# **Xây dựng chatbot RAG song ngữ Anh–Lào cho ngân hàng**

## **1. Chiến lược embedding dữ liệu tối ưu (nhúng vector dữ liệu)**

Để hỗ trợ tìm kiếm ngữ nghĩa chính xác, ta nên **nhúng trường "prompt"** – tức là cụm từ/ngữ ngắn gọn thể hiện ý định câu hỏi. Việc nhúng các **đoạn văn bản ngắn, trọng tâm** giúp vector đại diện tập trung vào ý nghĩa cụ thể của câu hỏi, thay vì bị nhiễu bởi nội dung quá dài​

[pinecone.io](https://www.pinecone.io/learn/chunking-strategies/#:~:text=When%20a%20sentence%20is%20embedded%2C,in%20a%20paragraph%20or%20document)

. Ngược lại, nếu nhúng toàn bộ câu trả lời dài, vector có thể mang nhiều thông tin không liên quan, làm giảm độ chính xác khi so khớp với truy vấn ngắn của khách hàng​

[pinecone.io](https://www.pinecone.io/learn/chunking-strategies/#:~:text=that%20captures%20the%20broader%20meaning,querying%20the%20index%20more%20difficult)

. Do đó:

* Mỗi **FAQ prompt** (ví dụ: *“full name of LVB”*) nên được nhúng thành vector và lưu vào kho vector (vd. ChromaDB). Khi người dùng hỏi (bằng tiếng Anh hoặc Lào), hệ thống sẽ nhúng câu hỏi đó và tìm vector **gần nhất** trong kho, nhờ đó xác định đúng ý định và câu trả lời tương ứng.
* Có thể **kết hợp nhúng câu hỏi gốc và từ khóa**: Để tăng độ phủ ngữ nghĩa, một số hệ thống tạo 2 vector cho mỗi mục FAQ – một cho câu hỏi đầy đủ, một cho cụm từ khóa chính​  
  [community.openai.com](https://community.openai.com/t/how-to-best-prepare-a-faq-document-for-embeddings/323504#:~:text=Community%20community,concatenation%20of%20all%20the%20texts). Điều này giúp bắt được cả ngôn ngữ tự nhiên lẫn ý rút gọn của khách hàng.
* Ngoài ra, cần đảm bảo **chiến lược chunking** hợp lý: vì mỗi mục FAQ đã ngắn gọn, ta giữ nguyên cụm “prompt” là một đoạn văn bản để nhúng. Quy tắc chung: *“nếu đoạn văn tự nó có nghĩa đầy đủ với con người, thì nhúng riêng đoạn đó sẽ có ý nghĩa với mô hình”*​  
  [pinecone.io](https://www.pinecone.io/learn/chunking-strategies/#:~:text=For%20example%2C%20in%20semantic%20search%2C,results%20are%20accurate%20and%20relevant). Điều này phù hợp với các mục FAQ rời rạc hiện có.

Tóm lại, **nhúng semantic cho trường "prompt"** của từng FAQ sẽ giúp chatbot hiểu đúng ý định khách hàng và tìm kiếm câu trả lời chính xác hơn.

## **2. Các phương pháp RAG hiện đại (và ngoài RAG)**

**Retrieval-Augmented Generation (RAG)** hiện là kỹ thuật chủ đạo để tăng cường LLM bằng kiến thức bên ngoài. Những phương pháp RAG mới nhất tập trung vào **cải thiện độ chính xác ngữ nghĩa và ngữ cảnh**:

