



Kế hoạch chi tiết 14 tuần xây dựng Giai đoạn 1 (Text-Based IR từ audio)

Tuần 1: Nghiên cứu & Lập kế hoạch

- **Mục tiêu:** Hiểu yêu cầu chung của hệ thống, thiết kế kiến trúc pipeline, chuẩn bị môi trường phát triển (cài đặt Python, thư viện ASR, vector DB, LLM frameworks...).
- **Các task:**
 - Tổng hợp yêu cầu: Xác định các thành phần chính (ASR, tách đoạn, embedding, vector DB, RAG, đánh giá).
 - Nghiên cứu RAG và pipeline retrieval: Đọc tài liệu về Retrieval-Augmented Generation (RAG) để nắm workflow tổng quát (chunk → embedding → truy vấn → LLM) [1](#) [2](#).
 - Lập kế hoạch sơ bộ: Phân chia công việc cho từng tuần. Chuẩn bị repo, tài khoản API (nếu dùng OpenAI), tài liệu tham khảo.
- **Thời gian:** Khoảng 1-2 ngày cho nghiên cứu tài liệu và lập sơ đồ kiến trúc, 2-3 ngày cài đặt môi trường (ASR, vector DB, LLM frameworks như LangChain/HuggingFace), 1 ngày tổng hợp và phê duyệt kế hoạch chi tiết.

Tài liệu tham khảo:

- “Retrieval Augmented Generation (RAG) and Semantic Search for GPTs” (OpenAI Help) [1](#) [2](#) : Giải thích khái niệm RAG (nạp ngữ cảnh bên ngoài vào prompt) và trình tự pipeline (chunking – embedding – lưu trữ – truy vấn – sinh câu trả lời). Tài liệu này giúp định hình luồng dữ liệu tổng quát cho hệ thống và nhấn mạnh các bước cốt lõi (như LangChain’s RetrievalQA) [2](#).
- “What Is Retrieval-Augmented Generation (RAG)?” (NVIDIA Blog) [3](#) [4](#) : Giới thiệu RAG là kỹ thuật tăng cường mô hình sinh ngôn ngữ với dữ liệu bên ngoài, cho phép mô hình trích dẫn nguồn như “chú thích chân trang” [4](#). Bài viết này minh họa lợi ích của RAG (độ chính xác, tin cậy cao hơn, dễ triển khai) [3](#), góp ý trong việc lựa chọn kết hợp LLM với kho tri thức bên ngoài.

Tuần 2: Xây dựng pipeline ASR (audio → transcript)

- **Mục tiêu:** Triển khai thành phần chuyển đổi audio thành văn bản có kèm mốc thời gian (timestamp).
- **Các task:**
 - Lựa chọn mô hình ASR: Thử nghiệm các mô hình speech-to-text mã nguồn mở (ví dụ OpenAI Whisper, Wav2Vec2, HuBERT). Tham khảo báo cáo hiệu năng (độ chính xác, tốc độ) và tính hỗ trợ timestamp của chúng. Whisper cho độ chính xác cao với 680k giờ huấn luyện [5](#).
 - Cài đặt và thử nghiệm: Sử dụng Hugging Face Transformers pipeline (`pipeline(task="automatic-speech-recognition", return_timestamps=True)`) với mô hình Whisper để thu được transcript và timestamp cho từng đoạn nói [6](#) [7](#). Kiểm tra kết quả với một vài file audio mẫu (lưu ý định dạng, chất lượng âm).

- Xử lý đầu ra: Ghép nối các mẩu transcript liên tiếp, lấy timestamp đầu và cuối của mỗi đoạn (như hướng dẫn trong ví dụ pipeline [8](#)). Xử lý lỗi thông dụng (điều chỉnh ngữбер xác suất, loại bỏ tiếng ồn).
- **Thời gian:** 1-2 ngày nghiên cứu mô hình và giải pháp timestamp, 2-3 ngày cài đặt thử nghiệm (HuggingFace pipeline, PyTorch/TensorFlow), 1-2 ngày kiểm thử và tối ưu (điều chỉnh tham số).

Tài liệu tham khảo:

- “Whisper: Robust Speech Recognition via Large-Scale Weak Supervision” (OpenAI) [5](#) : Mô tả tập dữ liệu huấn luyện Whisper (680k giờ speech) và kết quả benchmark. Whisper được chọn vì độ bền (robustness) và khả năng nhận dạng đa ngôn ngữ. Nội dung chính nêu rõ khả năng khái quát cao của Whisper, hữu ích cho khâu ASR mà không cần tinh chỉnh thêm [5](#).
- *HuggingFace Transformers - Pipeline documentation* [6](#) [7](#) : Hướng dẫn sử dụng pipeline ASR với tham số `return_timestamps=True`. Đoạn tài liệu này giải thích cách pipeline Whisper trả về danh sách “chunks” chứa text và timestamp (segment-level) [7](#). Ta sẽ áp dụng hướng dẫn này để thu thập transcript và thông tin thời gian tương ứng (bước chuẩn bị cho RAG).

