Cẩm nang Kỹ thuật AI

≈ Hiện đại ≈

*"Xây dựng các hệ thống AI **thông minh, đáng tin cậy** và **sẵn sàng triển khai**."*

① AI Engineering là gì?

AI Engineering là hoạt động xây dựng, triển khai và vận hành các ứng dụng được hỗ trợ bởi AI trong thế giới thực.

Nó kết hợp Mô hình AI/ML + Kỹ thuật Phần mềm + Kỹ thuật Dữ liệu để tạo ra các hệ thống có khả năng mở rộng và đáng tin cậy.

☆ **Mục tiêu:** Mang lại giá trị cho người dùng bằng AI một cách an toàn, hiệu quả và có đạo đức.

② Lộ trình AI Engineer (2026)

1. Học Python thật vững
2. Giỏi Cấu trúc dữ liệu & Giải thuật
3. Toán cho ML (Đại số tuyến tính, XS, TK)
4. Nền tảng ML & Deep Learning
5. LLM, Prompting & RAG
6. Xây dựng dự án AI & Triển khai
7. Thiết kế hệ thống cho ứng dụng AI

↑ Bắt đầu từ đây

↓ Trở thành Pro

③ AI vs ML vs Deep Learning vs Generative AI

AI

Lĩnh vực rộng về việc làm cho máy móc trở nên thông minh.
Gồm luật lệ, tìm kiếm, logic, ML, Robotics, NLP,...

→

ML (Machine Learning)

Tập con của AI.
Máy học từ dữ liệu và tự cải thiện mà không cần lập trình tường minh.

→

Deep Learning

Tập con của ML.
Dùng Mạng nơ-ron nhiều lớp để học các mẫu từ dữ liệu lớn.

→

Generative AI

Tập con của Deep Learning.
Mô hình tạo nội dung mới như văn bản, ảnh, mã, âm thanh.

④ LLM giải thích đơn giản

LLM (Mô hình Ngôn ngữ Lớn) là một loại mô hình nền tảng được huấn luyện trên lượng văn bản khổng lồ để hiểu và tạo ra văn bản giống con người.

Dữ liệu văn bản khổng lồ → Tiền huấn luyện (Dự đoán token kế tiếp) → Tinh chỉnh (Tùy chọn) →

LLM (GPT-4, Claude, Llama,...)

⑤ AI Engineering Stack (Tổng quan)

Mô hình	OpenAI, Llama, Mistral, Claude
Frameworks	LangChain, LlamaIndex, LangGraph
Dữ liệu	Pinecone, Weaviate, Chroma, FAISS
Hạ tầng	AWS, GCP, Azure, Vercel, Hugging Face
Công cụ	Python, FastAPI, Docker, Git, Kubernetes
Observability	LangSmith, Weights & Biases, Prometheus
Frontend	Next.js, React, Streamlit, Gradio

☆ Kinh nghiệm phỏng vấn

Tập trung vào nền tảng + thiết kế hệ thống + dự án thực tế

⚠ Sai lầm thường gặp

- Chỉ học prompting
- Bỏ qua DSA & Thiết kế hệ thống

✓ Thực hành tốt

- Xây từ đơn giản → phức tạp
- Hiểu cách vận hành

✦ Ghi nhớ

AI Engineering là 20% mô hình + 80% kỹ thuật.

② Cách LLM hoạt động

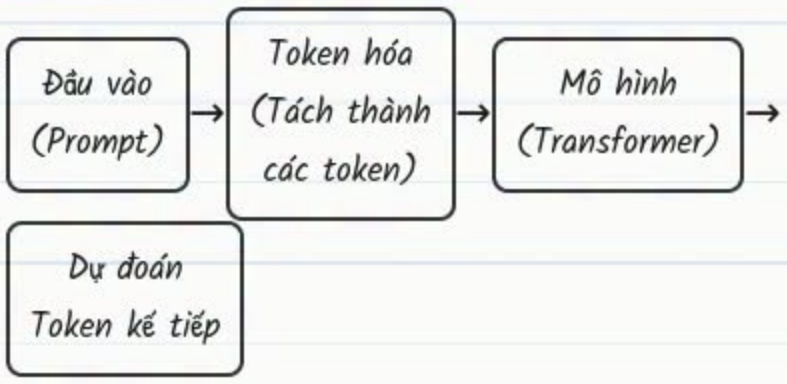
AI

LLM (Mô hình Ngôn ngữ Lớn) được huấn luyện trên lượng văn bản khổng lồ để dự đoán token kế tiếp.