* **RAG tiêu chuẩn**: Pipeline cơ bản gồm nhúng truy vấn, tìm kiếm **vector tương tự** trong kho, rồi dùng LLM sinh phản hồi dựa trên đoạn tri thức tìm được​  
  [jillanisofttech.medium.com](https://jillanisofttech.medium.com/the-evolution-of-rag-a-comprehensive-guide-to-modern-retrieval-augmented-generation-approaches-5b981af06a7e#:~:text=,retriever%3Dretriever%2C%20return_source_documents%3DTrue). Cách này đã được áp dụng rộng rãi (như Zendesk dùng RAG để truy xuất bài hỗ trợ khách hàng nhanh chóng)​  
  [jillanisofttech.medium.com](https://jillanisofttech.medium.com/the-evolution-of-rag-a-comprehensive-guide-to-modern-retrieval-augmented-generation-approaches-5b981af06a7e#:~:text=While%20simple%2C%20Naive%20RAG%20powers,many%20successful%20applications). Triển khai qua API khá dễ: ví dụ dùng OpenAI Embedding ADA-002 để tìm kiếm, sau đó gọi GPT-4/GPT-3.5 với prompt chứa ngữ cảnh​  
  [jillanisofttech.medium.com](https://jillanisofttech.medium.com/the-evolution-of-rag-a-comprehensive-guide-to-modern-retrieval-augmented-generation-approaches-5b981af06a7e#:~:text=,retriever%3Dretriever%2C%20return_source_documents%3DTrue).
* **RAG nâng cao**: Kết hợp nhiều kỹ thuật để tăng độ chính xác:  
  + **Hybrid search** (kết hợp tìm kiếm ngữ nghĩa và từ khóa): song song dùng **vector search** và **BM25** (tìm kiếm từ khóa) rồi hợp nhất kết quả, đảm bảo không bỏ sót chi tiết quan trọng​  
    [jillanisofttech.medium.com](https://jillanisofttech.medium.com/the-evolution-of-rag-a-comprehensive-guide-to-modern-retrieval-augmented-generation-approaches-5b981af06a7e#:~:text=Advanced%20RAG%20builds%20upon%20the,basic%20framework%20by%20introducing).
  + **Query expansion**: Mở rộng truy vấn bằng cách nhờ LLM tạo các cách hỏi tương tự (từ đồng nghĩa, cách diễn đạt khác)​  
    [jillanisofttech.medium.com](https://jillanisofttech.medium.com/the-evolution-of-rag-a-comprehensive-guide-to-modern-retrieval-augmented-generation-approaches-5b981af06a7e#:~:text=def%20_expand_query%28self%2C%20query%3A%20str%29%20,query%7D%20Alternative%20phrasings)​  
    [jillanisofttech.medium.com](https://jillanisofttech.medium.com/the-evolution-of-rag-a-comprehensive-guide-to-modern-retrieval-augmented-generation-approaches-5b981af06a7e#:~:text=,expand_query%28query). Sau đó lần lượt tìm kiếm bằng các truy vấn mở rộng này để bao quát hết ngữ nghĩa người dùng có thể hỏi.
  + **Multi-stage re-ranking**: Xếp hạng lại kết quả tìm kiếm qua nhiều tầng. Ví dụ, lấy top 10 kết quả từ vector search, rồi dùng **cross-encoder** (mô hình Transformers đánh giá mức độ phù hợp của cặp truy vấn - đoạn văn) để chọn ra 3 kết quả tốt nhất. Cách này tận dụng cả tốc độ của vector search lẫn độ chính xác của mô hình so khớp có ngữ cảnh​  
    [jillanisofttech.medium.com](https://jillanisofttech.medium.com/the-evolution-of-rag-a-comprehensive-guide-to-modern-retrieval-augmented-generation-approaches-5b981af06a7e#:~:text=Advanced%20RAG%20builds%20upon%20the,basic%20framework%20by%20introducing).
  + **Điều chỉnh cửa sổ ngữ cảnh động**: Với truy vấn dài hoặc phức tạp, kỹ thuật RAG mới có thể điều chỉnh độ dài đoạn trích dẫn đưa vào LLM, đảm bảo thông tin vừa đủ và nằm trọn trong ngưỡng context window của mô hình​  
    [jillanisofttech.medium.com](https://jillanisofttech.medium.com/the-evolution-of-rag-a-comprehensive-guide-to-modern-retrieval-augmented-generation-approaches-5b981af06a7e#:~:text=Advanced%20RAG%20builds%20upon%20the,basic%20framework%20by%20introducing).
* **Agentic RAG (RAG tác tử)**: Đây là kiến trúc RAG dùng nhiều **agent (tác tử)** LLM phối hợp. Ví dụ, một agent đảm nhiệm **lập kế hoạch tác vụ** – phân tích câu hỏi phức tạp và chia thành các bước con; các agent khác chuyên xử lý từng bước (tìm kiếm thông tin, thực hiện tính toán, vv.), và một **agent điều phối** sẽ tổng hợp kết quả​  
  [jillanisofttech.medium.com](https://jillanisofttech.medium.com/the-evolution-of-rag-a-comprehensive-guide-to-modern-retrieval-augmented-generation-approaches-5b981af06a7e#:~:text=Agentic%20RAG%20introduces%20sophisticated%20components%3A). Cách tiếp cận đa tác tử cho phép chatbot giải quyết truy vấn phức tạp yêu cầu lập luận nhiều bước hoặc tương tác với nhiều nguồn dữ liệu khác nhau.
* **Phương pháp ngoài RAG**: Trong một số trường hợp, có thể kết hợp hoặc thay thế RAG bằng phương pháp khác:  
  + **Knowledge base QA trực tiếp**: Với bộ FAQ cố định, ta có thể dùng mô hình so sánh câu hỏi (sentence similarity) để tìm câu trả lời **mà không cần LLM sinh văn bản**. Ví dụ, nhúng truy vấn và tính cosine với các vector câu hỏi FAQ, sau đó *trả về trực tiếp câu trả lời* tương ứng nếu độ tương đồng vượt ngưỡng. Cách này nhanh và tránh được việc LLM “hallucinate”, phù hợp nếu câu trả lời là các thông tin ngắn cố định. Tuy nhiên, nó kém linh hoạt khi người dùng hỏi phức tạp hoặc cần diễn giải.
  + **Fine-tune model**: Huấn luyện tinh chỉnh một mô hình ngôn ngữ nhỏ (vd. AlBERT, DistilBERT) trên bộ dữ liệu Hỏi-Đáp của ngân hàng. Mô hình này học trả lời trực tiếp mà không cần retrieval. Phương pháp này có thể cho đáp án rất nhanh và chính xác trong phạm vi kiến thức đã huấn luyện. Nhược điểm là thiếu linh hoạt (khó cập nhật kiến thức mới) và tốn công huấn luyện, cũng như có thể không xử lý tốt câu hỏi ngoài phạm vi.
  + **Prompt-RAG (RAG không dùng vector)**: Một kỹ thuật mới đề xuất việc **loại bỏ bước nhúng vector** bằng cách để LLM duyệt qua **mục lục** hoặc tiêu đề tài liệu để định vị câu trả lời​  
    [cobusgreyling.medium.com](https://cobusgreyling.medium.com/prompt-rag-vector-embedding-free-retrieval-augmented-generation-c37446b43cdd#:~:text=Prompt,for%20domain%20specific%20implementations)​  
    [cobusgreyling.medium.com](https://cobusgreyling.medium.com/prompt-rag-vector-embedding-free-retrieval-augmented-generation-c37446b43cdd#:~:text=The%20image%20below%20shows%20the,RAG%20approach). Ví dụ, LLM có thể được cung cấp danh sách tiêu đề FAQ và tự chọn mục liên quan, sau đó truy xuất nội dung chi tiết. Cách này giảm phụ thuộc vào vector database, nhưng chỉ hiệu quả khi tập tài liệu nhỏ và có cấu trúc rõ (như mục lục).