Tuần 3: Làm sạch transcript và chuẩn bị chunk semantically

- **Mục tiêu:** Tiền xử lý văn bản từ ASR và thực hiện chia nhỏ (chunking) văn bản thành các đoạn có ý nghĩa (semantic chunks).
- **Các task:**
 - **Làm sạch văn bản:** Loại bỏ ký tự lạ, chỉnh sửa lỗi đánh máy nếu có, chuẩn hóa viết hoa, dấu câu. Đảm bảo transcript có định dạng dễ chia đoạn.
 - **Nghiên cứu phương pháp phân đoạn:** Tìm hiểu các chiến lược chia chuỗi văn bản thành đoạn (chunks) phục vụ truy xuất thông tin. So sánh chia theo độ dài cố định, chia theo ngữ cảnh (câu, đoạn), và **semantic chunking** (dựa trên ngữ nghĩa). Semantic chunking đảm bảo mỗi chunk vừa có ý nghĩa hoàn chỉnh, cải thiện độ chính xác khi tìm kiếm [9](#).
- **Phân tích ví dụ:** Thử nghiệm chia đoạn theo câu hoặc đoạn văn thông thường (ví dụ cài đặt `RecursiveCharacterTextSplitter` của LangChain) và đánh giá kết quả ban đầu.
- **Thời gian:** 2 ngày làm sạch và định dạng văn bản (chạy thử, chỉnh sửa script), 2 ngày nghiên cứu và thử nghiệm các phương pháp chia đoạn (tài liệu và code mẫu), 1 ngày so sánh kết quả và lên quyết định chiến lược chunking ban đầu.

Tài liệu tham khảo:

- “Semantic Chunking for RAG” (blog by Plaban Nayak) [9](#) : Giải thích khái niệm Semantic Chunking – chia văn bản theo ý nghĩa để đảm bảo tính toàn vẹn ngữ cảnh của mỗi đoạn. Bài viết nhấn mạnh semantic chunking “đảm bảo sự toàn vẹn thông tin, dẫn đến kết quả truy xuất chính xác hơn” [9](#). Chọn tài liệu này vì nó phân tích chi tiết cách nhóm các câu có ngữ nghĩa tương đồng, phù hợp cho bước xử lý transcript trước khi lưu vào cơ sở vectơ.
- *LangChain - SemanticChunker Guide* [10](#) [11](#) : Hướng dẫn cách sử dụng `SemanticChunker` trong LangChain: chia văn bản thành câu rồi gộp những câu tương tự (dựa trên embedding). Nội dung chính mô tả thuật toán: tính embedding mỗi câu, nếu khoảng cách embedding giữa hai câu vượt ngưỡng (breakpoint threshold), tách đoạn [11](#). Tài liệu này hữu ích để triển khai thực tế semantic chunking tự động theo ngữ nghĩa, giúp hiện thực hóa nhiệm vụ chia đoạn thông minh.

Tuần 4: Triển khai Semantic Chunking và đánh giá ban đầu

- **Mục tiêu:** Hoàn thiện phương pháp chia đoạn theo ngữ nghĩa, tạo lập các chunk cuối cùng để nhúng embedding.
- **Các task:**
 - Cài đặt Semantic Chunker: Sử dụng `SemanticChunker(OpenAIEmbeddings())` (LangChain) hoặc tự triển khai dựa trên hướng dẫn ¹⁰ ¹¹. Tinh chỉnh tham số (kích thước chunk tối thiểu, threshold) sao cho mỗi chunk khoảng 100-300 từ (tùy mô hình LLM nhập), vẫn giữ ngữ cảnh tốt.
 - Kiểm thử chunking: Áp dụng lên các transcript mẫu, so sánh chiến lược chunking thông thường (độ dài cố định) và semantic chunking. Đánh giá trực quan kết quả, xem có đoạn thông tin bị cắt nhầm hay không.
- **Tối ưu lại:** Nếu semantic chunking quá chậm hoặc quá cắt nhỏ, điều chỉnh cài đặt hoặc kết hợp phương pháp (ví dụ: tách theo câu cơ bản trước, sau đó merge/nối semantic).
- **Thời gian:** 1-2 ngày cài semantic chunker và tích hợp vào pipeline, 1-2 ngày kiểm thử kết quả trên transcripts dài, 1 ngày tối ưu (điều chỉnh tham số hoặc tăng tốc).

