

**ĐẠI HỌC ĐÀ NẴNG**  
**TRƯỜNG ĐẠI HỌC BÁCH KHOA**  
**KHOA CÔNG NGHỆ THÔNG TIN**



**ĐỒ ÁN CHUYÊN NGÀNH**

**ĐỀ TÀI :**

**Hệ thống AI hỗ trợ hướng dẫn thủ tục hành  
chính cấp phường, xã TP Đà Nẵng**

Họ tên sinh viên	Mã sinh viên	Nhóm HP
Trần Đình Thắng	102210186	21.15
Lê Quốc Vinh	102210335	21.15

**GVHD : PGS.TS Nguyễn Tấn Khôi**

*Đà Nẵng, 09/06/2025*

**BẢNG PHÂN CÔNG CÔNG VIỆC**

TT	Công việc	Người thực hiện
1	Xây dựng hệ thống RAG	Thắng
2	Fine-tune model	Thắng
3	Backend AI	Thắng
4	Xây dựng data	Vinh
5	Frontend	Vinh
6	Backend Web	Vinh

## DANH SÁCH HÌNH ẢNH

Hình 1: Hệ thống hỏi đáp tự động nhằm xây dựng, hỗ trợ và phát triển tính năng giao tiếp của Trí tuệ nhân tạo AI.....	3
Hình 2: Quy trình làm việc cơ bản của hệ thống RAG .....	5
Hình 3: Ensemble Retriever và Reranking.....	8
Hình 4: Kiến trúc Transformer .....	11
Hình 5: Kiến trúc Decoder-only Transformer .....	13
Hình 6: Instruction fine-tuning.....	14
Hình 7: Minh họa tinh chỉnh LLM với dữ liệu đặc thù.....	15
Hình 8: Langchain Framework.....	17
Hình 9: Streamlit .....	18
Hình 10: FastAPI .....	19
Hình 11: Tương tác với API docs tự động .....	21
Hình 12: Nguồn dữ liệu – Dịch vụ công TP Đà Nẵng .....	22
Hình 13: Dữ liệu fine-tune (Hugging Face) .....	24
Hình 14: Cấu hình phần cứng huấn luyện .....	26
Hình 15: Kết quả huấn luyện train/loss .....	27
Hình 16: Kết quả huấn luyện train/learning_rate .....	28
Hình 17: Kết quả huấn luyện train/grad_norm.....	28
Hình 18: Kiến trúc hệ thống .....	32
Hình 19: Kết quả - Giao diện đăng nhập.....	36
Hình 20: Kết quả - Giao diện cập nhật dữ liệu.....	37
Hình 21: Kết quả - Giao diện hỏi đáp hướng dẫn .....	37
Hình 22: Kết quả - Giao diện xem trích dẫn .....	38
Hình 23: Kết quả - Giao diện cung cấp đường link .....	38

# MỤC LỤC

MỞ ĐẦU .....	1
TỔNG QUAN ĐỀ TÀI .....	1
Lý do chọn đề tài: .....	1
Mục tiêu chung của đề tài:.....	1
CHƯƠNG 1: CƠ SỞ LÝ THUYẾT.....	3
1.1.    Tổng quan về hệ thống hỏi đáp .....	3
1.2.    Retrieval-Augmented Generation (RAG).....	4
1.2.1.    Ingestion .....	5
1.2.2.    Chunking .....	6
1.2.3.    Embeddings .....	6
1.2.4.    Ensemble Retriever và Reranking .....	7
1.2.5.    Response Generation / Synthesis.....	8
1.3.    Mô hình ngôn ngữ lớn (LLM).....	8
1.3.1.    LLM sử dụng trong hệ thống.....	8
1.3.2.    Tinh chỉnh LLM (Fine-tuning) .....	14
1.4.    LangChain Framework .....	17
1.5.    Streamlit .....	18
1.5.1.    Đặc điểm nổi bật của Streamlit: .....	18
1.5.2.    Cấu trúc cơ bản của một ứng dụng Streamlit: .....	18
1.6.    FastAPI.....	19
1.6.1.    Cài đặt.....	20
1.6.2.    Tạo file main và chạy ứng dụng .....	20
1.6.3.    Tương tác với API docs tự động .....	21
1.7.    Kết chương .....	21
CHƯƠNG 2: GIẢI PHÁP ĐỀ XUẤT.....	22
2.1 Tập dữ liệu.....	22
2.1.1 Dữ liệu truy xuất.....	22
2.1.2 Dữ liệu fine-tune.....	24
2.2 Mô hình .....	25
2.2.1 Cấu hình tham số huấn luyện LoRA .....	25
2.2.2 Cấu hình phần cứng.....	26
2.2.3 Kết quả huấn luyện.....	27
CHƯƠNG 3: PHÂN TÍCH THIẾT KẾ HỆ THỐNG .....	29
3.1    Phát biểu bài toán .....	29
3.1.1 Mục tiêu.....	29
3.1.2 Bài toán cần giải quyết .....	29
3.1.3 Tiêu chí đánh giá: .....	29

3.2 Phân tích hiện trạng .....	29
3.2.1. Đối tượng sử dụng .....	30
3.2.2 Chức năng.....	30
3.2.3 Công nghệ sử dụng .....	31
3.3 Thiết kế hệ thống .....	32
3.3.2 Sơ đồ tổng thể.....	32
3.3.3 Mô tả chi tiết.....	32
3.4 Xây dựng chương trình.....	33
3.4.2 Tổ chức thư mục .....	33
3.4.3 Frontend.....	33
3.4.4 Backend .....	33
3.5 Kết chương .....	34
CHƯƠNG 4: TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ .....	35
3.1. Triển khai chương trình .....	35
3.1.1. Môi trường triển khai.....	35
3.1.2. Cấu hình hệ thống.....	35
3.2. Kết quả thực nghiệm.....	36
3.2.1. Đăng nhập hệ thống .....	36
3.2.2. Chức năng cập nhật dữ liệu (Đang hoàn thiện) .....	36
3.2.3. Chức năng hỏi đáp, hướng dẫn.....	37
3.2.4. Chức năng cung cấp đường link , trích dẫn tài liệu liên quan .....	38
KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN.....	39
1. KẾT QUẢ ĐẠT ĐƯỢC .....	39
2. KIẾN NGHỊ VÀ HƯỚNG PHÁT TRIỂN .....	40
TÀI LIỆU THAM KHẢO .....	42

# MỞ ĐẦU

## TỔNG QUAN ĐỀ TÀI

### Lý do chọn đề tài:

Trong những năm gần đây, cải cách hành chính và chuyển đổi số là hai xu hướng song hành trong các chương trình phát triển của Chính phủ Việt Nam. Một trong những mục tiêu trọng tâm là nâng cao khả năng tiếp cận và hiểu biết của người dân, doanh nghiệp đối với thủ tục hành chính (TTHC), thông qua các kênh tự động và thân thiện. Tuy nhiên, hệ thống cung cấp thông tin Thủ Tục Hành Chính hiện nay vẫn còn nhiều hạn chế như:

- Nội dung trình bày khó hiểu, sử dụng ngôn ngữ pháp lý.
- Thiếu tương tác linh hoạt theo ngữ cảnh người hỏi.
- Không tích hợp khả năng liên kết đến các văn bản pháp luật liên quan.
- Nhiều câu hỏi lặp lại hoặc bị hiểu sai do thiếu khả năng xử lý ngôn ngữ tự nhiên.

Từ thực tiễn đó, nhóm đề tài phát triển hệ thống “Chuyển hướng: Hỏi đáp thủ tục hành chính” nhằm tích hợp trí tuệ nhân tạo (AI) với ngôn ngữ tự nhiên tiếng Việt, giúp người dùng truy xuất thông tin nhanh chóng, chính xác, có dẫn nguồn và có khả năng mở rộng.

### Mục tiêu chung của đề tài:

- Xây dựng một hệ thống hỏi đáp thủ tục hành chính ứng dụng mô hình ngôn ngữ lớn (LLM), có khả năng:
- Hiểu câu hỏi tiếng Việt tự nhiên.
- Trả lời theo từng bước, có dẫn nguồn văn bản pháp lý liên quan.
- Hiện thị rõ ràng, dễ đọc, có thể trích xuất hoặc in ấn.
- Cung cấp nền tảng ban đầu để mở rộng sang nhiều lĩnh vực thủ tục hành chính khác.
- Kiểm chứng khả năng cải thiện độ chính xác và thân thiện của mô hình thông qua tinh chỉnh (fine-tuning) trên mô hình VisStar.

### **Phạm vi thực hiện:**

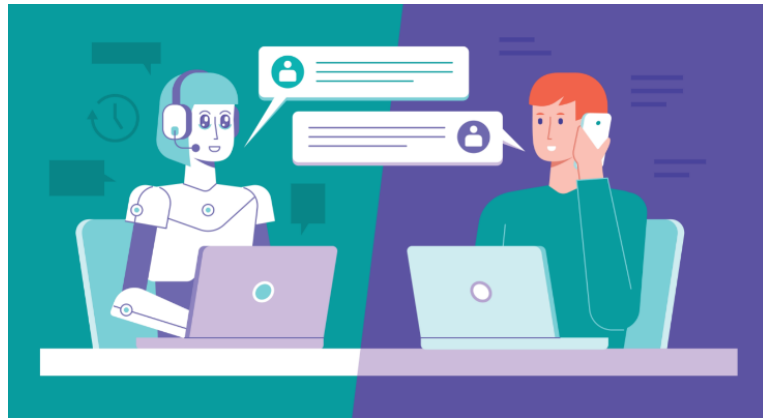
Hệ thống được triển khai thử nghiệm trong phạm vi cấp phường, cụ thể:

- Về dữ liệu thủ tục: Gồm 175 nhóm thủ tục hành chính cấp phường, xã được lưu trữ, định dạng rõ các mục trong file word.
- Về công nghệ triển khai: Mô hình nền VisStar: Mô hình ngôn ngữ tiếng Việt do AI4VN phát triển, có khả năng hiểu và sinh ngôn ngữ tiếng Việt kèm với FE (Frontend) và BE (Backend) giúp tạo giao diện người dùng đơn giản, dễ sử dụng, hệ thống backend đảm bảo hiệu suất xử lý cao.

# CHƯƠNG 1: CƠ SỞ LÝ THUYẾT

## 1.1. Tổng quan về hệ thống hỏi đáp

Hệ thống hỏi đáp (Question Answering - QA) là một lĩnh vực quan trọng trong xử lý ngôn ngữ tự nhiên (NLP), cho phép người dùng đặt câu hỏi bằng ngôn ngữ tự nhiên và nhận được câu trả lời phù hợp từ một nguồn dữ liệu cụ thể. Trong thời đại của trí tuệ nhân tạo và mô hình ngôn ngữ lớn, hệ thống hỏi đáp có thể được xây dựng trên nền tảng các mô hình tạo sinh mạnh mẽ như GPT, BERT, T5,... kết hợp với cơ chế truy xuất ngữ nghĩa.



*Hình 1: Hệ thống hỏi đáp tự động nhằm xây dựng, hỗ trợ và phát triển tính năng giao tiếp của Trí tuệ nhân tạo AI.*

Hiện nay, một hướng tiếp cận hiệu quả là sử dụng kiến trúc **RAG (Retrieval-Augmented Generation)**, trong đó mô hình ngôn ngữ lớn được cung cấp thêm ngữ cảnh truy xuất từ nguồn dữ liệu đầu vào như văn bản, tài liệu, PDF, v.v., từ đó nâng cao độ chính xác và phù hợp của câu trả lời.