Nhìn chung, cho chatbot ngân hàng đòi hỏi độ chính xác cao, **RAG nâng cao** là hướng nên dùng, kết hợp tìm kiếm ngữ nghĩa + từ khóa, mở rộng truy vấn và xếp hạng lại kết quả​

[jillanisofttech.medium.com](https://jillanisofttech.medium.com/the-evolution-of-rag-a-comprehensive-guide-to-modern-retrieval-augmented-generation-approaches-5b981af06a7e#:~:text=Advanced%20RAG%20builds%20upon%20the,basic%20framework%20by%20introducing)

. Các phương pháp ngoài RAG có thể dùng bổ trợ (vd. trực tiếp trả lời từ FAQ nếu tìm được câu khớp hoàn toàn), giúp tối ưu tốc độ và độ chính xác trong những trường hợp đơn giản.

## **3. Thư viện và công cụ open-source hỗ trợ RAG (tương thích ChromaDB, Python 3.x)**

Hiện nay có nhiều thư viện mã nguồn mở mạnh mẽ để xây dựng hệ thống RAG cho chatbot. Các công cụ này tương thích tốt với Python 3.x (bao gồm bản mới nhất) và tích hợp được với **ChromaDB** làm kho vector. Dưới đây là một số lựa chọn nổi bật và tính năng chính:

| **Công cụ/Thư viện** | **Đặc điểm chính** | **Hỗ trợ liên quan** |
| --- | --- | --- |
| **LangChain + LangGraph** | Framework linh hoạt để tạo ứng dụng LLM, hỗ trợ **chuỗi truy vấn** (Chains) và **agent**. LangChain tích hợp sẵn nhiều vector DB (Chroma, FAISS...) và LLM API. LangGraph (một module của LangChain) mở rộng khả năng xây dựng chatbot dạng đồ thị trạng thái, cho phép quản lý phiên tương tác, gọi tool và chuyển đổi giữa các node hội thoại phức tạp​  [langchain-ai.github.io](https://langchain-ai.github.io/langgraph/tutorials/introduction/#:~:text=%E2%9C%85%20Answer%20common%20questions%20by,state%20to%20control%20its%20behavior)  ​  [langchain-ai.github.io](https://langchain-ai.github.io/langgraph/tutorials/introduction/#:~:text=1.%20Multi,right%20tool%20for%20each%20task)  . | • Tích hợp ChromaDB trực tiếp​  [blog.futuresmart.ai](https://blog.futuresmart.ai/using-langchain-and-open-source-vector-db-chroma-for-semantic-search-with-openais-llm#:~:text=Using%20Langchain%20and%20Open%20Source,with%20Langchain%20for%20semantic%20search)  .  • Hỗ trợ gọi OpenAI, Cohere, Huggingface Hub… qua API.  • Cộng đồng lớn, cập nhật nhanh. |
| **LlamaIndex (GPT Index)** | Thư viện tập trung vào **xử lý tài liệu và truy vấn**. Cho phép xây dựng **chỉ mục** từ tài liệu (FAQ, văn bản) và truy vấn tự nhiên thông qua LLM. LlamaIndex tối ưu việc chia nhỏ tài liệu, tạo index linh hoạt (theo tree, list, vector) và trích xuất thông tin liên quan một cách hiệu quả. | • Kết nối được với Chroma (hoặc các vector store khác) để lưu index.  • Dễ triển khai QA cho tập tài liệu nhỏ.  • Hỗ trợ cả API lẫn mô hình tự host. |
| **Haystack (deepset)** | Khung công tác end-to-end cho **tìm kiếm và hỏi đáp bằng LLM**. Haystack thiết kế dưới dạng **nodes & pipelines** – mỗi node là một thành phần (tìm kiếm, sinh trả lời, etc.) trong một pipeline. Ví dụ: node Retriever để truy xuất tài liệu, node PromptNode dùng LLM trả lời trên ngữ cảnh​  [medium.com](https://medium.com/aimonks/haystack-an-alternative-to-langchain-carrying-llms-bf7c515c9a7e#:~:text=Haystack%20is%20your%20doorway%20to,friendly%20way)  ​  [medium.com](https://medium.com/aimonks/haystack-an-alternative-to-langchain-carrying-llms-bf7c515c9a7e#:~:text=Think%20of%20nodes%20as%20specialized,answering)  . Haystack hỗ trợ sẵn các thành phần: DensePassageRetriever, Transformers Reader (trích xuất), PromptNode (GPT/LLM)… | • Tích hợp nhiều backend: Elastic, FAISS, Weaviate, v.v. (Chroma có thể tích hợp bằng wrapper).  • Hỗ trợ đa ngôn ngữ (ví dụ tutorial dùng Cohere đa ngôn ngữ)​  [medium.com](https://medium.com/deepset-ai/multilingual-generative-question-answering-with-haystack-and-cohere-96ff1d693bf1#:~:text=Multilingual%20Generative%20Question%20Answering%20with,using%20Cohere%20models%20and%20Haystack)  .  • Phù hợp triển khai thực tế (có REST API, pipeline linh hoạt). |