1. Ý tưởng cơ bản

Bạn đưa vào một đoạn văn bản (prompt) → Mô hình đọc nó → hiểu mẫu hình → dự đoán token có khả năng cao nhất tiếp theo → lặp lại bước này liên tục.

2. Quy trình xử lý



Ví dụ:
Prompt: "AI is going to"
Token dự đoán: change

Tạo phản hồi

3. Token

Văn bản được chia nhỏ thành các mảnh gọi là token.

Ví dụ: "Hello AI"

4. Embedding

Mỗi token được chuyển thành một vector (dãy số) đại diện cho ý nghĩa của nó.

Ví dụ:
"Hello" → [0.12, -0.31, 0.85, ...]

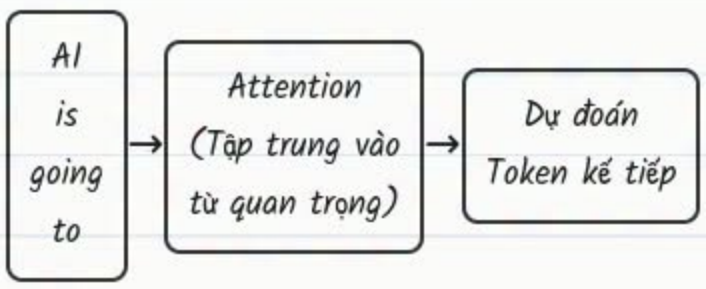
5. Context Window (Cửa sổ ngữ cảnh)

LLM nhìn vào một số token gần nhất (context window) để hiểu ý nghĩa và dự đoán token tiếp theo.

↕ Mô hình dự đoán token kế tiếp - vùng bên dưới là **Context Window**

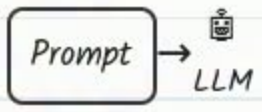
6. Transformer (Nói đơn giản)

Transformer dùng cơ chế attention để tập trung vào các token quan trọng trong ngữ cảnh và dự đoán token kế tiếp.



💡 **Điều cốt lõi:** *LLM về cơ bản là những cỗ máy dự đoán từ tiếp theo cực kỳ giỏi, được huấn luyện trên một lượng dữ liệu khổng lồ.*

③ Prompt Engineering



Prompt Engineering là nghệ thuật viết hướng dẫn tốt hơn để nhận được kết quả tốt hơn từ LLM.

1. Prompt là gì?

Prompt là nội dung đầu vào hoặc hướng dẫn mà ta đưa cho LLM.

Prompt tốt = Kết quả tốt hơn, chính xác và hữu ích.

Prompt tồi = Kết quả gây nhầm lẫn, sai hoặc không đầy đủ.

2. Các kiểu Prompting

- **Zero-shot** → Mô hình làm nhiệm vụ mà không cần ví dụ.

VD: Dịch sang tiếng Việt: "Good morning"

- **One-shot** → Mô hình được cho một ví dụ.

Q: 2+2 = 4 Q: 3+3 = ?

- **Few-shot** → Mô hình được cho vài ví dụ.

Q: Thủ đô Ấn Độ? A: New Delhi.

Q: Thủ đô Pháp? A: Paris.

- **Chain of Thought (CoT)** → Mô hình suy nghĩ từng bước.

"Hãy suy nghĩ từng bước." Q: Nếu 3x+2=11, x bằng bao nhiêu?

3. Nguyên tắc viết Prompt tốt

- ☑ Rõ ràng và cụ thể
- ☑ Cung cấp ngữ cảnh
- ☑ Yêu cầu định dạng đầu ra mong muốn
- ☑ Chia nhỏ nhiệm vụ phức tạp thành từng bước
- ☑ Dùng ví dụ nếu cần
- ☑ Lặp lại và cải thiện prompt

Prompt tồi ✗

Giải thích về AI.

Prompt tốt ✓

Giải thích Trí tuệ Nhân tạo bằng ngôn ngữ đơn giản cho người mới bắt đầu, trong 5 gạch đầu dòng.

4. Mẫu Prompt

Vai trò: Bạn là chuyên gia về _____.
Nhiệm vụ: _____
Ngữ cảnh: _____

Định dạng: _____
Giọng văn: _____
Đầu ra: _____

5. Ví dụ – Từ tồi đến tốt

✗ Prompt tồi

Viết về JavaScript.

✓ Prompt tốt

Bạn là giáo viên JavaScript. Hãy giải thích JavaScript bằng ngôn ngữ đơn giản cho người mới bắt đầu trong 5 gạch đầu dòng, kèm ví dụ.