Trong hệ thống được xây dựng, các thành phần chính bao gồm:

- Bộ tải và xử lý tài liệu (PDF)
- Bộ tạo vector nhúng (embedding) văn bản
- Cơ sở dữ liệu vector Qdrant để lưu trữ và truy xuất ngữ nghĩa
- Mô hình reranker để tối ưu kết quả tìm kiếm



- Mô hình ngôn ngữ lớn sinh câu trả lời
- Framework LangChain để kết nối các thành phần trên

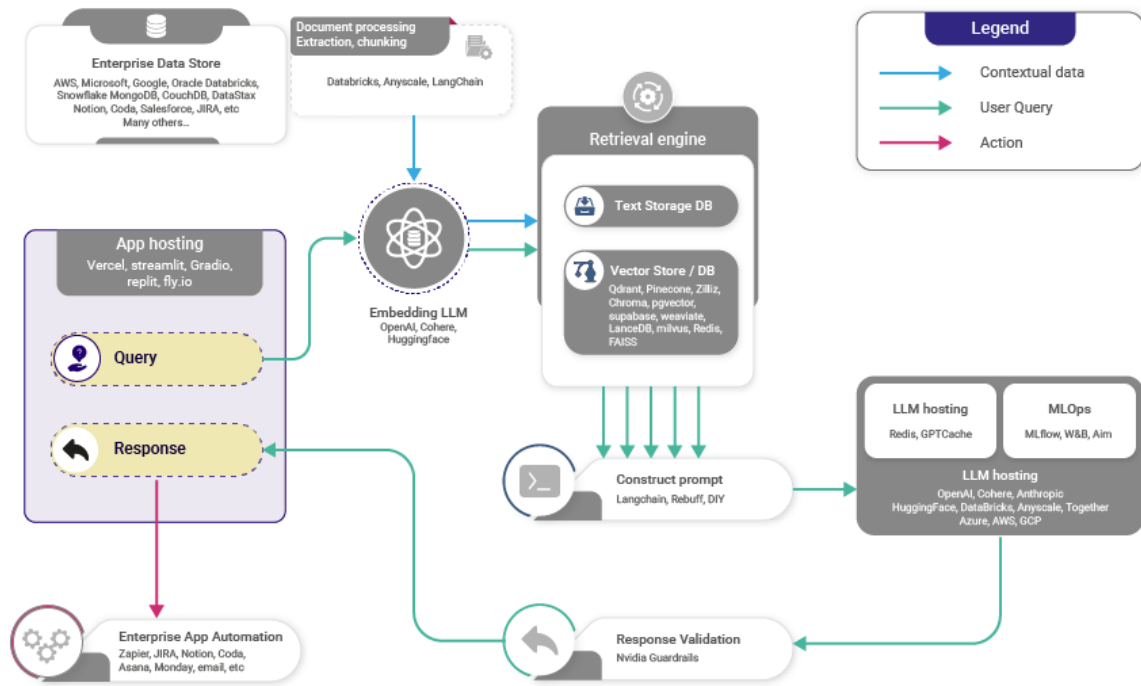
## 1.2. Retrieval-Augmented Generation (RAG)

**Retrieval-Augmented Generation (RAG)** là kiến trúc kết hợp giữa hai kỹ thuật:

- **Truy xuất (retrieval):** Tìm kiếm các đoạn văn bản liên quan đến truy vấn từ cơ sở dữ liệu vector embedding.
- **Tạo sinh (generation):** Dựa trên truy vấn và các đoạn ngữ cảnh, mô hình LLM sinh câu trả lời phù hợp.

Quy trình cụ thể:

1. **Create Vector database:** Đầu tiên, convert toàn bộ dữ liệu tri thức thành các vector và lưu trữ chúng vào một vector database.
2. **User input:** User cung cấp 1 câu truy vấn (query) bằng ngôn ngữ tự nhiên nhằm tìm kiếm câu trả lời hoặc để hoàn thành câu truy vấn đó.
3. **Information retrieval:** Cơ chế retrieval quét toàn bộ vector trong database để xác định các phân đoạn tri thức (chính là paragraphs) nào có ngữ nghĩa tương đồng với câu truy vấn của người dùng. Các paragraphs này sau đó được vào LLM để làm tăng context cho quá trình sinh ra câu trả lời.
4. **Combining data:** Các paragraphs được lấy sau quá trình retrieval từ database được kết hợp với câu query ban đầu của user tạo thành 1 câu prompt.
5. **Generate text:** Câu prompt được bổ sung thêm context sau đó được đưa qua LLM để sinh ra câu phản hồi cuối cùng theo context bổ sung.



Hình 2: Quy trình làm việc cơ bản của hệ thống RAG

RAG mang lại ưu điểm vượt trội:

- Với RAG, LLM có thể tận dụng dữ liệu bên ngoài để cung cấp tri thức cho nó.
- RAG không yêu cầu training lại mô hình, tiết kiệm thời gian và tài nguyên tính toán.
- Nó hiệu quả ngay cả với một lượng dữ liệu gán nhãn hạn chế.
- Tuy nhiên, RAG cũng có nhược điểm đó là hiệu suất của RAG phụ thuộc vào chất lượng độ chính xác của model retrieval; tính toàn diện và chính xác của kho tri thức có sẵn.
- RAG phù hợp nhất cho các tình huống có nhiều dữ liệu chưa được gán nhãn (unlabeled-data) nhưng nguồn dữ liệu gán nhãn khan hiếm và lý tưởng cho các ứng dụng như trợ lý ảo cần truy cập theo thời gian thực vào thông tin cụ thể như hướng dẫn sử dụng phần mềm, tin tức, ...

### 1.2.1. Ingestion

Ingestion là quá trình có thể hiểu chung là biến đổi dữ liệu, về cơ bản nó gồm các quá trình:

- Thu thập dữ liệu

- Tiền xử lý dữ liệu
- Lập chỉ mục và lưu trữ vào database (Indexing/ Embedding/ Storage DB)

### 1.2.2. Chunking

- Chunking (phân đoạn) là quá trình chia prompts hoặc documents thành nhiều chunks, hoặc gọi là segments nhỏ hơn nhưng vẫn có nghĩa. Các chunks này có thể được cắt theo một kích thước cố định, chẳng hạn như số lượng ký tự, câu hoặc đoạn văn cụ thể.
- Trong RAG, mỗi chunk được mã hóa (encode) thành 1 vector embedding để retrieval. Phân tách các chunks đủ nhỏ, chính xác sẽ giúp kết quả truy xuất thông tin giữa câu truy vấn của user và nội dung chunk chính xác hơn.
- Việc phân tách chunks quá lớn có thể bao gồm nhiều thông tin không liên quan, gây nhiễu và có khả năng làm giảm độ chính xác của retrieval. Bằng các việc kiểm soát kích thước của các chunks, RAG có thể cân bằng giữa lượng thông tin đầy đủ và tính chính xác.

### 1.2.3. Embeddings

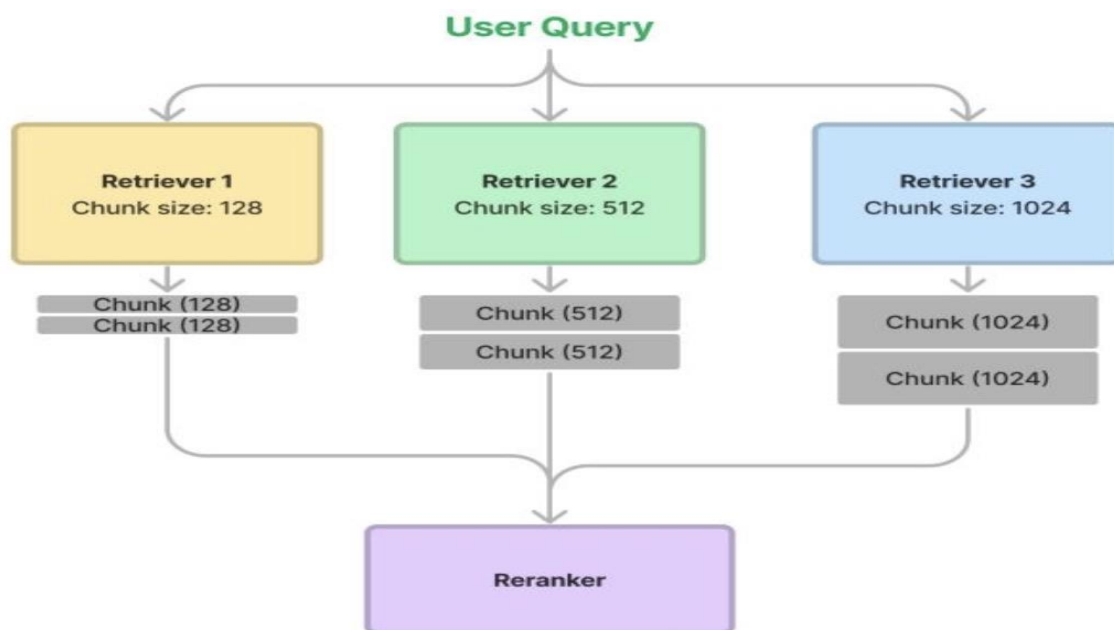
- Sau khi chúng ta chunking documents, bước tiếp theo là embed chúng. Embed documents trong RAG tức là biến đổi cả câu query của user và các documents trong kho tri thức thành một định dạng, một không gian vector để có thể so sánh về mức độ liên quan giữa chúng. Quá trình này rất quan trọng đối với khả năng của RAG trong việc truy xuất thông tin phù hợp nhất từ kho tri thức với câu query của user.
- Để lựa chọn mô hình embedding phù hợp cho bài toán của chúng ta cũng là cần thiết. Chúng ta cùng phân tích xem lợi ích của từng loại.
  - **Sparse embedding:** Sparse embedding chẳng hạn như TF-IDF rất phù hợp để so khớp (matching) từ vựng giữa câu query và documents. Ứng dụng tốt nhất cho trường hợp là mức độ liên quan giữa các từ khóa

(keyword) là rất quan trọng. Nó ít đòi hỏi chi phí tính toán chuyên sâu hơn nhưng có thể không nắm bắt được ngữ nghĩa sâu sắc.

- **Semantic embedding:** Semantic embedding/ Dense embedding chẳng hạn như các mô hình transformers-based: BERT, SentenceBERT, ... thường được sử dụng nhiều do khả năng nắm bắt ngữ nghĩa của câu. Loại này thường được sử dụng nhiều trong hệ thống RAG.

#### 1.2.4. Ensemble Retriever và Reranking

- Chúng ta có thể thử nhiều kích thước chunk cùng 1 lúc, và sử dụng phương pháp ranking nào đấy để lấy kết quả cuối cùng. Cách thức này có 2 mục đích:
  - Kết quả retrieval tốt hơn bằng cách gộp các quả từ nhiều kích thước chunk khác nhau, và ranking lại chúng một cách hiệu quả.
  - Đây cũng là 1 cách để benchmark được các chiến lược retrieval khác nhau dựa trên sự thay đổi kích thước chunk.
- Quá trình này sẽ như sau:
  - Chia nhỏ cùng một tài liệu theo nhiều cách khác nhau, ví dụ như với các kích thước chunk: 128, 256, 512 và 1024.
  - Trong quá trình retrieval, chúng ta tìm các chunk có liên quan từ mỗi bộ retriever, sau đó tập hợp chúng lại với nhau để truy xuất.
  - Cuối cùng, sử dụng bộ ranker để rank lại kết quả



Hình 3: Ensemble Retriever và Reranking

### 1.2.5. Response Generation / Synthesis

Bước cuối cùng trong hệ thống RAG là quá trình tạo ra phản hồi cho người dùng. Trong bước này, mô hình ngôn ngữ lớn (LLM) sẽ kết hợp thông tin được truy xuất từ kho dữ liệu bên ngoài với tri thức có sẵn trong nó (pre-trained knowledge) để tạo ra các phản hồi mạch lạc và phù hợp với ngữ cảnh. Quá trình này đòi hỏi khả năng tích hợp các hiểu biết từ nhiều nguồn khác nhau, đảm bảo tính chính xác, mức độ liên quan và sự tự nhiên trong câu trả lời. Phản hồi tạo ra không chỉ cung cấp thông tin chính xác mà còn duy trì giọng điệu trò chuyện, phản ánh sự hiểu biết sâu sắc về truy vấn của người dùng.

Trong chương tiếp theo, chúng ta sẽ tìm hiểu rõ hơn về vai trò và cơ chế hoạt động của LLM trong hệ thống RAG, làm rõ cách thức mà các mô hình này xử lý ngôn ngữ tự nhiên và tạo ra phản hồi chất lượng cao.