Các công cụ trên đều **mã nguồn mở** và tương thích Python 3.10+ (thử nghiệm trên Python 3.11, 3.12 đều ổn). Ngoài ra, một số thư viện khác cũng hữu ích:

* **ChromaDB**: bản thân Chroma là **vector database** gọn nhẹ, chạy cục bộ, rất phù hợp cho prototyping RAG. Chroma cung cấp API Python thân thiện để lưu và tìm kiếm vector (được LangChain hỗ trợ sẵn​  
  [blog.futuresmart.ai](https://blog.futuresmart.ai/using-langchain-and-open-source-vector-db-chroma-for-semantic-search-with-openais-llm#:~:text=Using%20Langchain%20and%20Open%20Source,with%20Langchain%20for%20semantic%20search)).
* **Hugging Face Transformers & SentenceTransformers**: cung cấp các mô hình embedding mở như *multi-qa-MiniLM, LaBSE*… để nhúng ngữ nghĩa văn bản. Cũng có pipeline trả lời (QA) dạng trích xuất có thể kết hợp trong pipeline nếu cần kiểm chứng câu trả lời.
* **Công cụ dịch**: Đối với song ngữ Anh–Lào, có thể dùng thư viện dịch mã mở (như MarianMT của Huggingface) hoặc API như Google Translate, DeepL cho bước dịch (nếu áp dụng chiến lược dịch, xem mục 5).

Tùy vào nhu cầu, nhóm phát triển có thể kết hợp các thư viện trên: ví dụ dùng **LangChain** làm khung chính orchestrator (kết hợp với LangGraph để quản lý hội thoại phức tạp), bên dưới sử dụng **ChromaDB** lưu vector embedding tạo bởi **SentenceTransformers** (mô hình đa ngôn ngữ), và **Haystack** để tận dụng module re-ranker hoặc tích hợp vào hệ thống tìm kiếm hiện có. Tất cả đều chạy tốt trên Python 3.13 khi phát hành.

## **4. Đánh giá các nền tảng LangGraph/framework hội thoại tốt nhất hiện nay**

Để xây dựng chatbot web (và dễ mở rộng đa kênh), ta cần một framework quản lý **luồng hội thoại (conversation flow)** và logic phân nhánh. Hiện có hai hướng chính: (1) Sử dụng **LangGraph** (hoặc các công cụ LLM agents tương tự) cho hội thoại linh hoạt; (2) Sử dụng **khung hội thoại truyền thống** như Rasa, kết hợp khả năng của LLM. Dưới đây là đánh giá:

* **LangGraph (LangChain)** – *nền tảng hiện đại cho chatbot dùng LLM*: LangGraph định nghĩa luồng hội thoại dạng **đồ thị trạng thái** (state graph). Mỗi **node** có thể là một hành động: gọi LLM, gọi hàm/tool, hoặc logic điều kiện; các **edge** xác định điều kiện chuyển tiếp giữa các node. Ưu điểm:  
  + **Kiểm soát dòng hội thoại**: Lập trình viên có thể thiết kế các bước hội thoại phức tạp, ví dụ: trước tiên **tìm kiếm** câu trả lời FAQ, nếu không tìm thấy thì **hỏi thông tin bổ sung** hoặc **chuyển người** hỗ trợ. LangGraph hỗ trợ duy trì **state** qua nhiều lượt tương tác, giúp chatbot nhớ ngữ cảnh​  
    [langchain-ai.github.io](https://langchain-ai.github.io/langgraph/tutorials/introduction/#:~:text=%E2%9C%85%20Answer%20common%20questions%20by,state%20to%20control%20its%20behavior).
  + **Đa tác vụ, đa công cụ**: Cho phép tích hợp nhiều *tool/agent*. Ví dụ node A dùng LLM trả lời FAQ, node B gọi API tra cứu số dư tài khoản, node C kích hoạt thông báo cho nhân viên. LangGraph thiết kế để agent có thể sử dụng nhiều công cụ và chọn đúng tool cho từng yêu cầu​  
    [langchain-ai.github.io](https://langchain-ai.github.io/langgraph/tutorials/introduction/#:~:text=1.%20Multi,right%20tool%20for%20each%20task).
  + **Điều hướng linh hoạt & can thiệp con người**: Có thể cài đặt để bot **nhận biết giới hạn** – ví dụ khi câu hỏi quá phức tạp hoặc nằm ngoài phạm vi, bot sẽ chuyển sang node “chuyển cuộc chat cho nhân viên”​  
    [langchain-ai.github.io](https://langchain-ai.github.io/langgraph/tutorials/introduction/#:~:text=%E2%9C%85%20Answer%20common%20questions%20by,state%20to%20control%20its%20behavior). LangGraph cũng hỗ trợ “rewind” quay lại trạng thái trước, thử đường khác nếu cần​  
    [langchain-ai.github.io](https://langchain-ai.github.io/langgraph/tutorials/introduction/#:~:text=%E2%9C%85%20Answer%20common%20questions%20by,state%20to%20control%20its%20behavior).
  + **Tích hợp tự nhiên với RAG**: Vì LangGraph nằm trong hệ sinh thái LangChain, rất dễ để nhúng chuỗi RAG vào một node. Ví dụ node “Trả lời FAQ” sẽ thực hiện: nhận câu hỏi -> dùng retriever ChromaDB -> gọi GPT-4 trả lời. Sau đó node kế có thể kiểm tra *confidence* của câu trả lời, nếu thấp thì chuyển hướng sang kịch bản khác.
  + **Hạn chế**: LangGraph còn mới (ra mắt khoảng cuối 2024), cần người phát triển có kỹ năng Python. Việc triển khai đa kênh phải tự xây dựng lớp kết nối (ví dụ viết một service Flask nhận message từ webchat rồi gọi LangGraph). Tuy nhiên, do logic tách bạch, việc tích hợp kênh mới (mobile app, chat FB…) chỉ là thêm **adapter** giao tiếp, logic cốt lõi không cần viết lại.
* **Rasa (Open source)** – *nền tảng hội thoại truyền thống, tùy biến cao*: Rasa nổi tiếng về khả năng **NLU + quản lý dialog** mạnh mẽ, cho phép triển khai chatbot đa kênh dễ dàng. Ưu điểm:  
  + **Multi-channel out-of-the-box**: Rasa hỗ trợ kết nối sẵn đến Web chat, Facebook Messenger, Zalo (thông qua REST), v.v. Bạn có thể triển khai bot Rasa một lần và **cấu hình các connector** để nhận/gửi tin nhắn qua các kênh khác nhau​  
    [quidget.ai](https://quidget.ai/blog/ai-automation/top-5-open-source-chatbot-platforms/#:~:text=,other%20info%20your%20bot%20needs).
  + **Dialog flow dựa trên stories/rules**: Rasa dùng cách định nghĩa **stories** (kịch bản hội thoại mẫu) và **rules** để điều hướng bot. Ta có thể quy định: nếu **intent = hỏi\_số\_dư** và đã thu thập đủ thông tin tài khoản, thì thực hiện action tra cứu; nếu thiếu, thì hỏi lại cho đủ. Cách này đảm bảo bot đi đúng flow cho các nhiệm vụ định trước (ví dụ quy trình xác thực OTP, tra cứu tài khoản,…).
  + **Kết hợp LLM**: Phiên bản Rasa mới (từ 3.5+) đã bắt đầu tích hợp LLM trong pipeline (gọi là Rasa Labs - Conversational AI with LLM)​  
    [rasa.com](https://rasa.com/docs/rasa/next/llms/large-language-models/#:~:text=The%20recent%20advances%20in%20large,tuning%20of%20large%20models)​  
    [rasa.com](https://rasa.com/docs/rasa/next/llms/large-language-models/#:~:text=An%20adjustable%20risk%20profile). Ta có thể dùng LLM để phân loại ý định hoặc sinh câu trả lời linh hoạt khi cần. Thậm chí có dự án **RasaGPT** kết hợp Rasa với LangChain để tận dụng LLM cho trả lời tự do dựa trên tri thức, trong khi vẫn giữ được khung hội thoại có kiểm soát của Rasa​  
    [github.com](https://github.com/paulpierre/RasaGPT#:~:text=RasaGPT%20is%20the%20first%20headless,implementation%20of%20Rasa%20and%20Telegram).
  + **Quản lý ngữ cảnh và slot**: Rasa hỗ trợ khái niệm **slot (biến lưu thông tin)**, giúp duy trì ngữ cảnh (context) rõ ràng. Điều này hữu ích trong ngân hàng: ví dụ sau khi khách hàng cung cấp số tài khoản, các lượt hỏi kế tiếp bot không cần hỏi lại số tài khoản, mà dùng giá trị đã lưu.
  + **Nhược điểm**: Xây dựng bot Rasa truyền thống cần dữ liệu huấn luyện (câu mẫu cho intent, entity). Việc tích hợp RAG đòi hỏi viết **custom action** gọi đến vector DB và LLM, không có sẵn như LangChain. Ngoài ra, nếu lạm dụng LLM sinh text tự do trong Rasa, có nguy cơ bot nói lệch kịch bản kiểm soát.

**Kết luận lựa chọn:** Nếu ưu tiên *tính hiện đại, linh hoạt LLM* và có đội ngũ Python mạnh, **LangChain/LangGraph** là lựa chọn tốt để xây dựng chatbot từ đầu, vì dễ tích hợp RAG và mở rộng logic theo nhu cầu đặc thù. LangGraph giúp tạo ra một **conversation flow có điều kiện** mà vẫn tận dụng được sức mạnh LLM. Tuy nhiên, nếu cần triển khai nhanh đa kênh và tận dụng các **flow hội thoại chuẩn hóa** (như xử lý yêu cầu tra cứu thông tin cá nhân, giao dịch…), có thể kết hợp **Rasa** như một lớp điều phối kênh và xử lý những intent cố định, đồng thời gọi LangChain (LLM) trong những trường hợp cần trả lời linh hoạt từ kiến thức. Nhiều tổ chức tích hợp hai hướng này để đạt hiệu quả cao và kiểm soát tốt.

