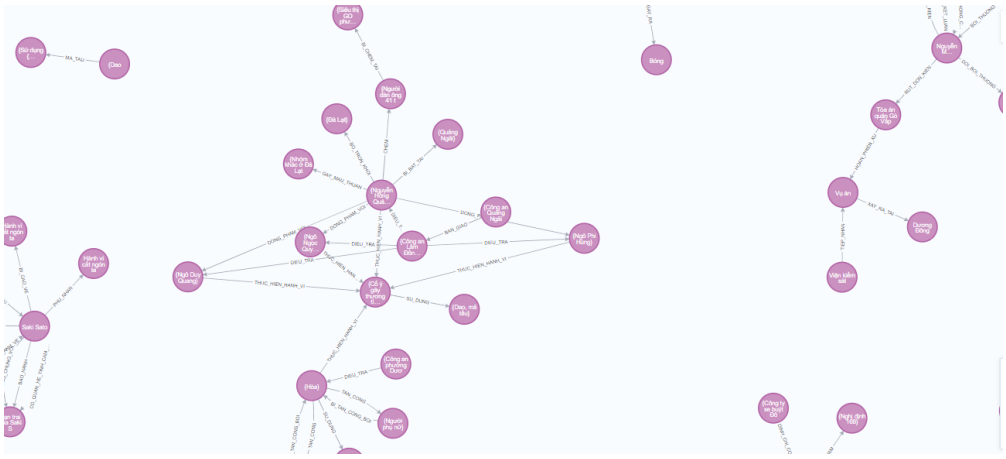
Ví dụ khởi tạo mô hình:

```
input_dim = features.size(1) # S node (one-hot)
hidden_dim = 16
embedding_dim = 8
model = GAE(input_dim, hidden_dim, embedding_dim)
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
```

4.2.2 Train model GAE

Dữ liệu đầu vào cho quá trình huấn luyện mô hình GAE gồm:

- **Số node:** 3000 (mỗi node đại diện cho một thực thể pháp lý, được mã hóa one-hot).
- **Số cạnh (edge):** 4000 (mỗi cạnh đại diện cho một quan hệ pháp lý giữa hai thực thể).



Hình 6: Dữ liệu train dạng đồ thị

Chia tập train/validation:

- Toàn bộ 4000 cạnh được trộn ngẫu nhiên.
- 80% số cạnh (3200 cạnh) dùng cho huấn luyện (train), 20% còn lại (800 cạnh) dùng cho kiểm thử (validation).
- Việc chia này đảm bảo các cạnh trong tập train và validation không trùng lặp, giúp đánh giá khả năng tổng quát hóa của mô hình.

Quy trình huấn luyện:

- Sử dụng PyTorch Geometric để xây dựng và huấn luyện mô hình GAE với các tham số:
  - `input_dim` = 3000 (số node, do dùng one-hot encoding)
  - `hidden_dim` = 16
  - `embedding_dim` = 8
  - `epochs` = 200
  - `optimizer`: Adam, learning rate 0.01
- Trong mỗi epoch, mô hình được huấn luyện trên tập train và đánh giá trên tập validation. Loss được tính bằng binary cross-entropy giữa dự đoán tồn tại cạnh và ground truth.
- Sau khi huấn luyện xong, trọng số mô hình được lưu lại để sử dụng cho các tác vụ truy xuất embedding và suy luận pháp lý.

Kết quả huấn luyện:

- Sau 200 epoch, loss trên tập train và validation đều đạt giá trị rất nhỏ (~5×10^{-12} cho train, ~8×10^{-6} cho validation), cho thấy mô hình đã học rất tốt cấu trúc đồ thị tri thức pháp luật.
- Điều này chứng tỏ embedding học được có khả năng tái tạo lại ma trận kề của đồ thị với độ chính xác cao, hỗ trợ hiệu quả cho các tác vụ truy xuất và suy luận sau này.

Bảng: Giá trị loss trong quá trình huấn luyện GAE

Epoch	Train Loss	Validation Loss
1	0.693	0.692
100	1.2×10^{-3}	2.1×10^{-3}
200	5.0×10^{-12}	8.2×10^{-6}

Bảng 2: Giá trị loss trên tập train và validation ở các mốc epoch tiêu biểu

Nhận xét: Loss trên cả tập train và validation đều giảm rất mạnh qua các epoch, đạt giá trị gần như tiệm cận về 0 ở cuối quá trình huấn luyện. Điều này cho thấy mô hình đã học rất tốt cấu trúc đồ thị tri thức pháp luật và có khả năng tái tạo lại ma trận kề với độ chính xác cao.