## 1.3. Mô hình ngôn ngữ lớn (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 nơ-ron, 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 chuyển đổi từng từ trong văn bản đầu vào thành biểu diễn vector nhiều chiều (high-dimensional). Những vec-tơ 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 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 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 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.

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 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.

### 1.3.1. LLM sử dụng trong hệ thống

Hệ thống sử dụng mô hình [1TuanPham/T-VisStar-7B-v0.1], một mô hình ngôn ngữ lớn (Large Language Model – LLM) chuyên biệt cho tiếng Việt, được xây dựng dựa trên kiến trúc nền tảng Mistral – một biến thể tiên tiến và hiệu quả của dòng mô hình Transformer Decoder-only. Mô hình có quy mô khoảng 7 tỷ tham số, được huấn luyện nhằm tối ưu hiệu suất cho các tác vụ sinh ngôn ngữ như:

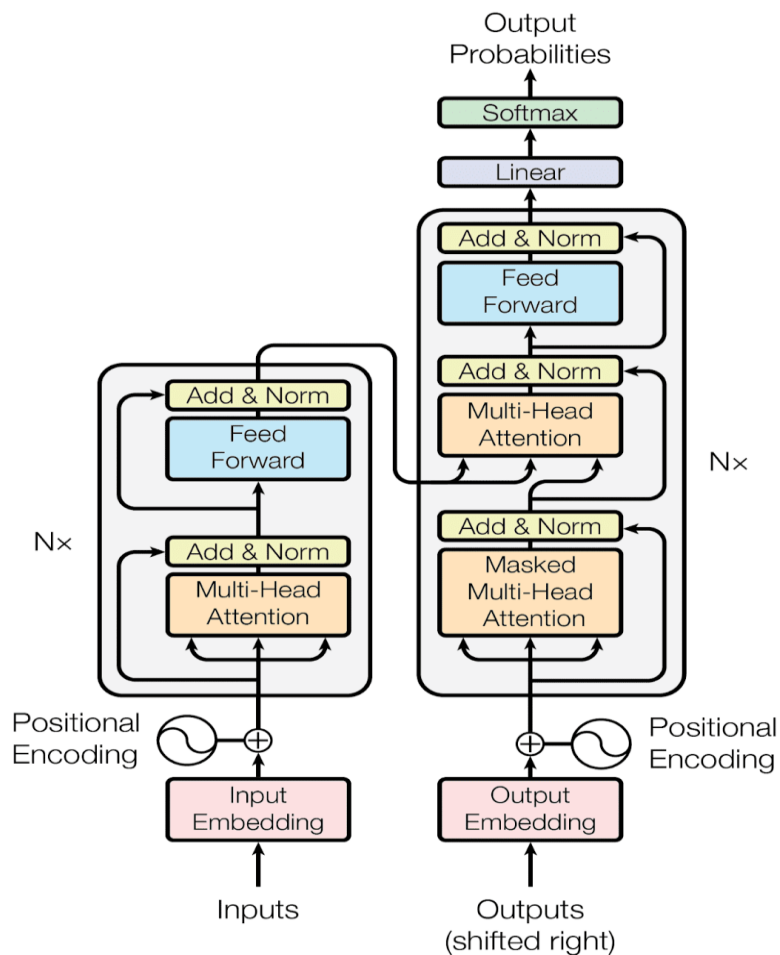
- Trả lời câu hỏi
- Đối thoại
- Tóm tắt văn bản
- Dịch thuật

Trong hệ thống, mô hình được tải về từ thư viện Hugging Face thông qua lớp AutoModelForCausalLM của thư viện transformers. Bên cạnh đó, tokenizer tương ứng được sử dụng để:

- Mã hóa câu hỏi đầu vào thành chuỗi token.
- Sinh đầu ra là văn bản tiếng Việt tự nhiên thông qua cơ chế sinh tuần tự (causal generation).

Nhờ kế thừa kiến trúc mạnh mẽ của Mistral, T-VisStar-7B-v0.1 có khả năng xử lý ngữ cảnh dài, sinh ngôn ngữ mượt mà và phù hợp với nhiều ứng dụng thực tế. Kiến trúc nền tảng của mô hình – Transformer và đặc biệt là biến thể Decoder-only Transformer – sẽ được trình bày chi tiết trong các phần tiếp theo.

#### a. Transformer



Hình 4: Kiến trúc Transformer

- Transformer là một kiến trúc mô hình học sâu dựa trên cơ chế self-attention, cho phép mô hình này hiểu được mối quan hệ giữa các từ trong một câu mà



không cần đến kiến trúc tuần tự truyền thống như RNN (Recurrent Neural Networks) hay LSTM (Long Short-Term Memory). Transformer có khả năng xử lý toàn bộ câu cùng một lúc, điều này giúp tăng tốc độ huấn luyện và cải thiện hiệu quả xử lý.

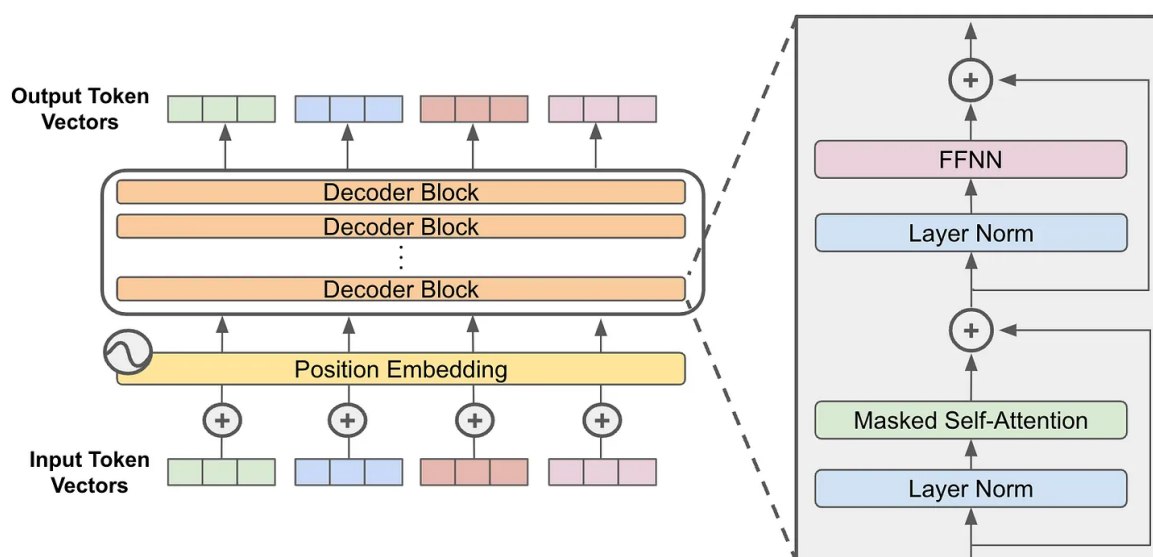
- Trước khi mô hình Transformer được phát triển và trở nên phổ biến trong lĩnh vực xử lý ngôn ngữ tự nhiên (NLP), các kiến trúc chủ yếu được sử dụng là các mạng nơ-ron hồi quy (RNNs), bao gồm cả Long Short-Term Memory (LSTM) và Gated Recurrent Units (GRUs). Mặc dù những công nghệ này đã đạt được nhiều thành công nhất định, chúng vẫn tồn tại một số hạn chế nghiêm trọng đã thúc đẩy sự phát triển của mô hình Transformer.
- Các điểm nổi bật chính của Transformer là:
  - Xử lý song song: Transformer xử lý đầu vào theo từng khối, không tuần tự. Điều này cho phép việc huấn luyện mô hình được thực hiện song song, giảm đáng kể thời gian cần thiết để huấn luyện mô hình.
  - Self-Attention: Cơ chế này giúp Transformer xác định được mối quan hệ giữa tất cả các từ trong một câu, bất kể khoảng cách giữa chúng trong văn bản, giải quyết vấn đề về phụ thuộc dài hạn.
  - Hiệu suất và mở rộng: Với khả năng xử lý đồng thời, Transformer tận dụng tối đa sức mạnh của phần cứng hiện đại, như GPU và TPU, để xử lý các tác vụ NLP một cách hiệu quả.
  - Do những ưu điểm nổi bật này, Transformer đã trở thành một tiêu chuẩn mới trong việc xây dựng các mô hình NLP, dẫn đến sự phát triển của nhiều kiến trúc hiện đại hơn như BERT, GPT-3, và các mô hình tương tự khác.
- Transformer được cấu trúc thành hai phần chính là encoder và decoder.
  - Encoder: Encoder xử lý dữ liệu đầu vào (gọi là "Source") và nén dữ liệu vào vùng nhớ hoặc context mà Decoder có thể sử dụng sau đó.
  - Decoder: Decoder nhận đầu vào từ đầu ra của Encoder (gọi là "Encoded input") kết hợp với một chuỗi đầu vào khác (gọi là "Target") để tạo ra chuỗi đầu ra cuối cùng.

- Mỗi encoder và decoder đều bao gồm nhiều lớp, mỗi lớp chứa các thành phần self-attention và feed-forward neural networks.

#### b. Decoder-only Transformer

**T-VisStar-7B** được xây dựng dựa trên kiến trúc Mistral, là một biến thể tối ưu và cải tiến của kiến trúc Transformer nguyên gốc. Một số điểm nổi bật trong kiến trúc này bao gồm:

- Kiến trúc Base: Mistral (Decoder-only Transformer)
  - Giống như GPT, mô hình chỉ bao gồm phần decoder, phù hợp với các tác vụ sinh chuỗi (causal language modeling).



Hình 5: Kiến trúc Decoder-only Transformer

- Attention mechanism: Multi-Head Causal Self-Attention
  - Cho phép mô hình học được các mối quan hệ phụ thuộc trong chuỗi đầu vào.
  - Dựa trên causal masking để đảm bảo rằng các token sau không được nhìn thấy khi dự đoán token hiện tại.
- Sliding Window Attention (SWA)
  - Khả năng xử lý các chuỗi dài hơn với hiệu suất tốt hơn, bằng cách áp dụng attention theo cửa sổ trượt thay vì attention toàn cục.

- Grouped Query Attention (GQA)
  - Tối ưu hiệu suất tính toán và giảm độ phức tạp của attention, phù hợp với mô hình kích thước lớn như 7B.

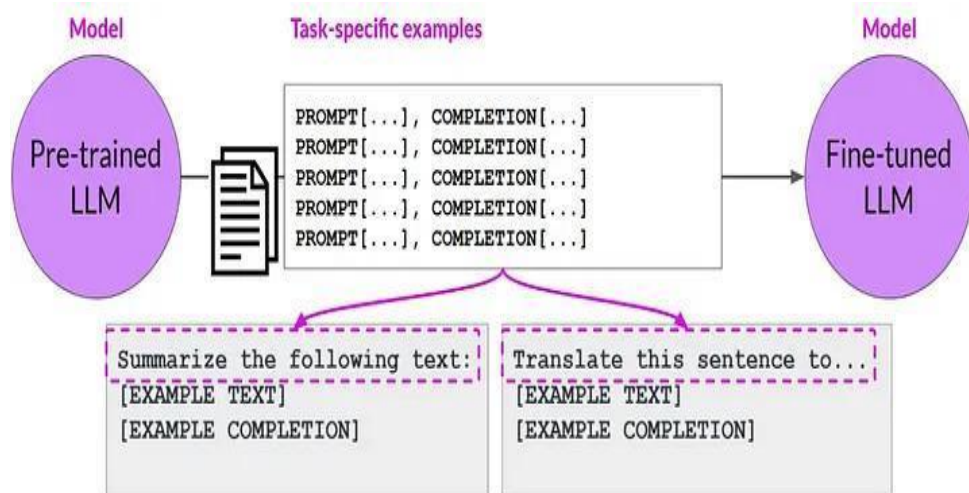
### 1.3.2. Tinh chỉnh LLM (Fine-tuning)

Fine-tuning mô hình là việc adapt mô hình có nhiều tác vụ về mô hình có một số tác vụ nhất định một cách chính xác và hiệu quả

Việc fine-tuning có nhiều lợi ích. Đầu tiên, nó tận dụng được những kiến thức mà mô hình pre-training đã được học rồi, điều này sẽ tiết kiệm được rất nhiều thời gian cũng như là tài nguyên tính toán so với việc training from scratch. Thứ hai, việc fine-tuning cho phép mô hình hoạt động tốt hơn trên một số task cụ thể.