## **5. Chiến lược xử lý song ngữ Anh – Lào hiệu quả**

Hỗ trợ hai ngôn ngữ đòi hỏi thiết kế cẩn thận ở bước nhúng và sinh phản hồi. Có một số phương án chính:

**(a) Dùng mô hình nhúng đa ngôn ngữ**: Sử dụng mô hình embedding đa ngôn ngữ (multilingual) để **nhúng cả tiếng Anh và tiếng Lào vào cùng không gian vector**. Ví dụ, các model như **LaBSE**, **multilingual MPNet**, hoặc mới nhất là **OpenAI text-embedding-ada-002** đều được huấn luyện trên nhiều ngôn ngữ. Cách này cho phép: câu hỏi tiếng Lào và câu hỏi tiếng Anh tương đương sẽ cho vector gần nhau, và so khớp với cùng vector của câu trả lời trong kho. Ưu điểm: **bỏ qua bước dịch**, giảm độ trễ và tránh mất sắc thái do dịch thuật​

[medium.com](https://medium.com/contact-research/how-to-deal-with-different-language-questions-in-your-rag-application-714eb3ccb772#:~:text=Multilingual%20embeddings%20are%20designed%20to,nuances%20of%20the%20original%20query)

. Tuy nhiên, nhược điểm là độ chính xác có thể hơi giảm so với mô hình đơn ngữ tối ưu cho một ngôn ngữ​

[medium.com](https://medium.com/contact-research/how-to-deal-with-different-language-questions-in-your-rag-application-714eb3ccb772#:~:text=These%20results%20indicate%20that%20while,always%20deliver%20the%20highest%20recall)

. Nghiên cứu cho thấy nhúng đa ngôn ngữ tuy tiện lợi nhưng đôi khi **recall thấp hơn** so với cách dịch sang một ngôn ngữ rồi tìm kiếm​

[medium.com](https://medium.com/contact-research/how-to-deal-with-different-language-questions-in-your-rag-application-714eb3ccb772#:~:text=Multilingual%20embeddings%20can%20be%20effective%2C,model%2C%20which%20we%20ran%20locally)

​

[medium.com](https://medium.com/contact-research/how-to-deal-with-different-language-questions-in-your-rag-application-714eb3ccb772#:~:text=These%20results%20indicate%20that%20while,always%20deliver%20the%20highest%20recall)

.

**(b) Dịch truy vấn trước khi nhúng (Translate-then-Embed)**: Đây là phương án nhiều hệ thống sử dụng để đảm bảo độ chính xác. Cụ thể: **phát hiện ngôn ngữ** đầu vào, nếu là tiếng Lào thì dùng bộ dịch (máy hoặc LLM) dịch sang **tiếng Anh**, sau đó nhúng bằng mô hình tiếng Anh (ví dụ sentence-transformers Anh). Toàn bộ kho FAQ có thể lưu vector theo ngôn ngữ cơ sở (tiếng Anh). Khi trả lời, nếu câu hỏi gốc là tiếng Lào thì **dịch ngược câu trả lời sang Lào**. Ưu điểm: tận dụng được sức mạnh của các mô hình nhúng tối ưu cho tiếng Anh – vốn thường có hiệu quả cao hơn. Thực nghiệm chỉ ra phương pháp này cho kết quả truy xuất tốt hơn đôi chút so với nhúng đa ngôn ngữ trực tiếp​

[medium.com](https://medium.com/contact-research/how-to-deal-with-different-language-questions-in-your-rag-application-714eb3ccb772#:~:text=their%20merits,Therefore%2C%20choosing%20the%20right)

(ngay cả khi dùng mô hình dịch nhỏ)​

[medium.com](https://medium.com/contact-research/how-to-deal-with-different-language-questions-in-your-rag-application-714eb3ccb772#:~:text=Both%20methods%20for%20handling%20different,RAG%20systems%20for%20multilingual%20users)

. Nhược điểm: phụ thuộc chất lượng bộ dịch. Cần lựa chọn engine dịch Anh–Lào tốt để không dịch sai ý user. Nếu dùng API dịch, cũng cân nhắc vấn đề chi phí và độ trễ.

**(c) Chỉ mục song song theo từng ngôn ngữ**: Xây dựng hai kho kiến thức riêng cho Anh và Lào. Ví dụ, ngân hàng có thể chuẩn bị sẵn **bản dịch tiếng Lào cho mỗi câu hỏi/đáp** trong FAQ. Sau đó, ta tạo hai bộ vector: một cho nội dung tiếng Anh, một cho tiếng Lào (dùng mô hình nhúng tương ứng tối ưu cho từng ngôn ngữ). Khi người dùng hỏi, dựa vào ngôn ngữ, hệ thống truy vấn trong **kho tương ứng**. Cách này đảm bảo độ chính xác cao (vì dùng mô hình bản địa cho từng ngôn ngữ), nhưng tốn kém công sức quản lý: phải duy trì đồng bộ nội dung hai ngôn ngữ. Nếu có sự thay đổi, phải cập nhật cả hai. Ngoài ra, mô hình Lào chất lượng cao có thể khó tìm hơn.