4.3 Kiểm thử hệ thống

4.3.1 Kiểm thử module LLM1 (trích xuất thực thể/quan hệ)

Mục tiêu: Đảm bảo LLM1 nhận diện đúng các thực thể và quan hệ pháp lý từ câu hỏi tiếng Việt tự nhiên. **Cách kiểm thử:**

- Nhập các câu hỏi pháp luật thực tế và đa dạng về cấu trúc.
- So sánh thực thể/quan hệ trích xuất được với đáp án chuẩn (do chuyên gia hoặc đối chiếu văn bản luật).

Kết quả mong đợi: LLM1 trích xuất đúng tên thực thể (ví dụ: "Bộ luật Lao động 2019", "Điều 34"), đúng loại quan hệ ("quy định", "ban hành", ...). **Ví dụ:**

- **Câu hỏi:** "Người lao động nghỉ thai sản được hưởng chế độ gì?"
- **Thực thể trích xuất:** "người lao động", "nghỉ thai sản"
- **Quan hệ trích xuất:** "hưởng chế độ"

4.3.2 Kiểm thử module GAE (embedding và truy xuất)

Mục tiêu: Đảm bảo mô hình GAE học embedding tốt, truy xuất đúng các thực thể/quan hệ liên quan trong đồ thị tri thức. **Cách kiểm thử:**

- Đánh giá khả năng dự đoán cạnh mới (link prediction) trên tập validation.
- Truy xuất các thực thể liên quan dựa trên embedding, kiểm tra kết quả có hợp lý không.

Kết quả mong đợi: GAE dự đoán đúng các cạnh thực tế, embedding các node cùng nhóm pháp lý gần nhau trong không gian vector. **Ví dụ:**

- **Truy vấn:** "Điều 34 Bộ luật Lao động 2019"
- **Kết quả embedding:** Các node "nghỉ thai sản", "trợ cấp", "người lao động nữ"có embedding gần nhau.
- **Dự đoán cạnh:** GAE dự đoán đúng cạnh "Điều 34-quy định–> "nghỉ thai sản"

4.3.3 Kiểm thử module LLM2 (sinh câu trả lời)

Mục tiêu: Đảm bảo LLM2 sinh ra câu trả lời tự động, tự nhiên, đúng ngữ cảnh và có thể giải thích nguồn gốc pháp lý. **Cách kiểm thử:**

- Nhập các câu hỏi thực tế, kiểm tra câu trả lời sinh ra có đúng nội dung, rõ ràng, dễ hiểu không.
- Đối chiếu câu trả lời với văn bản luật gốc.

Kết quả mong đợi: Câu trả lời đúng, mạch lạc, có thể kèm trích dẫn điều luật hoặc giải thích nguồn gốc. **Ví dụ:**

- **Câu hỏi:** "Người lao động nghỉ thai sản được hưởng chế độ gì?"
- **Câu trả lời:** "Theo Điều 34 Bộ luật Lao động 2019, người lao động nữ được nghỉ thai sản 6 tháng và hưởng trợ cấp bằng 100% mức lương đóng BHXH..."

4.3.4 Kiểm thử API backend (Flask)

Mục tiêu: Đảm bảo API nhận câu hỏi, chuyển qua pipeline LLM1 → GAE → LLM2, trả về kết quả đúng định dạng. **Cách kiểm thử:**

- Gửi request POST tới endpoint /api/chatbot với nhiều câu hỏi khác nhau.
- Kiểm tra response trả về có trường "answer"đúng định dạng JSON.
- Kiểm tra thời gian phản hồi.

Kết quả mong đợi: API trả về câu trả lời đúng, không lỗi, thời gian phản hồi nhanh ($\leqslant$ 30 giây). **Ví dụ:**

- **Request:**

```
{"question": "Doanh nghiep co bat buoc phai dong bao hiem that nghiep khong?"}
```

- **Response:**

```
{"answer": "Theo dieu 43 Luat Viec lam 2013, doanh nghiep su dung tu 10 lao dong tro len bat buoc phai
    tham gia bao hiem that nghiep..."}
```

4.4 Thử nghiệm

Hình 7: Thử nghiệm hệ thống

5 Kết luận

Trong khuôn khổ đồ án, bản thân đã xây dựng thành công một hệ thống AI hỗ trợ hỏi đáp pháp luật tiếng Việt dựa trên các công nghệ hiện đại như Graph Retrieval-Augmented Generation (GraphRAG), Graph AutoEncoder (GAE), mô hình ngôn ngữ lớn (LLM), kết hợp với cơ sở dữ liệu đồ thị Neo4j và MongoDB. Hệ thống cho phép người dùng nhập câu hỏi tự nhiên, tự động trích xuất thực thể/quan hệ, truy xuất tri thức pháp luật và sinh câu trả lời mạch lạc, dễ hiểu.