#### a. Instruction fine-tuning

- Instruction fine-tuning là một kỹ thuật để adapt LLMs thực hiện một số task cụ thể dựa trên những lời dẫn cụ thể (**explicit instruction**). Trong khi các cách fine-tuning thông thường thì chỉ huấn luyện dựa trên một tập dataset cụ thể cho task nào đó thì instruction fine-tuning thực hiện sâu hơn bằng cách sử dụng high-level instruction để hướng dẫn cho model thực hiện một số task cụ thể.



Hình 6: Instruction fine-tuning

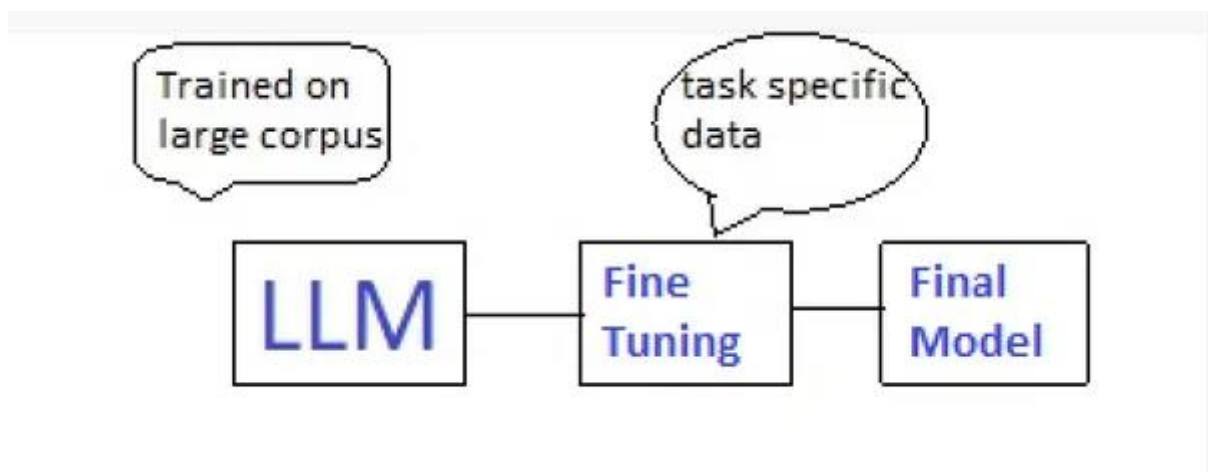
- Việc tạo dữ liệu sẽ làm tăng khả năng và sự linh hoạt của mô hình ngôn ngữ. Với mục đích là hướng dẫn cho model thực hiện một task nào đó

bằng việc sử dụng các prompt hướng dẫn. Hướng tiếp cận này cho phép ta có thể chỉ định outputs mong muốn, khuyến khích model thực hiện behaviors nào đó, hoặc có thể quản lý đầu ra của mô hình tốt hơn.

- Sự khác biệt giữa fine-tuning thông thường và instruction fine-tuning : fine-tuning thông thường dựa trên một lượng đáng kể dữ liệu có nhãn cho một task cụ thể nào đó, trong khi instruction fine-tuning dựa trên lời dẫn cụ thể(explicit instruction) để hướng dẫn mô hình adapt hơn với một lượng dữ liệu có hạn.

#### b. Parameter Efficient Finetuning (PEFT)

Các phương pháp PEFT đã nổi lên như một cách tiếp cận hiệu quả để tinh chỉnh các LLM được đào tạo trước trong khi giảm đáng kể số lượng các tham số có thể đào tạo. Các kỹ thuật này cân bằng hiệu quả tính toán và hiệu suất tác vụ, giúp tinh chỉnh ngay cả các LLM lớn nhất mà không ảnh hưởng đến chất lượng.



Hình 7: Minh họa tinh chỉnh LLM với dữ liệu đặc thù

PEFT mang lại một số lợi ích thực tế, chẳng hạn như giảm sử dụng bộ nhớ, chi phí lưu trữ và độ trễ suy luận. Nó cho phép nhiều tác vụ chia sẻ cùng một mô hình được đào tạo trước, giảm thiểu nhu cầu duy trì các phiên bản độc lập. Tuy nhiên, PEFT có thể đưa ra thời gian đào tạo bổ sung so với các phương pháp tinh chỉnh truyền thống và hiệu suất của nó có thể nhạy cảm với các lựa chọn siêu tham số. **LoRA (Low-Rank Adaptation)** là một phương pháp tiêu biểu trong PEFT, cho phép thêm các ma trận hạng thấp vào mô hình để đạt hiệu quả cao mà không cần thay đổi toàn bộ trọng số

LoRA là một kỹ thuật cải tiến được thiết kế để tinh chỉnh hiệu quả các mô hình ngôn ngữ được đào tạo trước bằng cách đưa các ma trận bậc thấp có thể đào tạo vào từng lớp của kiến trúc Transformer. LoRA nhằm mục đích giảm số lượng tham số có thể đào tạo và gánh nặng tính toán trong khi vẫn duy trì hoặc cải thiện hiệu suất của mô hình đối với các tác vụ hạ lưu.

#### *Cách hoạt động của LoRA:*

- Bảo toàn điểm bắt đầu : Trong LoRA, giả thuyết điểm bắt đầu là rất quan trọng. Giả thuyết này giả định rằng trọng số của mô hình được đào tạo trước đã gần với giải pháp tối ưu cho các tác vụ hạ lưu. Do đó, LoRA đóng băng trọng số của mô hình được đào tạo trước và thay vào đó tập trung vào việc tối ưu hóa các ma trận hạng thấp có thể đào tạo được.
- Ma trận hạng thấp : LoRA đưa các ma trận hạng thấp, được biểu diễn dưới dạng ma trận A và B, vào mô-đun tự chú ý của mỗi lớp. Các ma trận hạng thấp này hoạt động như bộ điều hợp, cho phép mô hình thích ứng và chuyên biệt hóa cho các nhiệm vụ cụ thể trong khi giảm thiểu số lượng tham số bổ sung cần thiết.
- Rank-Deficiency : Một hiểu biết thiết yếu đằng sau LoRA là rank-deficiency của các thay đổi trọng số ( $\Delta W$ ) được quan sát thấy trong quá trình thích ứng. Điều này cho thấy rằng quá trình thích ứng của mô hình liên quan đến những thay đổi có thể được biểu diễn hiệu quả với rank thấp hơn nhiều so với ma trận trọng số ban đầu. LoRA tận dụng quan sát này để đạt được hiệu quả tham số.

#### *Ưu điểm của LoRA:*

- Giảm chi phí tham số : Sử dụng ma trận cấp thấp thay vì tinh chỉnh tất cả các tham số, LoRA giảm đáng kể số lượng tham số có thể đào tạo, giúp tiết kiệm bộ nhớ hơn và rẻ hơn về mặt tính toán.
- Chuyển đổi tác vụ hiệu quả : LoRA cho phép mô hình được đào tạo trước được chia sẻ trên nhiều tác vụ, giảm nhu cầu duy trì các phiên bản tinh chỉnh riêng

biệt cho từng tác vụ. Điều này tạo điều kiện chuyển đổi tác vụ nhanh chóng và liền mạch trong quá trình triển khai, giảm chi phí lưu trữ và chuyển đổi.

- Không có độ trễ suy luận : Thiết kế tuyến tính của LoRA đảm bảo không có độ trễ suy luận bổ sung so với các mô hình được tinh chỉnh hoàn toàn, khiến nó phù hợp với các ứng dụng thời gian thực.

#### 1.4. LangChain Framework

**LangChain** là một framework mã nguồn mở hỗ trợ xây dựng các ứng dụng sử dụng mô hình ngôn ngữ lớn. LangChain giúp đơn giản hóa việc kết nối giữa các thành phần như: LLM, kho dữ liệu văn bản, cơ sở dữ liệu vector, mô hình embedding, và các công cụ tìm kiếm.



Hình 8: Langchain Framework

Các thành phần chính của LangChain sử dụng trong hệ thống:

- DocumentLoader và TextSplitter: Tải và cắt nhỏ tài liệu đầu vào.
- Embeddings: Tạo vector embedding cho văn bản.
- VectorStore: Kết nối với Qdrant để lưu và truy xuất vector.
- RetrievalQA: Thành phần chính của LangChain thực hiện quá trình RAG (xem mục 1.4).

LangChain hỗ trợ tích hợp mô hình tạo sinh và tìm kiếm để trả lời câu hỏi có ngữ cảnh, giúp xây dựng hệ thống hỏi đáp một cách linh hoạt và có thể mở rộng.

## 1.5. Streamlit



*Hình 9: Streamlit*

**Streamlit** là một thư viện mã nguồn mở của Python, được thiết kế để xây dựng nhanh chóng giao diện người dùng cho các ứng dụng machine learning, khoa học dữ liệu và các công cụ tương tác. Với cú pháp đơn giản và khả năng tích hợp mạnh mẽ với hệ sinh thái Python, Streamlit cho phép các nhà phát triển triển khai ứng dụng web chỉ với vài dòng lệnh mà không cần kiến thức sâu về HTML, CSS hay JavaScript

### 1.5.1. Đặc điểm nổi bật của Streamlit:

**Dễ sử dụng:** Streamlit cho phép xây dựng UI thông qua việc viết Python script thông thường, không yêu cầu tách riêng phần frontend và backend.

**Tương tác thời gian thực:** Thông qua các widget như `st.button`, `st.slider`, `st.text_input`, v.v..., người dùng có thể tương tác với ứng dụng và nhận phản hồi ngay lập tức.

**Tích hợp dễ dàng với mô hình ML/AI:** Streamlit hỗ trợ trực tiếp việc hiển thị dữ liệu, biểu đồ, hình ảnh, văn bản và đầu ra của các mô hình machine learning mà không cần cấu hình phức tạp.

**Tự động làm mới giao diện:** Mỗi thay đổi từ phía người dùng (ví dụ nhập liệu, nhấn nút) sẽ tự động kích hoạt việc chạy lại toàn bộ script, giúp giao diện luôn đồng bộ với trạng thái hiện tại.

### 1.5.2. Cấu trúc cơ bản của một ứng dụng Streamlit:

Một ứng dụng Streamlit thường bao gồm:

- Các phần nhập liệu từ người dùng (widgets): như `st.text_input`, `st.selectbox`, `st.file_uploader`.
- Phần xử lý logic: nơi các dữ liệu đầu vào được xử lý, truy vấn hoặc đưa vào mô hình AI/ML.
- Phần hiển thị kết quả: hiển thị văn bản (`st.write`), hình ảnh (`st.image`), bảng (`st.dataframe`), biểu đồ (`st.pyplot`, `st.plotly_chart`), hoặc các kết quả dự đoán.

c. Ưu điểm của việc sử dụng Streamlit trong hệ thống RAG:

- Triển khai nhanh: Phù hợp cho việc tạo giao diện trình diễn nguyên mẫu (prototype) hoặc công cụ demo mô hình AI.
- Thân thiện với người dùng: Giao diện rõ ràng, dễ sử dụng và trực quan cho cả người phát triển lẫn người sử dụng cuối.
- Khả năng mở rộng: Có thể kết hợp với các thư viện backend khác (Flask, FastAPI) hoặc tích hợp với cơ sở dữ liệu để xử lý các tác vụ phức tạp hơn.

## 1.6. FastAPI

FastApi là 1 web framework dùng để build API có hiệu năng cao, code dễ d, đơn giản nhưng cũng hỗ trợ tốt cho việc làm sản phẩm.