6. Kỹ thuật nâng cao

- Role Prompting (đóng vai)
- Structured Output (đầu ra có cấu trúc)
- Self Consistency
- Prompt Chaining (chuỗi prompt)
- ReAct (Reason + Act)
- Quardrails & Constraints (giới hạn & ràng buộc)

↺ Tinh chỉnh & Lặp lại

☆ Kinh nghiệm phỏng vấn

Nhà tuyển dụng có thể hỏi: các kỹ thuật prompting, cách cải thiện đầu ra không ổn định.

⚠ Sai lầm thường gặp

✗ Prompt mơ hồ và không rõ ràng.

💡 Mẹo nhanh

Luôn cho mô hình biết định dạng đầu ra mong muốn.

④ RAG (Truy xuất tăng cường sinh nội dung)

RAG kết hợp sức mạnh của LLM với kiến thức bên ngoài để đưa ra câu trả lời chính xác, cập nhật và đáng tin cậy hơn.

1. Cách RAG hoạt động (Tổng quan)

1. Câu hỏi người dùng
Người dùng hỏi điều gì đó.
→ 2. Truy xuất
Tìm thông tin liên quan từ kho kiến thức (Vector DB).
→ 3. Tăng cường
Thêm ngữ cảnh truy xuất được vào prompt.
→ 4. Sinh nội dung
LLM tạo câu trả lời dùng cả câu hỏi + ngữ cảnh.
→ 5. Trả lời
Câu trả lời được hiển thị cho người dùng.

2. Quy trình RAG (Chi tiết)

Câu hỏi người dùng
↓
1. Embedding (Câu hỏi)
↓
2. Tìm kiếm Vector (Vector DB)
↓
3. Xây dựng Prompt (Câu hỏi + Ngữ cảnh)
↓
4. LLM (Sinh câu trả lời)
↓
5. Câu trả lời cuối (cho người dùng)

Vector Database
→ Top-k tài liệu liên quan

3. Thành phần chính

- **Embedding Model** → Chuyển văn bản thành vector.
- **Vector Database** → Lưu trữ và tìm kiếm embedding.
- **Retriever** → Tìm các đoạn liên quan nhất.
- **Context (Chunks)** → Thông tin được truy xuất.
- **LLM** → Sinh ra câu trả lời cuối cùng.

4. Vì sao dùng RAG?

☑ Giảm ảo giác (hallucination)
☑ Dùng dữ liệu thực tế / công ty / lĩnh vực
☑ Câu trả lời cập nhật
☑ Độ chính xác tốt hơn
☑ Hoạt động mà không cần fine-tuning

5. Ví dụ

Q: JWT là gì?

Ngữ cảnh truy xuất:
JWT (JSON Web Token) là cách nhỏ gọn để truyền tải thông tin an toàn giữa các bên...

Prompt cho LLM:
Câu hỏi: JWT là gì?
Ngữ cảnh: JWT (JSON Web Token) là cách nhỏ gọn để...
Trả lời bằng ngôn ngữ đơn giản.

Câu trả lời của LLM:
JWT (JSON Web Token) là một token nhỏ gọn dùng để truyền tải thông tin an toàn giữa các bên. Nó thường được dùng để xác thực.

6. Lưu ý quan trọng

- Chia tài liệu thành các đoạn nhỏ (chunk).
- Chunking tốt = Truy xuất tốt hơn.
- Lưu embedding, không lưu văn bản thô.
- Dùng top-k kết quả (không dùng toàn bộ tài liệu).
- Chỉ thêm ngữ cảnh hữu ích vào prompt.
- Giữ kích thước prompt trong giới hạn token.

☆ Kinh nghiệm phỏng vấn
RAG giúp LLM trả lời chính xác hơn bằng dữ liệu thực tế, hạn chế bịa đặt thông tin.

⚠ Sai lầm thường gặp
Chunking kém, truy xuất dư thừa, bỏ qua giới hạn token.

📌 Mẹo hay
Luôn đánh giá chất lượng truy xuất trước khi tối ưu prompt.

⑤ AI AGENT

AI Agent là hệ thống được vận hành bởi LLM, có thể suy nghĩ, lập kế hoạch và hành động để đạt được mục tiêu.