Tài liệu tham khảo:

- *LangChain – Semantic Splitter Tutorial* ¹⁰ ¹¹: Trình bày chi tiết cách dùng SemanticChunker để tách dựa trên embedding. Các đoạn mã mẫu và giải thích thuật toán tại đây giúp ta triển khai đúng semantic chunking (tách khi khoảng cách embedding giữa hai câu vượt ngưỡng) ¹¹. Chọn tài liệu này vì nó cung cấp ví dụ thực tế với LangChain, giúp việc cài đặt linh hoạt.
- *DataHub – “Building and Evaluating RAG Pipelines” (đoạn Transcript)* ¹²: Phần transcript trên DataHub mô tả tầm quan trọng của chiến lược chia đoạn và nêu rõ SemanticChunker trong LangChain. Tuy đây là tài liệu hướng dẫn học tập, nó cung cấp khái quát về cách semantic chunking hoạt động và được đánh giá so với chiến lược chia truyền thống ¹². Tham khảo này giúp củng cố hiểu biết về lý do chọn semantic chunking.

Tuần 5: Tạo embedding và lưu trữ vào Vector Database

- **Mục tiêu:** Chọn mô hình nhúng văn bản (text embedding), biến các chunk thành vector và lưu vào cơ sở dữ liệu vector để truy xuất.
- **Các task:**
 - **Lựa chọn embedding model:** Thủ nghiệm các embedding model (OpenAI `text-embedding-ada-002`, HuggingFace SentenceTransformers, Cohere API...). Đánh giá tốc độ và chất lượng (khả năng phân biệt ngữ nghĩa). Ví dụ, Ada-002 của OpenAI thường được dùng sản xuất vì chất lượng cao ¹³.
 - **Tạo embedding:** Sử dụng thư viện (LangChain, SentenceTransformers, HuggingFace) để chuyển từng chunk thành vector. Cần thận giới hạn kích thước input để không vượt token của model.
 - **Thiết lập Vector DB:** Lựa chọn Vector Database (FAISS, Chroma, Milvus, Pinecone, Weaviate...). Ví dụ, FAISS là thư viện phổ biến cho tìm kiếm tương tự nhanh ¹⁴. Xây dựng index và thêm các vector của chunks vào cơ sở dữ liệu. Kiểm thử truy vấn mẫu để đảm bảo quá trình thêm vector thành công.
- **Thời gian:** 2 ngày chọn và thử embedding model, 1 ngày viết script tạo embedding cho toàn bộ tập dữ liệu, 1-2 ngày cài đặt và cấu hình vector DB (ví dụ cài FAISS hay chạy dịch vụ Pinecone), 1 ngày kiểm thử lưu/truy vấn ban đầu.

Tài liệu tham khảo:

- "Understanding Vector Databases and Embeddings" (Medium) ¹³ ¹⁵: Giới thiệu cơ bản về embedding và vector DB. Nêu rõ embeddings chuyển văn bản thành vectơ số để biểu diễn ý nghĩa và rằng hệ thống chứa vector DB để tìm kiếm tương tự ¹³. Tác giả cũng giải thích cách lưu vector nhanh trong DB (ví dụ FAISS) ¹⁵. Tài liệu này giúp hiểu nguyên lý hoạt động của cơ sở vectơ và lý do chọn thuật toán embedding.
- "Building a RAG Pipeline with LangChain and OpenAI" (Medium) ¹⁶ ¹⁷: Hướng dẫn cụ thể từng bước mă nguồn cho pipeline RAG. Đặc biệt, phần code minh họa sử dụng `OpenAIEmbeddings` để nhúng văn bản và FAISS để lưu trữ ¹⁶. Tài liệu này chọn vì trình bày thực tế việc tạo vectorstore từ các tài liệu đã tách nhỏ, sát với yêu cầu tuần này.