*Hình 10: FastAPI*

Đặc điểm:

- Fast: Hiệu suất cao ngang với NodeJS và Go.
- Fast to code: Code nhanh hơn, tốc độ code các features tăng khoảng 200 đến 300 %.
- Fewer bugs: do đơn giản nên giảm số bugs của developer đến 40%.
- Intuitive: hỗ trợ code dễ hơn với tự động gợi ý, debug cần ít thời gian hơn so với trước.
- Easy: được thiết kế sao cho dễ dùng dễ học.
- Short: Tối thiểu việc lập code. Các tham số truyền vào có nhiều tính năng. Ít bugs.
- Robust: hiệu năng mạnh mẽ, có thể tương tác API qua docs.



### 1.6.1. Cài đặt

Để cài đặt framework này trên Ubuntu, bạn cần phiên bản python  $\geq 3.6$ .

```
pip install fastapi
```

Bạn cũng cần ASGI server khi deploy sản phẩm như Uvicorn hoặc Hypercorn.

```
pip install uvicorn
```

### 1.6.2. Tạo file main và chạy ứng dụng

#### Bước 1: Tạo file ứng dụng

Tạo một file Python, ví dụ main.py, với nội dung đơn giản:

```
from fastapi import FastAPI
app = FastAPI()
@app.get("/")
def read_root():
    return {"message": "Hello World!"}
```

#### Bước 2: Chạy server FastAPI

Chạy server bằng lệnh:

```
uvicorn main:app --reload
```

Giải thích:

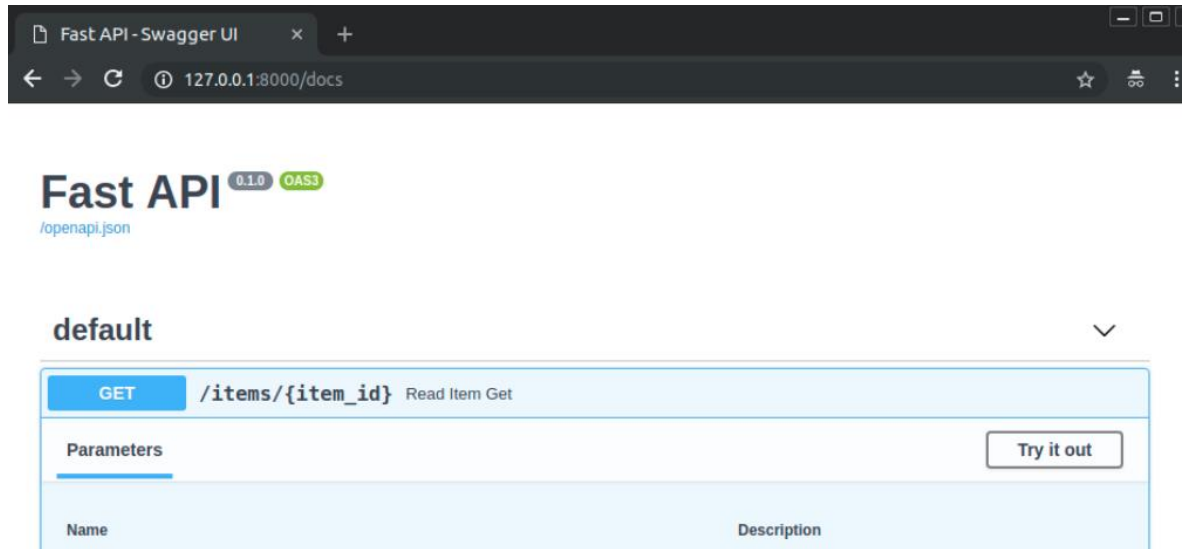
- main: tên file (main.py)
- app: tên biến app khởi tạo
- --reload: tự động reload server khi bạn sửa code (chỉ dùng trong phát triển)

#### Bước 3: Truy cập thử API

- Truy cập API: <http://127.0.0.1:8000/>

### 1.6.3. Tương tác với API docs tự động

Truy cập <http://127.0.0.1:8000/docs> sẽ có Swagger UI như sau:



Hình 11: Tương tác với API docs tự động

### 1.7. Kết chương

Trong chương này, các lý thuyết nền tảng cho việc xây dựng hệ thống hỏi đáp đã được trình bày chi tiết, bao gồm:

- Kiến trúc hệ thống hỏi đáp hiện đại dựa trên mô hình ngôn ngữ lớn
- Kiến trúc RAG giúp nâng cao hiệu quả trả lời truy vấn nhờ truy xuất tri thức
- Lý thuyết và mô hình LLM sử dụng trong hệ thống
- Framework LangChain hỗ trợ tổ chức pipeline xử lý
- Framework Streamlit cho Frontend và FastAPI cho Backend

Những kiến thức lý thuyết này đóng vai trò nền tảng cho việc triển khai hệ thống trong chương tiếp theo.

## CHƯƠNG 2: GIẢI PHÁP ĐỀ XUẤT

### 2.1 Tập dữ liệu

Giải pháp dữ liệu của tôi được thiết kế dựa trên hai loại dữ liệu chính: dữ liệu truy xuất và dữ liệu fine-tune. Dữ liệu truy xuất được dùng để cung cấp thông tin nhanh chóng, chính xác thông qua các kỹ thuật truy vấn và nhúng ngữ nghĩa. Trong khi đó, dữ liệu fine-tune được sử dụng để huấn luyện tinh chỉnh mô hình ngôn ngữ, giúp mô hình hiểu sâu hơn ngữ cảnh chuyên ngành và cải thiện chất lượng phản hồi.

#### 2.1.1 Dữ liệu truy xuất

Dữ liệu thủ tục hành chính từ cổng thông tin (ví dụ: [dichvucong.danang.gov.vn](https://dichvucong.danang.gov.vn)), bao gồm tên thủ tục, mã thủ tục, lĩnh vực, điều kiện thực hiện, thành phần hồ sơ, cơ quan xử lý, thời gian giải quyết, trình tự thực hiện, kết quả thực hiện, căn cứ pháp lý, đối tượng thực hiện,... và các đường dẫn văn bản (.doc, .pdf, link trực tiếp).

Dữ liệu đầu vào được lấy từ <https://dichvucong.danang.gov.vn/thu-tuc-hanh-chinh>. Đây là trang uy tín cung cấp đầy đủ các thông tin về tất cả các thủ tục hành chính của các cấp khác nhau. Ở đây tui em lấy tổng cộng 175 thủ tục hành chính phù hợp với quy mô đề án PBL 7 này ( thủ tục hành chính cấp phường )



Hình 12: Nguồn dữ liệu – Dịch vụ công TP Đà Nẵng

Để crawl dữ liệu, chúng em đã sử dụng các thư viện, framework của Python để dễ dàng lấy dữ liệu và được trình bày chi tiết ở bản dưới đây:

Thư viện	Công dụng
requests	Gửi HTTP request, lấy nội dung web (HTML)
BeautifulSoup	Phân tích và trích xuất thông tin từ HTML
docx.Document	Tạo và ghi dữ liệu vào file Word (.docx)
os, time, random	Xử lý hệ thống, chờ ngẫu nhiên để tránh spam
ThreadPoolExecutor	Đa luồng, giúp tăng tốc crawl dữ liệu
Retry, HTTPAdapter	Tự động thử lại khi gặp lỗi mạng (timeout, 500...)

### Tóm tắt cách crawl dữ liệu thủ tục hành chính:

**Thiết lập cấu hình:** Xác định thư mục lưu trữ, số lượng trang cần crawl, số luồng xử lý song song, đường dẫn lưu lỗi.

**Tạo session có retry:** Dùng requests.Session kết hợp Retry để tự động lặp lại nếu gặp lỗi kết nối, quá tải hoặc timeout.

**Crawl danh sách link thủ tục:** Với mỗi trang, gửi yêu cầu HTTP, phân tích HTML bằng BeautifulSoup để lấy các đường dẫn chi tiết của từng thủ tục.

**Lấy nội dung chi tiết từng thủ tục:** Gửi yêu cầu đến từng link, trích xuất tiêu đề, nội dung chi tiết, tài liệu đính kèm. Dữ liệu được làm sạch và định dạng lại theo tiêu chí rõ ràng.

**Xử lý song song:** Dùng ThreadPoolExecutor để tải nhiều link cùng lúc, giúp tăng tốc độ crawl.

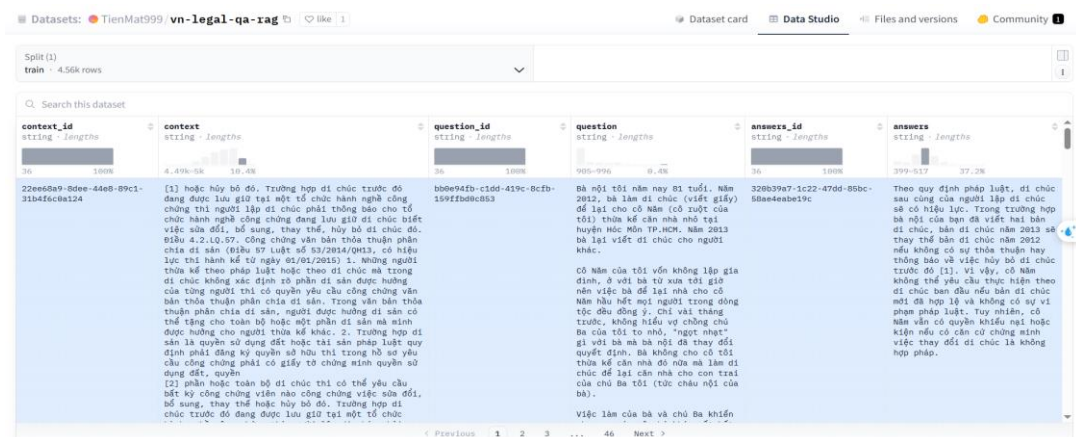
**Ghi dữ liệu ra file Word:** Từng thủ tục được ghi vào file Word, có tiêu đề, nội dung, và link đến tài liệu nếu có.

## 2.1.2 Dữ liệu fine-tune

Để tinh chỉnh mô hình ngôn ngữ lớn phục vụ cho bài toán hỏi đáp pháp luật, tôi sử dụng bộ dữ liệu [vn-legal-qa-rag](#) được công bố trên nền tảng Hugging Face. Đây là tập dữ liệu chuyên biệt cho lĩnh vực pháp luật Việt Nam, được thiết kế nhằm hỗ trợ các mô hình học sâu trong việc hiểu ngữ cảnh pháp lý và tạo ra câu trả lời chính xác, đáng tin cậy.