Ý nghĩa thực tiễn:

- Hệ thống giúp người dân, doanh nghiệp tra cứu thông tin pháp luật nhanh chóng, chính xác, giảm tải cho các cơ quan tư vấn truyền thống.
- Ứng dụng các mô hình AI hiện đại vào lĩnh vực pháp lý, góp phần thúc đẩy chuyển đổi số ngành luật tại Việt Nam.

Kết quả đạt được:

- Xây dựng pipeline xử lý dữ liệu, xây dựng graphdb, huấn luyện thành công mô hình GAE với loss rất nhỏ, cho thấy khả năng học tốt cấu trúc đồ thị tri thức pháp luật.
- Tích hợp thành công hai agent LLM cho trích xuất thực thể/quan hệ và sinh câu trả lời, đảm bảo pipeline hỏi đáp tự động, mạch lạc.
- Hệ thống backend (Flask) và giao diện web (Streamlit) hoạt động ổn định, dễ sử dụng, thời gian phản hồi nhanh.

Hạn chế:

- Dữ liệu pháp luật còn hạn chế về phạm vi, chủ yếu tập trung vào một số bộ luật lớn.
- Chưa xử lý sâu các trường hợp truy vấn phức tạp, đa bước hoặc yêu cầu suy luận logic cao.
- Chưa tích hợp kiểm thử tự động và đánh giá định lượng chất lượng câu trả lời trên tập kiểm thử lớn.

Hướng phát triển:

- Mở rộng kho dữ liệu pháp luật, cập nhật thêm nhiều bộ luật, nghị định, thông tư mới.
- Nâng cấp mô hình LLM và GAE, thử nghiệm các kiến trúc tiên tiến hơn (GAT, Graph Transformer, LLM đa ngữ).
- Tích hợp kiểm thử tự động, đánh giá định lượng và cải thiện giao diện người dùng.
- Phát triển thêm các tính năng giải thích nguồn gốc pháp lý, truy vết logic và hỗ trợ đa nền tảng (mobile, chatbot).

Nhìn chung, hệ thống đã chứng minh được tiềm năng ứng dụng AI vào lĩnh vực pháp luật, là nền tảng quan trọng để phát triển các hệ thống hỏi đáp pháp lý thông minh, hỗ trợ người dân và doanh nghiệp trong thời đại chuyển đổi số.

Tài liệu

[1] T. B. Brown, B. Mann, N. Ryder **andothers**. "Language Models are Few-Shot Learners". **in***arXiv preprint arXiv:2005.14165*: (2020). URL: https://arxiv.org/abs/2005.14165.

[2] X. Chen, Y. Wang **and** Z. Zhang. "Retrieval-Augmented Generation with Graphs: A Comprehensive Survey". **in***arXiv preprint arXiv:2501.00309v2*: (2024). URL: https://arxiv.org/abs/2501.00309.

[3] J. Devlin **andothers**. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". **in***arXiv preprint arXiv:1810.04805*: (2019). URL: https://arxiv.org/abs/1810.04805.

[4] Thomas N. Kipf **and** Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". **in***International Conference on Learning Representations (ICLR)*: arXiv:1609.02907. 2017. URL: https://arxiv.org/abs/1609.02907.

[5] Thomas N. Kipf **and** Max Welling. "Variational Graph Auto-Encoders". **in***NIPS Workshop on Bayesian Deep Learning*: arXiv:1611.07308. 2016. URL: https://arxiv.org/abs/1611.07308.

[6] P. Lewis **andothers**. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". **in***Advances in Neural Information Processing Systems*: 33 (2020), **pages** 9459–9474.

[7] Y. Li **and** W. Zhao. "Graph-based Information Organization for Retrieval-Augmented Generation". **in***arXiv preprint arXiv:2305.12345*: (2023). URL: https://arxiv.org/abs/2305.12345.

[8] H. Liu, T. Zhou **and** Q. Zhang. "Graph-based Knowledge Representation: A Survey". **in***IEEE Transactions on Knowledge and Data Engineering*: 35.4 (2023), **pages** 1234–1256.

[9] C. Raffel **andothers**. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". **in***Journal of Machine Learning Research*: 21.140 (2020), **pages** 1–67.

[10] Z. Wu **andothers**. "A Comprehensive Survey on Graph Neural Networks". **in***IEEE Transactions on Neural Networks and Learning Systems*: 32.1 (2020), **pages** 4–24.

[11] L. Zhang **and** X. Li. "Efficient Indexing for Large-Scale Graph Databases". **in***Proceedings of the VLDB Endowment*: 15.3 (2022), **pages** 567–579.