Tuần 6: Thiết lập Vector Database và thử nghiệm truy vấn tương tự

- **Mục tiêu:** Hoàn thiện hệ thống lưu trữ vector, đảm bảo có thể truy vấn tương tự (similarity search) hiệu quả.
- **Các task:**
 - Cài đặt chi tiết DB: Nếu dùng FAISS (thư viện) thì kiểm tra tạo index (`faiss.IndexFlatL2`) hoặc IVF và add vectors. Nếu dùng dịch vụ (Pinecone/Weaviate/Chroma), set up API key và cấu hình bucket/namespace. Xác nhận số lượng vector, kích thước dimension, định dạng dữ liệu.
 - Kiểm thử truy vấn: Viết script lấy query mẫu, chuyển thành embedding và gọi phương thức tìm kiếm. Ví dụ, với FAISS có thể dùng `index.search(query_vector, k)` để lấy top-k kết quả ¹⁸. Đoạn tutorial của HuggingFace cũng chỉ ra cách thêm index FAISS trong Dataset và tìm nearest neighbors ¹⁸ ¹⁹.
- Tối ưu hiệu năng: Đo lường tốc độ truy vấn cho bộ dữ liệu thử (vd. vài nghìn vector). Nếu chậm, cân nhắc cấu hình lại index (IVF, HNSW) hoặc chuyển DB khác. Xem xét khả năng scaling (nếu dữ liệu lớn).
- **Thời gian:** 1 ngày hoàn tất cấu hình chi tiết (index/cluster), 2 ngày kiểm thử truy vấn với nhiều truy vấn và điều chỉnh, 1 ngày đánh giá hiệu năng và lên kế hoạch tối ưu nếu cần.

Tài liệu tham khảo:

- "Understanding Vector Databases and Embeddings" (Medium) ¹⁵: Mô tả khái quát về tính năng cơ sở dữ liệu vector (lưu và tìm kiếm nhanh các embedding) ¹⁵. Ví dụ bài viết giới thiệu FAISS là thư viện phổ biến để tìm kiếm tương tự ²⁰. Tài liệu này hỗ trợ về mặt lý thuyết khi chọn FAISS hoặc các giải pháp khác.
- *Hugging Face LLM Course - Semantic Search with FAISS* ¹⁸ ¹⁹: Hướng dẫn sử dụng FAISS trong thư viện Datasets. Phần này giải thích nguyên tắc FAISS (tạo cấu trúc index cho phép tìm gần nhất) ¹⁸ và ví dụ cụ thể cách thêm index FAISS rồi truy vấn nearest neighbors ¹⁹. Tài liệu giúp tham khảo cách tích hợp FAISS trong pipeline Python và kiểm thử tìm kiếm cơ bản.

Tuần 7: Xây dựng pipeline truy vấn (Retrieval)

- **Mục tiêu:** Hoàn thiện thành phần truy xuất thông tin: nhận câu hỏi đầu vào, chuyển thành vector truy vấn, tìm các chunk liên quan từ vector DB.
- **Các task:**
 - Tạo truy vấn đầu vào (query): Xây dựng chức năng chuyển truy vấn dạng text thành embedding (cùng model như chunks) và gửi tới vector DB.

- Trích chọn kết quả: Lấy top-K chunk tương tự nhất theo cosine/Euclid. Chỉ định số lượng kết quả (ví dụ k=3). Kiểm tra tính đa dạng và liên quan của kết quả.
- Xây dựng interface thử nghiệm: Viết script đơn giản cho phép nhập câu hỏi, hiển thị các chunk được truy xuất kèm similarity score (giúp đánh giá trực quan).
- **Thời gian:** 1 ngày viết chức năng tạo vector từ truy vấn và gọi truy vấn, 1 ngày lấy k=3-5 kết quả và đánh giá sơ bộ (nhìn kết quả có hợp lý không), 1 ngày viết giao diện thử nghiệm (console hoặc Jupyter notebook) và kiểm thử với nhiều câu hỏi khác nhau.

Tài liệu tham khảo:

- AWS Blog – “What is RAG?”²¹: Mô tả luồng RAG, trong đó truy vấn được chuyển thành vector và so khớp với cơ sở vectơ để lấy tài liệu liên quan²¹. Đoạn này minh họa ví dụ cụ thể về truy xuất (nhân viên hỏi về phép năm, kết quả lấy chính sách tương ứng). Tham khảo này giúp hiểu rõ bước convert query thành vectơ và khớp với DB.
- “Understanding Vector Databases and Embeddings” (Medium)²²: Trình bày ba cách truy vấn vector DB, bao gồm gọi `similarity_search(query)` để tự động chuyển văn bản thành vectơ và tìm kiếm²². Tác giả cũng giới thiệu interface truy vấn thống nhất (`db.as_retriever()`) của LangChain) cho phép gọi trực tiếp query. Tài liệu này giúp chọn cách triển khai nhanh truy vấn tương tự cho pipeline.