Tập dữ liệu bao gồm ba trường thông tin chính:

- **context:** Là đoạn văn bản pháp lý (ví dụ: trích dẫn từ bộ luật, nghị định, hoặc văn bản quy phạm pháp luật) đóng vai trò làm ngữ cảnh nền cho câu hỏi. Trường này giúp mô hình hiểu rõ bối cảnh pháp lý để đưa ra câu trả lời phù hợp.
- **question:** Là câu hỏi được đặt ra bởi người dùng hoặc hệ thống, thường liên quan đến các tình huống thực tiễn cần áp dụng pháp luật. Các câu hỏi này được thiết kế đa dạng, bao phủ nhiều chủ đề trong lĩnh vực pháp lý như hình sự, dân sự, hành chính, lao động, v.v.
- **answer:** Là câu trả lời chính xác, ngắn gọn và rõ ràng dựa trên thông tin được cung cấp trong phần context. Câu trả lời phản ánh đúng nội dung pháp luật và phù hợp với yêu cầu của câu hỏi.



context_id	context	question_id	question	answer_id	answer
22ee68a9-8dee-4468-89c1-31b45c8a124	[1] hoặc hủy bỏ đó. Trường hợp di chúc trước đó đang được lưu giữ tại một tổ chức hành nghề công chứng thì người lập di chúc phải thông báo cho tổ chức hành nghề công chứng đang lưu giữ di chúc biết việc sửa đổi, bổ sung, thay thế, hủy bỏ di chúc đó. Điều 4.2.10.57. Công chứng văn bản thỏa thuận phân chia di sản (Điều 57 Luật số 53/2014/QH13, có hiệu lực thi hành kể từ ngày 01/01/2015) 1. Những người thừa kế theo pháp luật hoặc theo di chúc mà trong di chúc không xác định rõ phần di sản được hưởng của từng người thì có quyền yêu cầu công chứng văn bản thỏa thuận phân chia di sản. Trong văn bản thỏa thuận phân chia di sản, người được hưởng di sản có thể tặng cho toàn bộ hoặc một phần di sản mà mình được hưởng cho người thừa kế khác. 2. Trường hợp di sản là quyền sử dụng đất hoặc tài sản pháp luật quy định phải đăng ký quyền sở hữu thì trong hồ sơ yêu cầu công chứng phải có giấy tờ chứng minh quyền sử dụng đất, quyền [2] phần hoặc toàn bộ di chúc thì có thể yêu cầu bất kỳ công chứng viên nào công chứng việc sửa đổi, bổ sung, thay thế hoặc hủy bỏ đó. Trường hợp di chúc trước đó đang được lưu giữ tại một tổ chức	b0b99afb-c1dd-439c-8cfb-199f2b0ec8b3	Bà nội tôi năm nay 81 tuổi. Năm 2012, bà làm di chúc (viết giấy) để lại cho cô Năm (có ruột của tôi) thừa kế căn nhà nhỏ tại huyện Đức Môn 19 HCM. Năm 2013 bà lại viết di chúc cho người khác. Có Năm của tôi vốn không lập gia đình, ở với bà từ xưa tới giờ nên việc bà để lại nhà cho cô Năm hầu hết mọi người trong dòng tộc đều đồng ý. Chỉ vài tháng trước, không hiểu vợ chồng chú Ba của tôi to nhỏ, "ngọt nhạt" gì với bà mà bà nội đã thay đổi quyết định. Bà không cho cô tôi thừa kế căn nhà đó nữa mà làm di chúc để lại căn nhà cho con trai của chú Ba tôi (tức cháu nội của bà).	328b39a7-1c22-47d4-85bc-5b8a4eab219c	Theo quy định pháp luật, di chúc sau cùng của người lập di chúc sẽ có hiệu lực. Trong trường hợp bà nội của bạn đã viết hai bản di chúc, bản di chúc năm 2013 sẽ thay thế bản di chúc năm 2012 nếu không có sự thỏa thuận hay thông báo về việc hủy bỏ di chúc trước đó [1]. Vì vậy, cô Năm không thể yêu cầu thực hiện theo di chúc bên đầu nếu bản di chúc mới đã hợp lệ và không có sự vi phạm pháp luật. Tuy nhiên, cô Năm vẫn có quyền khiếu nại hoặc kiện nếu cô cảm cứ chứng minh việc thay đổi di chúc là không hợp pháp.

Hình 13: Dữ liệu fine-tune (Hugging Face)

Việc fine-tune mô hình trên tập dữ liệu này nhằm mục tiêu giúp mô hình nắm bắt được cách đặt vấn đề và cách trả lời trong ngữ cảnh tiếng Việt. Đồng thời, việc huấn luyện trên cặp câu hỏi - ngữ cảnh - câu trả lời giúp mô hình học được cách trích xuất thông tin có liên quan, lý giải và diễn đạt câu trả lời phù hợp với hành vi ngôn ngữ chuyên ngành.

## 2.2 Mô hình

Trong giải pháp này, tôi sử dụng mô hình **1TuanPham/T-VisStar-7B-v0.1** làm nền tảng để tinh chỉnh cho tác vụ hỏi đáp về lĩnh vực hành chính. Đây là một mô hình mã nguồn mở được xây dựng dựa trên kiến trúc **Mistral-7B**, một kiến trúc tiên tiến được thiết kế bởi Mistral AI, nổi bật với khả năng xử lý văn bản nhanh, hiệu quả, và dung lượng nhẹ so với hiệu suất mang lại.

T-VisStar-7B-v0.1 đã được tiền huấn luyện và tinh chỉnh bước đầu với dữ liệu tiếng Việt chất lượng cao, giúp mô hình hiểu tốt hơn ngôn ngữ và ngữ cảnh văn hóa Việt Nam. Do đó, đây là lựa chọn phù hợp để tiếp tục tinh chỉnh (fine-tune) cho các tác vụ đặc thù.

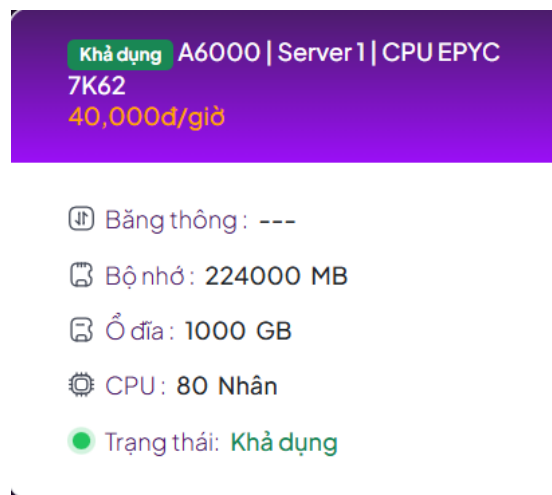
### 2.2.1 Cấu hình tham số huấn luyện LoRA

Tham số	Giá trị	Mô tả
peft_type	LORA	Phương pháp tinh chỉnh tham số hiệu quả (Parameter-Efficient Fine-Tuning)
task_type	CAUSAL_LM	Áp dụng cho mô hình ngôn ngữ sinh văn bản
r	8	Rank của ma trận low-rank; mức độ nén thông tin
lora_alpha	16	Hệ số khuếch đại đầu ra LoRA
lora_dropout	0.05	Tỷ lệ dropout để tránh overfitting
bias	none	Không tinh chỉnh bias trong các lớp

init_lora_weights	true	Khởi tạo trọng số LoRA khi bắt đầu huấn luyện
inference_mode	true	Kích hoạt chế độ suy luận sau huấn luyện
target_modules	["q_proj", "k_proj", "v_proj", "o_proj"]	Các lớp attention được áp dụng LoRA

Thông qua cấu hình này, LoRA chỉ điều chỉnh một lượng nhỏ tham số nhưng vẫn đảm bảo mô hình học được các đặc trưng mới từ tập dữ liệu pháp luật trong quá trình fine-tune, đồng thời giữ lại được kiến thức ngôn ngữ tổng quát từ mô hình nền.

### 2.2.2 Cấu hình phần cứng



Hình 14: Cấu hình phần cứng huấn luyện

Thành phần	Thông số
GPU	NVIDIA A6000
CPU	AMD EPYC 7K62 – 80 nhân xử lý
RAM (Bộ nhớ)	224,000 MB (~224 GB)
Ổ đĩa	1000 GB SSD

VRAM	48 GB
Chi phí thuê	40,000 VNĐ/giờ

### 2.2.3 Kết quả huấn luyện

#### a. *train/loss*

- Mô tả: Loss giảm dần từ khoảng ~0.95 xuống dưới 0.5 trong suốt quá trình training.
- Đánh giá: Đây là tín hiệu tốt cho thấy mô hình đang học được từ dữ liệu. Tuy vẫn còn dao động nhỏ ở cuối, nhưng xu hướng chung là giảm.



Hình 15: Kết quả huấn luyện *train/loss*

#### b. *train/learning\_rate*

- Mô tả: Learning rate cố định ở mức 0.0002 trong toàn bộ quá trình train.

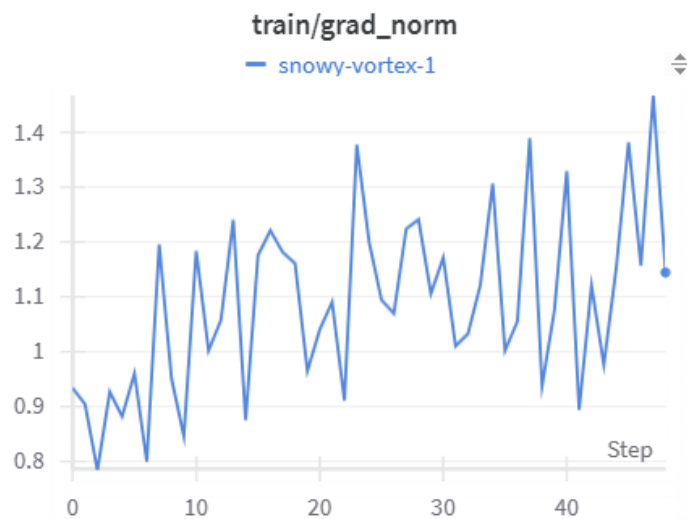




Hình 16: Kết quả huấn luyện  $\text{train/learning\_rate}$

*c. train/grad\_norm*

- Mô tả: Grad norm dao động khá mạnh từ  $\sim 0.8$  đến  $> 1.4$ .
- Đánh giá: Sự dao động là bình thường, nhưng nếu grad norm tiếp tục tăng thì gây ra instability (mất ổn định khi huấn luyện).



Hình 17: Kết quả huấn luyện  $\text{train/grad\_norm}$

## CHƯƠNG 3: PHÂN TÍCH THIẾT KẾ HỆ THỐNG

### 3.1 Phát biểu bài toán

#### 3.1.1 Mục tiêu

Xây dựng hệ thống chatbot hỏi đáp về thủ tục hành chính, cho phép người dùng đặt câu hỏi tự nhiên và nhận câu trả lời chính xác, dễ hiểu, có chèn link dẫn đến văn bản pháp luật liên quan.

#### 3.1.2 Bài toán cần giải quyết

- Hiểu câu hỏi người dùng (Question Understanding)
- Tìm và trích xuất thủ tục liên quan từ kho dữ liệu (Information Retrieval)
- Sinh câu trả lời từ thông tin tìm được (Answer Generation)
- Tự động chèn link văn bản phù hợp vào phần trả lời (Link Insertion)

Mô hình đề xuất: Sử dụng mô hình VisStar (LLM tiếng Việt) và thực hiện fine-tuning trên tập dữ liệu đặc thù về thủ tục hành chính. Mô hình được huấn luyện để sinh câu trả lời ngắn gọn, chính xác từ đoạn văn có sẵn và chèn link văn bản vào phần trả lời nếu người dùng yêu cầu.

#### 3.1.3 Tiêu chí đánh giá:

- Mức độ chính xác của nội dung trả lời.
- Mức độ phù hợp của link văn bản.
- Tỷ lệ câu hỏi được trả lời đầy đủ.
- Thời gian phản hồi.

### 3.2 Phân tích hiện trạng

- Nhu cầu tra cứu thủ tục hành chính cao nhưng trải nghiệm hiện tại còn hạn chế. Người dân phải tự tra cứu qua danh mục hoặc tìm kiếm từ khóa trên các cổng dịch vụ công. Giao diện không thân thiện, nội dung phân mảnh, gây khó khăn cho người không rành thủ tục hoặc công nghệ.
- Dữ liệu đã có nhưng thiếu chuẩn hóa và chưa được khai thác hiệu quả. Các thủ tục hành chính đã được công khai nhưng chủ yếu ở dạng HTML, không có cấu trúc dữ liệu rõ ràng. Nhiều thông tin (thành phần hồ sơ, điều kiện, thời gian...)

nằm rải rác trong văn bản, khó trích xuất tự động. Các file văn bản pháp luật thường ở dạng đính kèm, không dễ truy cập hay liên kết.

- Chưa có hệ thống hỏi đáp tự nhiên và thông minh. Người dùng không thể đặt câu hỏi bằng ngôn ngữ tự nhiên. Các hệ thống hiện tại chưa hỗ trợ hiểu ngữ cảnh, cũng không có khả năng sinh câu trả lời phù hợp từ dữ liệu có sẵn.
- Tài liệu pháp luật đính kèm chưa được tận dụng tốt. Mặc dù đã có văn bản đính kèm trong nhiều thủ tục, nhưng chúng chưa được tự động trích dẫn trong phần trả lời, làm giảm tính minh bạch và khả năng đối chiếu của người dùng.