**Khuyến nghị:** Nếu ngân hàng đã có sẵn dữ liệu FAQ tiếng Anh và chưa có bản tiếng Lào đầy đủ, phương án (b) – **dịch truy vấn và phản hồi** – là khả thi và thực dụng nhất để sớm ra mắt chatbot. Cụ thể: nhúng toàn bộ FAQ tiếng Anh vào ChromaDB bằng mô hình Anh (hoặc đa ngữ mạnh như ada-002). Khi người dùng hỏi tiếng Lào, bot dùng **LangGraph** gọi một node “dịch Lào->Anh” (có thể dùng GPT-4 API hoặc mô hình NLLB của Facebook nếu muốn self-host). Sau đó node tiếp theo thực hiện RAG tìm câu trả lời bằng tiếng Anh, rồi dùng một node nữa dịch câu trả lời sang tiếng Lào trước khi gửi lại người dùng. Phương án này đã được chứng minh cho chất lượng truy xuất cao hơn​

[medium.com](https://medium.com/contact-research/how-to-deal-with-different-language-questions-in-your-rag-application-714eb3ccb772#:~:text=their%20merits,Therefore%2C%20choosing%20the%20right)

. Về lâu dài, nếu lượng câu hỏi tiếng Lào nhiều và phong phú, ngân hàng có thể xây dựng dần kho dữ liệu Lào song song và tiến tới nhúng trực tiếp không cần dịch.

Cuối cùng, cần chú ý **giữ nhất quán ngôn ngữ trong phản hồi**: nếu khách hỏi bằng tiếng nào, trả lời bằng tiếng đó. Hệ thống nên thiết kế để xác định ngôn ngữ đầu vào và gắn cờ, sử dụng cờ này cho bước xử lý cuối cùng (dịch output hoặc chọn kênh trả lời phù hợp).

## **6. Đảm bảo độ chính xác cao: semantic & context match**

Với chatbot ngân hàng, sai lệch thông tin là không thể chấp nhận. Do đó, hệ thống cần các biện pháp đảm bảo **kết quả truy vấn thật chuẩn xác về ngữ nghĩa và ngữ cảnh**:

* **Mô hình nhúng chất lượng và phù hợp ngôn ngữ**: Chọn mô hình embedding đã được đánh giá cao trên ngôn ngữ và lĩnh vực liên quan. Ví dụ: dùng **multilingual-e5-large** của Microsoft hoặc **Cohere multilingual v3** nếu hỗ trợ tiếng Lào, vì theo thử nghiệm những mô hình này cho kết quả tốt trong nhiều ngữ cảnh​  
  [medium.com](https://medium.com/contact-research/how-to-deal-with-different-language-questions-in-your-rag-application-714eb3ccb772#:~:text=match%20at%20L159%20In%20our,72). Mô hình tốt giúp vector của câu hỏi và câu trả lời đúng nằm gần nhau hơn, tăng khả năng truy xuất chính xác.
* **Kết hợp từ khóa để không bỏ sót ngữ cảnh quan trọng**: Trong lĩnh vực ngân hàng, nhiều khi truy vấn chứa các từ khóa đặc thù (tên sản phẩm, mã giao dịch, con số…). Sử dụng **hybrid search (vector + từ khóa)** sẽ giúp tìm đúng kết quả ngay cả khi embedding chưa thể hiện hết thông tin quan trọng​  
  [jillanisofttech.medium.com](https://jillanisofttech.medium.com/the-evolution-of-rag-a-comprehensive-guide-to-modern-retrieval-augmented-generation-approaches-5b981af06a7e#:~:text=Advanced%20RAG%20builds%20upon%20the,basic%20framework%20by%20introducing). Ví dụ: câu hỏi “lãi suất tiết kiệm kỳ hạn 12 tháng” – embedding đảm bảo hiểu ý chung, còn BM25 đảm bảo những kết quả chứa “12 tháng” và “lãi suất” được ưu tiên.
* **Thu hẹp phạm vi tìm kiếm theo ngữ cảnh**: Tận dụng metadata như **topic/intent** của FAQ. Nếu có bộ phân loại ý định hoặc nhận dạng chủ đề (ví dụ intent = *thẻ tín dụng* vs *tài khoản*), ta có thể **filter** kho vector theo chủ đề tương ứng trước khi tìm kiếm. Điều này giảm nhiễu, tăng độ chính xác vì so sánh vector chỉ diễn ra trong tập dữ liệu liên quan.
* **Rerank kết quả bằng cross-encoder**: Như đề cập ở mục 2, việc dùng một mô hình ngôn ngữ (vd. **MiniLM cross-encoder đã fine-tune trên dữ liệu QA**) để đánh giá lại top K đoạn trích dẫn sẽ tăng độ chính xác. Cross-encoder đọc cả truy vấn và đoạn văn, cho điểm phù hợp ngữ nghĩa ở mức cao hơn so với so sánh vector thuần túy. Nhờ đó, chatbot chọn đúng nhất câu trả lời khớp ý người hỏi​  
  [jillanisofttech.medium.com](https://jillanisofttech.medium.com/the-evolution-of-rag-a-comprehensive-guide-to-modern-retrieval-augmented-generation-approaches-5b981af06a7e#:~:text=,Dynamic%20context%20window%20adjustment).
* **Kiểm soát bởi logic LangGraph**: LangGraph có thể giúp “gác cổng” câu trả lời của LLM. Ta có thể thiết lập node kiểm tra: nếu **LLM không tự tin** (ví dụ xác suất thấp hoặc câu trả lời quá ngắn/dài bất thường), thì không gửi ngay cho khách, mà chuyển sang hành động khác (như xin người dùng làm rõ hỏi lại, hoặc tra cứu dạng khác). Cách này thêm một lớp an toàn, tránh trả lời sai tự tin.
* **Prompt thiết kế chặt chẽ**: Khi gọi LLM, dùng prompt mẫu nhấn mạnh yêu cầu: *“Trả lời* ***chỉ dựa trên thông tin sau****. Nếu không chắc chắn, hãy nói không biết.”* và cung cấp chính xác đoạn văn liên quan. Điều này khiến LLM bám sát context có được, giảm nguy cơ tạo thông tin không có thật. Ngoài ra, có thể thêm hướng dẫn để LLM **trích dẫn nguồn** (ví dụ: tài liệu hay tên biểu phí) nếu phù hợp, tăng độ tin cậy của câu trả lời.
* **Test và tinh chỉnh liên tục**: Giai đoạn đầu, cần kiểm thử chatbot trên tập câu hỏi đa dạng của khách. Theo dõi các trường hợp bot trả lời chưa tốt để cập nhật: có thể bổ sung dữ liệu đào tạo cho mô hình phân loại intent, thêm mẫu paraphrase vào tập mở rộng truy vấn, hoặc hiệu chỉnh ngưỡng độ tương đồng để quyết định khi nào nên trả lời vs khi nào nên xin lỗi vì không tìm thấy câu trả lời phù hợp.