Tuần 8: Tích hợp RAG và mô hình sinh (LLM) – Tạo câu trả lời

- **Mục tiêu:** Xây dựng hệ thống trả lời câu hỏi: sử dụng LLM (GPT) để sinh đáp án dựa trên các chunk được truy xuất.
- **Các task:**
 - Chọn LLM và phương thức sinh: Lựa chọn mô hình GPT (có thể dùng API ChatGPT/GPT-3.5 hoặc model mã nguồn mở như LLaMA, Falcon). Thiết lập môi trường (OpenAI API key hoặc tải model qua HuggingFace).
 - Tạo prompt cho LLM: Xác định cách đưa thông tin chunk vào prompt (ví dụ: dùng RetrievalQA của LangChain). Có thể nối nhiều đoạn retrieved vào prompt, kèm câu hỏi.
 - Thử nghiệm sinh câu trả lời: Đặt câu hỏi mẫu, chạy LLM kết hợp context retrieved và nhận kết quả. Kiểm tra sự liên quan, tính chính xác tổng thể của câu trả lời.
- **Thời gian:** 2 ngày cài đặt và cấu hình LLM (API, model), 1 ngày xây dựng pipeline nối dữ liệu (ví dụ `RetrievalQA.from_chain_type` của LangChain) và 2 ngày thử nghiệm với một số câu hỏi thực tế, tinh chỉnh prompt và tham số nhiệt độ.

Tài liệu tham khảo:

- “Building a RAG Pipeline with LangChain and OpenAI”²³¹⁷: Đoạn code mẫu minh họa sử dụng `RetrievalQA.from_chain_type` kết hợp ChatGPT và FAISS retriever. Các dòng giải thích (L155-164) nêu rõ cách chia văn bản, tạo vectơ và cấu thành pipeline truy vấn¹⁷. Tham khảo này trực quan cho thấy cách triển khai pipeline RAG đầy đủ (embedding, retriever, LLM).
- NVIDIA Blog – “RAG và nguồn thông tin”⁴: Nêu bật lợi ích khi mô hình đưa ra câu trả lời kèm nguồn tham khảo (“như chú thích chân trang”)⁴. Tài liệu này giúp định hướng thiết kế phần trả lời có dẫn nguồn (timestamps) và nhấn mạnh giảm ảo tưởng của LLM bằng RAG.

Tuần 9: RAG nâng cao – Sinh câu trả lời có dẫn nguồn (timestamp)

- **Mục tiêu:** Hiện thực hóa yêu cầu sinh câu trả lời kèm nguồn dẫn (mốc thời gian từ transcript).
- **Các task:**
 - Chèn timestamp: Trong quá trình tạo câu trả lời, gắn nhãn nguồn cho từng thông tin. Ví dụ, mỗi chunk retrieved có trường timestamp của đoạn audio, sẽ dùng để trích dẫn nguồn. Thiết kế format trả lời (có thể như "(nguồn: [timestart-timestamp])").
 - Tinh chỉnh prompt: Khuyến khích LLM "trích dẫn" nguồn, có thể bằng template câu hỏi hoặc system prompt. Ví dụ: "Please answer and cite the timestamps of the source segments."
 - Đánh giá ban đầu: Kiểm thử với một số câu hỏi, đối chiếu câu trả lời và kiểm tra liệu các timestamp dẫn nguồn có chính xác với nội dung trả lời không.
- **Thời gian:** 2 ngày triển khai và tinh chỉnh format trích dẫn nguồn trong prompt, 2 ngày thử nghiệm với nhiều trường hợp (đoạn trả lời dài, nhiều nguồn), 1 ngày chỉnh sửa định dạng nếu cần (như đánh số hoặc hiển thị link).

Tài liệu tham khảo:

- NVIDIA Blog – *RAG và nguồn thông tin* ⁴ : Nhấn mạnh khả năng của RAG trong việc cho phép mô hình trả lời kèm nguồn tham khảo, xây dựng niềm tin người dùng ⁴. Tài liệu này hỗ trợ lý do tại sao cần dẫn nguồn (timestamp) và cách triển khai theo mô hình RAG.
- AWS – “*What is RAG?*” ²⁴ : Nêu lợi ích của RAG trong việc “cho phép mô hình trình bày thông tin chính xác với trích dẫn nguồn”. Phần này ủng hộ ý tưởng tích hợp timestamp như một nguồn đáng tin cậy ²⁴, phù hợp cho việc báo cáo kết quả cuối cùng của hệ thống.