### **3.2.1. Đối tượng sử dụng**

Hệ thống AI hỗ trợ hỏi đáp thủ tục hành chính được xây dựng nhằm phục vụ các nhóm đối tượng sau:

- Công dân Việt Nam: Người dân có nhu cầu tìm hiểu, thực hiện hoặc giải quyết các thủ tục hành chính ở cấp xã, phường. Hệ thống giúp họ tra cứu thông tin, yêu cầu giấy tờ, quy trình thực hiện một cách nhanh chóng, chính xác và thuận tiện, mà không cần hiểu rõ thuật ngữ pháp lý chuyên sâu.
- Doanh nghiệp và tổ chức: Các doanh nghiệp khi thực hiện các thủ tục pháp lý như đăng ký kinh doanh, xin cấp giấy phép, báo cáo thuế,... cũng có thể sử dụng hệ thống để tra cứu thủ tục, yêu cầu pháp lý và thời gian xử lý.
- Cán bộ hành chính – công chức: Hệ thống có thể hỗ trợ công chức trong việc chuẩn bị thông tin, hướng dẫn công dân, giảm tải khối lượng công việc trả lời các câu hỏi lặp lại hoặc xử lý những thắc mắc đơn giản.
- Nhà nghiên cứu và sinh viên: Những người làm trong lĩnh vực hành chính công, luật học, công nghệ thông tin cũng có thể sử dụng hệ thống như một công cụ hỗ trợ tra cứu và tìm hiểu dữ liệu thủ tục hành chính phục vụ nghiên cứu.

### **3.2.2 Chức năng**

Hệ thống được thiết kế với các chức năng chính như sau:

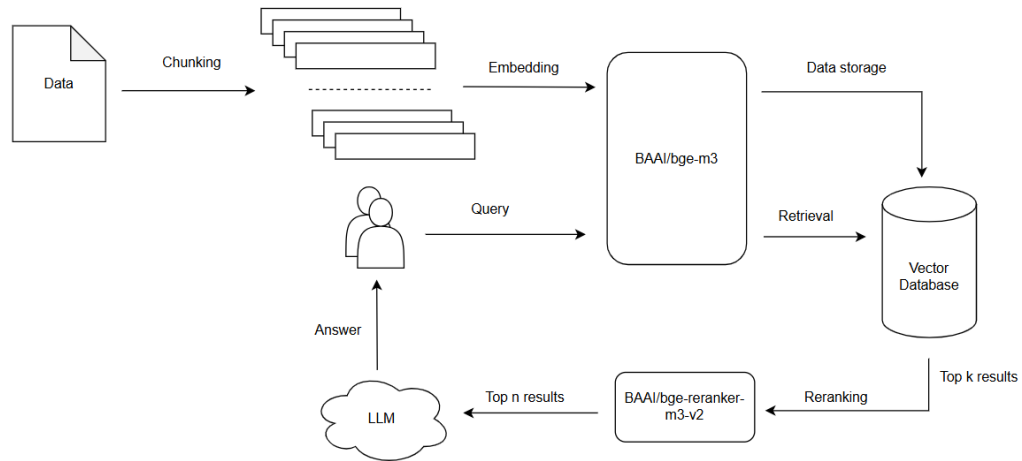
- a. Hỏi đáp tự động về thủ tục hành chính:
- Người dùng có thể đặt câu hỏi bằng ngôn ngữ tự nhiên (tiếng Việt) liên quan đến bất kỳ thủ tục hành chính nào.
  - Hệ thống sử dụng mô hình ngôn ngữ lớn (LLM) kết hợp với RAG (Retrieval-Augmented Generation) để tìm kiếm và tổng hợp thông tin từ các tài liệu chính thống, đưa ra câu trả lời rõ ràng, chính xác.
- b. Tìm kiếm thủ tục hành chính theo từ khóa hoặc danh mục:
- Cung cấp chức năng tìm kiếm theo tên thủ tục, lĩnh vực (ví dụ: hộ tịch, xây dựng, kinh doanh, y tế...), hoặc cơ quan thực hiện.
  - Hiển thị các thông tin như: thành phần hồ sơ, thời gian giải quyết, cơ quan tiếp nhận, lệ phí, kết quả.
- c. Trích xuất và hiển thị văn bản liên quan:
- Hệ thống cho phép người dùng truy cập nhanh đến các văn bản quy phạm pháp luật có liên quan đến thủ tục được hỏi.
  - Có thể hiển thị nội dung ngắn gọn hoặc toàn văn.
- d. Hỗ trợ đa nền tảng:
- Giao diện có thể tích hợp trên web hoặc ứng dụng di động để tăng tính tiện lợi cho người sử dụng.
- e. Cập nhật dữ liệu mới nhanh chóng
- Dữ liệu mới được cập nhật với thao tác nhanh chóng và đơn giản.

### 3.2.3 Công nghệ sử dụng

- RAG: LangChain Framework
- LLM: 1TuanPham/T-VisStar-7B-v0.1(Hugging Face)
- Vector Database: Qdrant, ElasticSearch
- Frontend: Streamlit
- Backend: FastAPI, Ngrok

### 3.3 Thiết kế hệ thống

#### 3.3.2 Sơ đồ tổng thể



Hình 18: Kiến trúc hệ thống

#### 3.3.3 Mô tả chi tiết

1. Data (Dữ liệu gốc) được đưa vào hệ thống.
2. Dữ liệu này được chia nhỏ (Chunking) thành nhiều đoạn văn bản nhỏ để dễ xử lý và tăng hiệu quả tìm kiếm.
3. Các đoạn văn bản sau khi chunking được đưa vào mô hình BAAI/bge-m3 để tạo embedding (vector biểu diễn ngữ nghĩa).
4. Các embedding này sẽ được lưu trữ vào cơ sở dữ liệu vector (Vector Database) để phục vụ cho truy xuất sau này.
5. Người dùng gửi một truy vấn (Query) vào hệ thống.
6. Truy vấn này cũng được xử lý bởi mô hình BAAI/bge-m3 để tạo ra vector embedding tương ứng với truy vấn.
7. Embedding của truy vấn được sử dụng để truy vấn trong Vector Database nhằm tìm ra top-k kết quả phù hợp nhất (dựa trên độ tương đồng về mặt ngữ nghĩa).
8. Các kết quả top-k được truy xuất từ vector database sẽ được gửi đến mô hình BAAI/bge-reranker-m3-v2 để reranking — tức là đánh giá lại và sắp xếp các kết quả dựa trên mức độ phù hợp ngữ nghĩa sâu hơn.

9. Top-n kết quả sau khi reranking được gửi đến LLM (mô hình ngôn ngữ lớn).
10. LLM sử dụng thông tin này để sinh ra câu trả lời cuối cùng gửi đến người dùng

### 3.4 Xây dựng chương trình

#### 3.4.2 Tổ chức thư mục

Hệ thống được chia thành hai phần chính, đặt tại hai môi trường khác nhau:

- Frontend: Chạy trên máy local
- Backend: Chạy trên notebook kaggle

#### 3.4.3 Frontend

- Phần frontend được xây dựng bằng thư viện Streamlit, giúp người dùng có thể nhập câu hỏi, gửi truy vấn đến backend và hiển thị kết quả trả lời một cách trực quan.

- Cấu trúc thư mục:

frontend/

├── main.py

├── .venv

└── requirements.txt

- main.py: File chính chạy giao diện Streamlit
- .venv: môi trường ảo tránh xung đột thư viện
- requirement.txt: Các thư viện cần cài đặt để chạy frontend

#### 3.4.4 Backend

- Backend được triển khai trên nền tảng Kaggle Notebook, chịu trách nhiệm xử lý toàn bộ pipeline: từ tiền xử lý, nhúng dữ liệu, lưu trữ vector, truy xuất thông tin liên quan, reranking và cuối cùng là tạo câu trả lời từ mô hình ngôn ngữ lớn

(LLM). Đồng thời, nếu cần tinh chỉnh mô hình, phần fine-tuning cũng được thực hiện tại đây.

- Cấu trúc tổ chức trên Kaggle:

```
backend/  
  
├── data/  
  
├── code/  
  
|   ├── create_database  
  
|   ├── rag  
  
|   ├── backend  
  
|   └── fine-tune
```

### 3.5 Kết chương

Trong chương này, hệ thống hỏi đáp dựa trên mô hình ngôn ngữ lớn đã được phân tích và thiết kế một cách có hệ thống, đầy đủ và chi tiết. Trước tiên, nhóm đã làm rõ yêu cầu bài toán, xác định các chức năng chính và phân tích luồng xử lý nghiệp vụ từ phía người dùng đến mô hình AI. Sau đó, kiến trúc tổng thể của hệ thống được đề xuất theo hướng phân tách rõ ràng giữa frontend và backend, giúp dễ dàng triển khai và mở rộng sau này.

Ngoài ra, chương này cũng đã trình bày chi tiết sơ đồ thiết kế hệ thống, tổ chức thư mục, vai trò của từng module, qua đó đảm bảo hệ thống vận hành mạch lạc và dễ bảo trì. Việc tách biệt giữa các module cũng giúp tăng khả năng mở rộng và tái sử dụng mã nguồn, thuận lợi cho các nghiên cứu hoặc cải tiến sau này.

Những nội dung được trình bày trong chương này đóng vai trò nền tảng cho quá trình hiện thực hoá hệ thống ở các chương tiếp theo, đặc biệt là trong khâu triển khai, huấn luyện mô hình và thử nghiệm hiệu quả hệ thống hỏi đáp.

## CHƯƠNG 4: TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ

### 3.1. Triển khai chương trình

#### 3.1.1. Môi trường triển khai

Hệ thống hỏi đáp được triển khai trên hai môi trường tách biệt, phù hợp với đặc thù của từng thành phần trong kiến trúc hệ thống:

- Frontend: được triển khai cục bộ (local) trên máy người dùng, sử dụng nền tảng Streamlit nhằm đơn giản hóa giao diện và dễ dàng thao tác.
- Backend: được triển khai và thực thi trên nền tảng Kaggle Notebook, tận dụng tài nguyên tính toán mạnh miễn phí (GPU/TPU) để xử lý các tác vụ nặng như sinh embedding, reranking và gọi mô hình LLM.

Việc phân tách môi trường như trên giúp giảm tải tài nguyên trên máy người dùng và tối ưu hóa hiệu suất xử lý phía backend.

#### 3.1.2. Cấu hình hệ thống

##### a. Máy local

- Hệ điều hành: Windows 10
- Python: 3.10+
- RAM: 4GB trở lên
- Thư viện yêu cầu: Streamlit, Requests

##### b. Kaggle NoteBook

- CPU: 2 vCPU (Intel Xeon)
- RAM: 13 GB
- GPU: NVIDIA Tesla T4 x2
- Disk: 20 GB (tạm thời)
- Python kernel: Python 3.10
- Runtime: Notebook execution (session-based)



## 3.2. Kết quả thực nghiệm

### 3.2.1. Đăng nhập hệ thống



**ĐĂNG NHẬP**

Số định danh cá nhân

Mật khẩu

Đăng nhập

*Hình 19: Kết quả - Giao diện đăng nhập*

### 3.2.2. Chức năng cập nhật dữ liệu (Đang hoàn thiện)

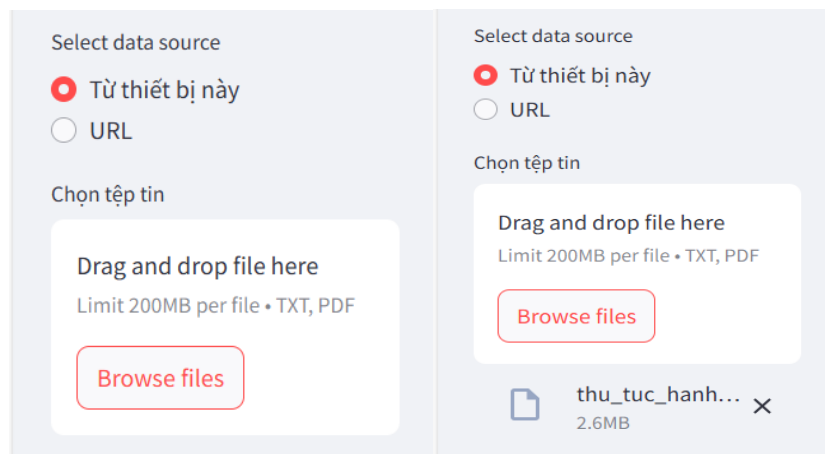
#### a) Mô tả

- Truy cập hệ thống và trong phần sidebar chọn ‘tải dữ liệu lên’
- Nhấn vào nút **“Chọn tệp”** hoặc **“Upload file”**

Hệ thống hỗ trợ các định dạng:

- Văn bản (.txt)
- PDF (.pdf)
- Word (.docx)
- Bảng dữ liệu (.csv)
- Nhấn ‘Cập nhật’ sau vài giây đến vài phút (tùy kích thước tài liệu), hệ thống sẽ hiển thị:
  - “Dữ liệu đã được cập nhật thành công!”
  - Nếu lỗi: “Không thể xử lý file. Vui lòng kiểm tra định dạng.”