Tóm lại, độ chính xác được đảm bảo tốt nhất bằng cách **kết hợp nhiều tầng**: (1) Tìm kiếm đúng dữ liệu (vector + từ khoá + filter ngữ cảnh), (2) Chọn lọc kết quả qua mô hình ngôn ngữ bổ trợ, (3) Ràng buộc LLM trả lời trong giới hạn thông tin cho phép. Cách tiếp cận đa tầng này sẽ cho chatbot khả năng **hiểu đúng và đáp đúng** gần như mọi câu hỏi hợp lệ của khách hàng.

## **7. Sơ đồ cấu trúc hệ thống chatbot RAG + LangGraph đa kênh**

*Sơ đồ pipeline RAG kết hợp LangGraph cho chatbot song ngữ đa kênh.* Sơ đồ mô tả luồng xử lý từ khi người dùng gửi câu hỏi đến khi nhận được phản hồi. Trước tiên, **người dùng** (có thể trên web, mobile, hoặc kênh chat nào đó) gửi câu hỏi bằng tiếng Anh hoặc Lào. Hệ thống sẽ qua các bước: (1) **LangGraph Orchestrator** nhận message, xác định ngôn ngữ. Nếu cần, bước (2) **Translation Service** dịch câu hỏi sang ngôn ngữ cơ sở (vd. tiếng Anh). Tiếp đó, (3) truy vấn đã chuẩn hóa được **nhúng bằng mô hình ngôn ngữ** và tìm kiếm **trong kho vector ChromaDB** để lấy ra những đoạn kiến thức liên quan nhất. (4) Các đoạn này cùng với câu hỏi được chuyển cho **LLM (mô hình ngôn ngữ lớn)** – ví dụ GPT-4 hoặc một mô hình nội bộ – để sinh ra câu trả lời phù hợp. Câu trả lời có thể trải qua bước hậu xử lý trong LangGraph (như dịch ngược sang tiếng Lào nếu người hỏi dùng tiếng Lào, hoặc định dạng lại theo yêu cầu ngân hàng). Cuối cùng, (5) **phản hồi** được gửi trả về người dùng qua kênh tương ứng. Toàn bộ quy trình được điều phối bởi LangGraph, cho phép bổ sung các nhánh xử lý khác nếu cần (như yêu cầu xác thực, tra cứu cơ sở dữ liệu nội bộ, hoặc chuyển human agent). Với kiến trúc này, chatbot có thể **hiểu đúng ý khách hàng và cung cấp câu trả lời chính xác**, đồng thời dễ dàng mở rộng ra nhiều kênh giao tiếp và tình huống đối thoại phức tạp trong lĩnh vực ngân hàng.

**Nguồn tham khảo:** Các công cụ và phương pháp trình bày trên được tổng hợp từ tài liệu kỹ thuật và kinh nghiệm triển khai thực tế, bao gồm hướng dẫn RAG đa ngôn ngữ​

[medium.com](https://medium.com/contact-research/how-to-deal-with-different-language-questions-in-your-rag-application-714eb3ccb772#:~:text=Both%20methods%20for%20handling%20different,RAG%20systems%20for%20multilingual%20users)

​

[medium.com](https://medium.com/contact-research/how-to-deal-with-different-language-questions-in-your-rag-application-714eb3ccb772#:~:text=their%20merits,Therefore%2C%20choosing%20the%20right)

, kỹ thuật RAG nâng cao từ cộng đồng (vd. hybrid search, rerank)​

[jillanisofttech.medium.com](https://jillanisofttech.medium.com/the-evolution-of-rag-a-comprehensive-guide-to-modern-retrieval-augmented-generation-approaches-5b981af06a7e#:~:text=Advanced%20RAG%20builds%20upon%20the,basic%20framework%20by%20introducing)

, cũng như tài liệu về LangGraph và Rasa cho xây dựng chatbot thông minh​

[langchain-ai.github.io](https://langchain-ai.github.io/langgraph/tutorials/introduction/#:~:text=%E2%9C%85%20Answer%20common%20questions%20by,state%20to%20control%20its%20behavior)

​

[quidget.ai](https://quidget.ai/blog/ai-automation/top-5-open-source-chatbot-platforms/#:~:text=,other%20info%20your%20bot%20needs)

. Những giải pháp đề xuất kỳ vọng sẽ giúp phát triển thành công chatbot AI song ngữ Anh–Lào cho ngân hàng với độ chính xác cao và trải nghiệm người dùng xuất sắc.