Tuần 10: Thiết kế bộ đánh giá và chỉ số hiệu năng

- **Mục tiêu:** Xác định các chỉ số đánh giá chất lượng hệ thống (cả khâu ASR và IR/QA). Tạo kế hoạch cho bộ dữ liệu đánh giá.
- **Các task:**
 - Chỉ số ASR: Xác định WER (Word Error Rate) làm tiêu chí đánh giá độ chính xác chuyển ngữ ²⁵. Lập kế hoạch thu thập văn bản gốc (script chuẩn) để so sánh với transcript do hệ thống tạo ra.
 - Chỉ số Retrieval: Định nghĩa Precision/Recall@k cho truy xuất chunk (dựa trên ground truth nếu có thể). Ví dụ, với tập câu hỏi thử, đánh giá xem chunk liên quan có được lấy ra không.
 - Chỉ số QA: Nếu có đáp án đúng (hoặc điểm chuẩn), tính Accuracy/F1 (với QA dạng trích xuất hoặc tự động chấm). Nếu không, dùng các tiêu chí không tham chiếu (như chất lượng/RAGAS) ²⁶.
- Chuẩn bị bộ câu hỏi kiểm thử: Tạo hoặc thu thập tập các câu hỏi mẫu (và trả lời mẫu nếu có) liên quan đến nội dung audio. Đảm bảo đủ đa dạng (câu hỏi ngắn dài, câu hỏi chi tiết, tổng quát).
- **Thời gian:** 1 ngày nghiên cứu thước đo và công cụ (ví dụ thư viện tính WER), 1 ngày chuẩn bị cơ sở dữ liệu kiểm thử (câu hỏi và ground truth), 1 ngày lập kịch bản tính toán metric (precision/recall, WER), 1 ngày rà soát và hoàn thiện bộ đánh giá.

Tài liệu tham khảo:

- “*RAGAS: Automated Evaluation of RAG*” (arXiv 2023) ²⁶ : Giới thiệu khung đánh giá RAG không cần tham chiếu (Ragas) gồm các chỉ số về truy vấn (liên quan ngữ cảnh) và sinh câu trả lời (faithfulness) ²⁶. Tài liệu

này cho thấy cần đánh giá riêng biệt retrieval và generation – phù hợp với kế hoạch đánh giá hệ thống.

- “10 Metrics to Evaluate Speech Recognition Systems” (blog) ²⁵ : Nêu rõ WER là chuẩn mực đánh giá độ chính xác của ASR (đếm lỗi so với tổng từ) ²⁵. Tài liệu này nhấn mạnh tầm quan trọng của WER trong đánh giá bộ phận ASR, bổ sung cho kế hoạch đo lường chất lượng transcript.

Tuần 11: Chuẩn bị bộ dữ liệu đánh giá (dataset)

- **Mục tiêu:** Thiết lập dữ liệu thử nghiệm cho toàn hệ thống, bao gồm audio (và transcript chuẩn) kèm câu hỏi-đáp mẫu.
- **Các task:**
 - Chọn nguồn dữ liệu: Tìm kiếm các nguồn audio phù hợp (podcast, hội thảo) có transcript chuẩn hoặc tự thực hiện thu âm và tạo transcript. Đảm bảo đa dạng nội dung (chuyên ngành khác nhau nếu có).
 - Xây dựng câu hỏi-đáp: Dựa trên nội dung các audio, tạo bộ câu hỏi thử (và trả lời tham khảo). Nên có ít nhất 20-30 câu hỏi (mix câu hỏi "lấy thông tin cụ thể" và "tổng quát"). Ghi lại đáp án đúng để tính toán đánh giá.
 - Định dạng dữ liệu: Lưu trữ audio, transcript, timestamp tham khảo (nếu có), và danh sách câu hỏi-trả lời trong format dễ dùng (CSV, JSON).
- **Thời gian:** 2 ngày thu thập/ghi âm và tạo transcript chuẩn, 2 ngày soạn câu hỏi-trả lời với lời giải (có thể nhờ người thứ ba hoặc script extract), 1 ngày tổng hợp và kiểm thử định dạng dataset.

Tài liệu tham khảo:

- “VoxRAG: A Step Toward Transcription-Free RAG Systems” (arXiv 2023) – tuy trọng tâm khác nhưng gợi ý về cách thiết kế bộ dữ liệu QA trên audio và nhóm câu hỏi. **Không trực tiếp trích dẫn, nhưng định hướng xây dựng tập kiểm thử phức tạp hơn trong tương lai.**
- **Các ví dụ về dataset QA (WikiQA, SQuAD) – tham khảo cấu trúc câu hỏi-trả lời để chuẩn hóa định dạng, mặc dù đây là bài bản dựa trên văn bản chứ không phải audio.** (Không cần trích dẫn cụ thể**).