#### b) Kết quả



Hình 20: Kết quả - Giao diện cập nhật dữ liệu

### c) Nhận xét

Chức năng Cập nhật dữ liệu trong hệ thống RAG được thiết kế trực quan, dễ sử dụng và phù hợp với người dùng không chuyên kỹ thuật. Giao diện tải lên đơn giản, hỗ trợ nhiều định dạng tài liệu phổ biến, giúp người dùng nhanh chóng bổ sung nguồn dữ liệu đầu vào cho hệ thống.

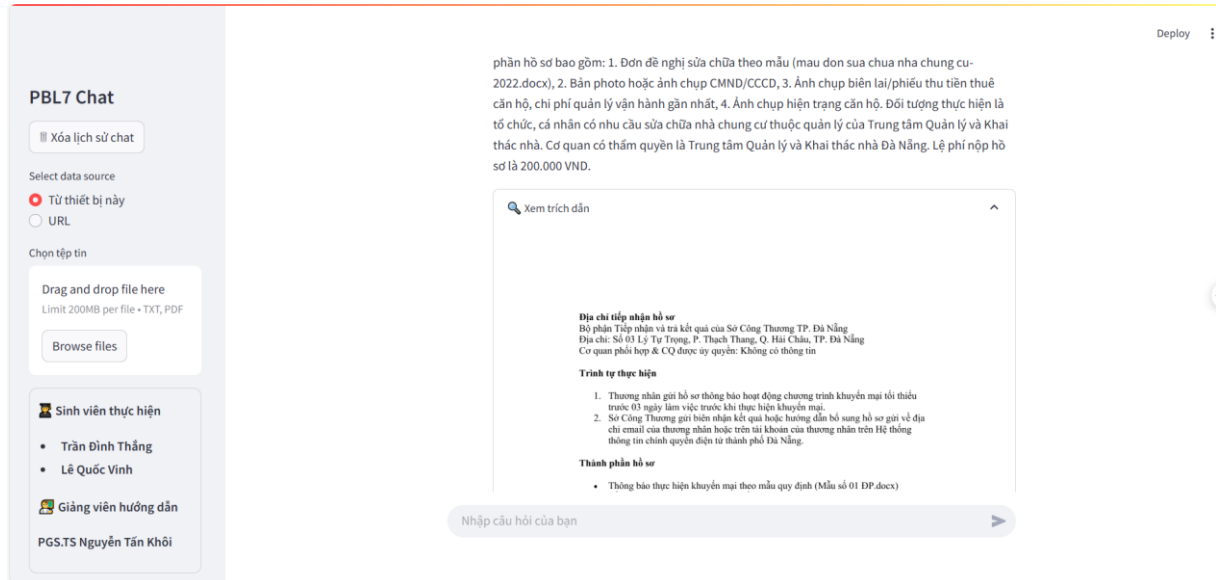
### 3.2.3. Chức năng hỏi đáp, hướng dẫn



Hình 21: Kết quả - Giao diện hỏi đáp hướng dẫn

### 3.2.4. Chức năng cung cấp đường link , trích dẫn tài liệu liên quan

Mô tả: Sau khi hệ thống trả lời, sẽ trích xuất phần tài liệu liên quan đến câu trả lời đó để tăng tính xác thực và chính xác cho câu trả lời đó, và cung cấp các thông tin khác nếu người dùng có thể cần



Hình 22: Kết quả - Giao diện xem trích dẫn



Hình 23: Kết quả - Giao diện cung cấp đường link

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## 1. KẾT QUẢ ĐẠT ĐƯỢC

Trong quá trình nghiên cứu lý thuyết và triển khai ứng dụng công nghệ, đề án đã đạt được những kết quả quan trọng như sau:

Về mặt lý thuyết

- Tìm hiểu và tổng hợp cơ sở lý thuyết về kỹ thuật Retrieval-Augmented Generation (RAG) và phương pháp fine-tuning mô hình ngôn ngữ lớn (LLM) trong ngữ cảnh xử lý ngôn ngữ tự nhiên và hỗ trợ trả lời câu hỏi.
- Phân tích đặc thù và quy trình thủ tục hành chính cấp phường xã tại TP Đà Nẵng, làm cơ sở xây dựng hệ thống hướng dẫn chính xác và phù hợp.

Về mặt thực tiễn ứng dụng

- Xây dựng thành công hệ thống hỗ trợ hướng dẫn thủ tục hành chính cấp phường xã TP Đà Nẵng dựa trên công nghệ RAG kết hợp fine-tuning LLM, giúp người dân tiếp cận thông tin nhanh, chính xác và dễ dàng hơn.
- Triển khai mô hình fine-tune LLM trên bộ dữ liệu đặc thù về thủ tục hành chính địa phương, tăng cường khả năng hiểu và trả lời các câu hỏi chuyên sâu liên quan đến các thủ tục.
- Ứng dụng RAG giúp hệ thống có thể truy vấn nhanh cơ sở dữ liệu tài liệu thủ tục, từ đó tạo câu trả lời chính xác và phù hợp với yêu cầu người dùng.

Các đóng góp chính của đề án

1. Phát triển và triển khai thành công thuật toán RAG kết hợp fine-tuning LLM để hỗ trợ trả lời các câu hỏi về thủ tục hành chính cấp phường xã, cải thiện đáng kể độ chính xác và thời gian phản hồi so với các phương pháp truyền thống.
2. Xây dựng bộ dữ liệu đặc thù và quy trình fine-tune mô hình ngôn ngữ lớn, chứng minh khả năng thích nghi và vận dụng mô hình vào lĩnh vực hành chính công.

3. Xác định và chuẩn hóa các quy trình thủ tục hành chính cấp phường xã TP Đà Nẵng, tạo cơ sở dữ liệu tham khảo tin cậy cho hệ thống.

Những hạn chế và vấn đề còn tồn tại

- Mô hình vẫn còn gặp khó khăn khi xử lý các câu hỏi phức tạp, đa nghĩa hoặc chứa nhiều yếu tố ngữ cảnh đặc thù chưa được đào tạo đầy đủ.
- Hệ thống phụ thuộc vào chất lượng và độ đầy đủ của dữ liệu đầu vào, trong khi thủ tục hành chính thường xuyên thay đổi, dẫn đến yêu cầu cập nhật dữ liệu liên tục để đảm bảo tính chính xác.
- Tốc độ truy vấn có thể chậm khi khối lượng tài liệu và dữ liệu lớn, cần tối ưu hóa hơn nữa để đáp ứng yêu cầu thực tế.

## **2. KIẾN NGHỊ VÀ HƯỚNG PHÁT TRIỂN**

### **1. Kiến nghị**

- Chuẩn hóa và cập nhật dữ liệu hành chính: Cơ quan quản lý nhà nước nên xây dựng một cơ sở dữ liệu chuẩn hóa, có cấu trúc, được cập nhật định kỳ và công khai, nhằm tạo điều kiện cho các hệ thống tự động như đồ án này có thể tích hợp và khai thác hiệu quả.
- Hỗ trợ truy cập API dữ liệu công: Đề xuất các cơ quan chức năng cung cấp API truy xuất thông tin thủ tục hành chính ở các cấp (phường, xã, quận, thành phố), góp phần thúc đẩy các giải pháp công nghệ tiếp cận dữ liệu nhanh chóng và chính xác hơn.
- Xây dựng cơ chế phản hồi và xác minh thông tin: Đề nghị xây dựng chức năng góp ý/phản hồi từ người dân, giúp hệ thống học hỏi và cải thiện theo thời gian, đồng thời phát hiện kịp thời thông tin không chính xác hoặc lỗi thời.

### **2. Hướng phát triển**

- Tích hợp giọng nói và giao diện đa phương tiện: Phát triển thêm khả năng giao tiếp bằng giọng nói (voice assistant), phù hợp với người lớn tuổi, người không quen dùng văn bản hoặc người khuyết tật.
- Mở rộng phạm vi ứng dụng: Mở rộng hệ thống hỗ trợ cho toàn bộ TP Đà Nẵng và các tỉnh thành khác, đồng thời hỗ trợ nhiều lĩnh vực hành chính công khác nhau như: đất đai, y tế, giáo dục, hộ tịch...
- Tối ưu hóa truy vấn và xử lý ngôn ngữ: Cải tiến thuật toán truy vấn tài liệu (retriever) nhằm nâng cao tốc độ và độ chính xác khi khối lượng dữ liệu tăng lên. Đồng thời, ứng dụng kỹ thuật continual learning để mô hình có thể học thêm mà không cần huấn luyện lại toàn bộ.
- Kết nối với các nền tảng chính quyền số: Đề xuất tích hợp hệ thống vào các cổng dịch vụ công trực tuyến hiện có của thành phố để người dân có thể vừa tra cứu vừa nộp hồ sơ trực tuyến thuận tiện.

## TÀI LIỆU THAM KHẢO

- [1] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Riedel, S. (2020). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. arXiv preprint arXiv:2005.11401. <https://arxiv.org/abs/2005.11401>
- [2] Gururangan, S., Marasović, A., Swayamdipta, S., Lo, K., Beltagy, I., Downey, D., & Smith, N. A. (2020). *Don't Stop Pretraining: Adapt Language Models to Domains and Tasks*. arXiv preprint arXiv:2004.10964. <https://arxiv.org/abs/2004.10964>
- [3] Pinecone. (2023). *Semantic Chunking Strategies for NLP*. Pinecone Blog. <https://www.pinecone.io/learn/chunking-strategies/>
- [4] Mistral AI. (2024). *Mistral Model Documentation*. <https://docs.mistral.ai/>
- [5] Công Dịch vụ công TP Đà Nẵng. (2024). *Danh mục và hướng dẫn thủ tục hành chính cấp phường/xã*. Truy cập tại: <https://dichvucong.danang.gov.vn>
- [6] VinAI Research. (2023). *PhoBERT, viT5, viMistral: Pretrained Language Models for Vietnamese*. HuggingFace. <https://huggingface.co/VinAI>
- [7] Bộ Tư pháp Việt Nam. (2023). *Cơ sở dữ liệu quốc gia về thủ tục hành chính*. Truy cập tại: <https://csdl.dichvucong.gov.vn>
- [8] LangChain AI. (2024). *LangChain: Framework for Building Applications with LLMs through Composition*. GitHub Repository. <https://github.com/langchain-ai/langchain>
- [9] Pinecone. (2023). *Semantic Search with Vector Databases*. <https://www.pinecone.io/learn>
- [10] Thaker, M. (2024). *Build your own RAG with Mistral-7B and Langchain*. Medium. <https://medium.com/@thakermadhav/build-your-own-rag-with-mistral-7b-and-langchain-97d0c92fa146>
- [11] HuggingFace. (2024). *Mistral — Transformers Documentation (v4.48.0)*. [https://huggingface.co/docs/transformers/v4.48.0/model\\_doc/mistral](https://huggingface.co/docs/transformers/v4.48.0/model_doc/mistral)

- [12] Networks. (2024). *A Step-by-Step Guide to Fine-Tuning the Mistral 7B LLM*.  
<https://www.e2enetworks.com/blog/a-step-by-step-guide-to-fine-tuning-the-mistral-7b-llm>



## **PHỤ LỤC**