Tuần 12: Thực hiện đánh giá và phân tích kết quả

- **Mục tiêu:** Chạy bộ đánh giá trên hệ thống, thu thập các chỉ số đo lường, phân tích kết quả để xác định điểm mạnh/yếu.
- **Các task:**
 - Tính toán WER: So sánh transcript hệ thống tạo ra với transcript chuẩn để tính WER cho từng file audio. Tổng hợp WER trung bình.
 - Đánh giá Retrieval: Với mỗi câu hỏi, xem liệu chunk đúng có được lấy (precision@k, recall@k). Tính trung bình trên tập câu hỏi.
 - Đánh giá QA: So sánh câu trả lời hệ thống với đáp án tham khảo (dùng F1/Exact Match nếu phù hợp). Đồng thời khảo sát chất lượng câu trả lời (độ liên quan, độ trung thực).
 - Báo cáo: Tổng hợp kết quả các metric, phân tích các trường hợp sai (ví dụ: ASR sai nhiều, chunk sai, LLM trả lời sai thông tin). Đề xuất cải tiến tiềm năng (xem tuần sau).
- **Thời gian:** 2 ngày chạy thử và tính metric tự động, 1 ngày đánh giá định tính (kiểm tra và phân tích các câu trả lời/đầu ra sai lệch), 1 ngày tổng hợp báo cáo kết quả và chuẩn bị kế hoạch cải tiến.

Tài liệu tham khảo:

- “RAGAS: Automated Evaluation of RAG”²⁶: Nhắc lại tầm quan trọng đánh giá đa chiều (retrieval vs generation) cho hệ thống RAG. Kết quả đánh giá được chia thành phần truy vấn và phần sinh, phù hợp với cách chúng ta phân tích.
- “RAG Evaluation Metrics Explained” (blog trên Medium) – nguồn tham khảo về các chỉ số như Context Precision/Recall, Faithfulness... Mặc dù bài viết đầy đủ cần đăng nhập, nó cung cấp ý tưởng đánh giá riêng biệt từng bước. (**Không có trích dẫn trực tiếp do giới hạn truy cập**).

Tuần 13: Cải tiến và tối ưu hóa

- **Mục tiêu:** Dựa trên kết quả đánh giá, điều chỉnh các thành phần hệ thống để cải thiện hiệu năng và độ chính xác.
- **Các task:**
 - Tinh chỉnh ASR: Nếu WER cao, thử các biện pháp như tăng độ dài đầu vào batch (nếu model hỗ trợ), đổi model lớn hơn (Whisper-Large), hoặc tiền xử lý (lọc tạp âm, tăng âm lượng). Đánh giá lại.
 - Cải thiện chunking/embedding: Nếu kết quả retrieval kém, thử thay đổi chiến lược chunking (tăng overlap, thay threshold semantic). Thử embedding khác (tăng dimension, đổi model). Tham khảo khuyến nghị “thử nghiệm với nhiều mô hình, chiến lược khác nhau”²⁷.
- Điều chỉnh pipeline RAG: Tinh chỉnh prompt và chain type (ví dụ từ “stuff” sang “map_reduce” trong LangChain) để cải thiện chất lượng đáp án. Kiểm thử độ ổn định, giảm độ ngẫu nhiên.
- **Thời gian:** 3 ngày cải tiến các thành phần (2 ngày sửa ASR/embedding/chunking, 1 ngày thử các phương án RAG khác), 1 ngày kiểm thử toàn bộ sau cải tiến, 1 ngày đánh giá lại các chỉ số để xác nhận tiến bộ.

Tài liệu tham khảo:

- “Understanding Vector Databases and Embeddings”²⁷: Phần kết luận khuyên “experiment with different models” và “try different chunking strategies”²⁷. Tài liệu này tổng kết tốt rằng cần thử nghiệm nhiều biến thể để tối ưu. Chọn vì nó khuyến khích vòng lặp lặp đi lặp lại cải tiến – đúng với mục tiêu tuần này.
- “RAGAS: Automated Evaluation of RAG”²⁶: Giúp so sánh lại hiệu năng sau cải tiến bằng các metric đa chiều (retrieval và generation), đảm bảo đánh giá toàn diện trước và sau khi thay đổi hệ thống.

Tuần 14: Hoàn thiện hệ thống và biên soạn tài liệu

- **Mục tiêu:** Đóng gói kết quả, viết báo cáo/tài liệu hướng dẫn sử dụng, kiểm thử cuối cùng.
- **Các task:**
 - Kiểm thử cuối cùng: Đánh giá lại toàn bộ pipeline trên bộ kiểm thử, đảm bảo không có lỗi nghiêm trọng. Hiệu chỉnh các scripts, đặt tên hàm rõ ràng, thêm bình luận code.
 - Viết tài liệu: Soạn hướng dẫn cài đặt, sử dụng pipeline. Ghi chú kiến trúc tổng quan, mô tả từng bước. Bao gồm cách bổ sung dữ liệu mới, chạy đánh giá.
- Chuẩn bị báo cáo cuối: Tổng hợp kết quả quan trọng (metric, ảnh chụp màn hình mẫu), đánh giá thành công so với mục tiêu.
- **Thời gian:** 2 ngày kiểm thử và sửa lỗi cuối, 2 ngày viết hướng dẫn và báo cáo, 1 ngày tổng hợp (format PDF/HTML) và xem lại.

Tài liệu tham khảo:

- “*Technical Documentation in Software Development*” (blog) ²⁸ ²⁹ : Giải thích khái niệm sản phẩm và quy trình tài liệu. Mục tiêu của tuần là tạo tài liệu kỹ thuật: tài liệu sản phẩm (requirements, kiến trúc, code...) và tài liệu quy trình (hướng dẫn sử dụng, release notes). Trích các đoạn trong tài liệu này nêu rằng “Product documentation bao gồm requirements, tech specs, business logic...” và “System documentation covers requirements, design, code, QA...” ²⁸ ²⁹ . Những chỉ dẫn này nhấn mạnh cấu trúc tài liệu cần chuẩn bị (yêu cầu, kiến trúc, QA...).
- (*Không có tài liệu cụ thể*) – Ngoài ra có thể tham khảo các hướng dẫn làm README, Wiki từ cộng đồng (ví dụ mẫu từ GitHub, Atlassian) để đảm bảo rõ ràng. (Không trích dẫn cụ thể trong văn bản)

Nguồn: Các tài liệu/trích dẫn trên đây được chọn từ các bài báo, blog và tài liệu kỹ thuật của OpenAI, NVIDIA, HuggingFace, LangChain, AWS, arXiv, v.v. để minh họa và hỗ trợ cho từng giai đoạn triển khai. Nội dung tóm tắt nêu lý do chọn tài liệu và cách ứng dụng vào bài toán (ví dụ RAG概念 để thiết kế pipeline ¹ , semantic chunking để chia nhỏ văn bản ⁹ , hoặc đánh giá WER cho ASR ²⁵).

¹ ² Retrieval Augmented Generation (RAG) and Semantic Search for GPTs | OpenAI Help Center

<https://help.openai.com/en/articles/8868588-retrieval-augmented-generation-rag-and-semantic-search-for-gpts>

³ ⁴ What Is Retrieval-Augmented Generation aka RAG | NVIDIA Blogs

<https://blogs.nvidia.com/blog/what-is-retrieval-augmented-generation/>

⁵ [2212.04356] Robust Speech Recognition via Large-Scale Weak Supervision

<https://arxiv.org/html/2212.04356>

⁶ ⁷ Pipelines

https://huggingface.co/docs/transformers/en/main_classes/pipelines

⁸ openai/whisper-large-v2 · Audio transcribing, timestamping for whole sentences.

<https://huggingface.co/openai/whisper-large-v2/discussions/16>

⁹ Semantic Chunking for RAG. What is Chunking ? | by Plaban Nayak | The AI Forum | Medium

<https://medium.com/the-ai-forum/semantic-chunking-for-rag-f4733025d5f5>

¹⁰ ¹¹ How to split text based on semantic similarity | LangChain

https://python.langchain.com/docs/how_to/semantic-chunker/

¹² Building and Evaluating RAG Pipelines - Transcript

<https://datahub.io/@donbr/datahub-pkm-rag-evals/rag-chunking-strategies-transcript>

¹³ ¹⁴ ¹⁵ ²⁰ ²² ²⁷ Understanding Vector Databases and Embeddings: A Complete Beginner’s Guide to Document Search and Retrieval | by Anil Goyal | Aug, 2025 | Medium

<https://medium.com/@anil.goyal0057/understanding-vector-databases-and-embeddings-a-complete-beginners-guide-to-document-search-and-4a042850a020>

¹⁶ ¹⁷ ²³ Building a Retrieval-Augmented Generation (RAG) Pipeline with LangChain and OpenAI | by Sindhu | Medium

<https://medium.com/@sindhujad6/building-a-retrieval-augmented-generation-rag-pipeline-with-langchain-and-openai-d4890a292adb>

¹⁸ ¹⁹ Semantic search with FAISS - Hugging Face LLM Course

<https://huggingface.co/learn/llm-course/en/chapter5/6>

²¹ ²⁴ What is RAG? - Retrieval-Augmented Generation AI Explained - AWS
<https://aws.amazon.com/what-is/retrieval-augmented-generation/>

²⁵ 10 Metrics to Assess Speech Recognition Systems Performance
<https://waywithwords.net/resource/speech-recognition-systems-performance/>

²⁶ [2309.15217] Ragas: Automated Evaluation of Retrieval Augmented Generation
<https://arxiv.org/abs/2309.15217>

²⁸ ²⁹ Technical Documentation in Software Development: Types and T
<https://www.altexsoft.com/blog/technical-documentation-in-software-development-types-best-practices-and-tools/>