1. AI Agent là gì?

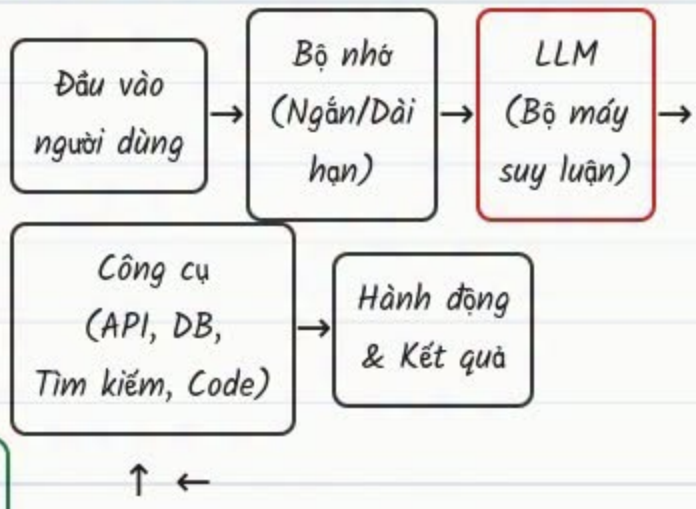
Một AI Agent có thể:

- Hiểu mục tiêu
- Chia nhỏ thành từng bước
- Sử dụng công cụ / API / dữ liệu
- Học từ kết quả
- Đưa ra đầu ra cuối cùng

☆ **Ý tưởng cốt lõi:**

Agent = LLM + Bộ nhớ + Công cụ + Lập kế hoạch

2. Kiến trúc Agent (Đơn giản)



3. Vòng lặp Agent (Cách hoạt động)

1. Nhận mục tiêu / nhiệm vụ từ người dùng.
2. Hiểu và lập kế hoạch cho bước tiếp theo.
3. Dùng công cụ hoặc truy xuất dữ liệu nếu cần.
4. Quan sát kết quả.
5. Lặp lại cho đến khi đạt mục tiêu.

4. Năng lực chính

- Hiểu mục tiêu
- Lập kế hoạch & Suy luận
- Gọi công cụ / hàm (Tool/Function Calling)
- Bộ nhớ (Ngắn hạn + Dài hạn)
- Xử lý lỗi & Tự sửa sai
- Học từ phản hồi
- Thực thi nhiều bước

💡 *Agent tốt được đánh giá bằng khả năng hoàn thành nhiệm vụ thực tế.*

5. Công cụ Agent có thể dùng

- Tìm kiếm (Web / Kho tri thức)
- API (Thời tiết, Bản đồ, Tài chính,...)
- Code Interpreter (Python, SQL,...)
- Vector DB (cho RAG)
- Email / Slack / Notion / Sheets
- Hàm tùy chỉnh (Custom Functions)

Ví dụ:

Search: Serper, Tavily

Vector DB: Pinecone, Weaviate, Qdrant

Code: Python REPL

APIs: Stripe, Twilio, OpenWeather

6. Các loại Agent

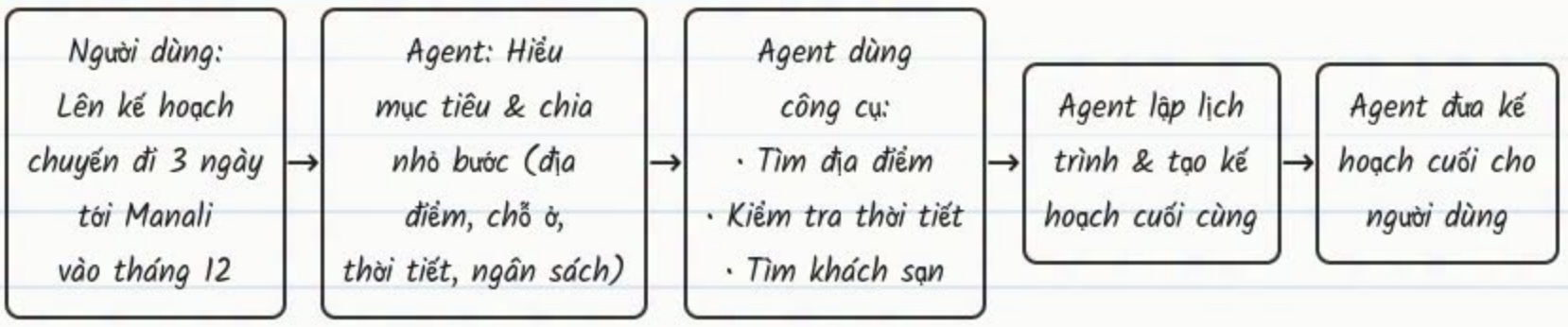
☑ Reactive Agent → Phản hồi trực tiếp đầu vào

☑ Deliberative Agent → Lập kế hoạch trước khi hành động

☑ Learning Agent → Cải thiện theo thời gian

☑ Multi-Agent System → Nhiều agent phối hợp cùng nhau

7. Ví dụ – Agent Lập kế hoạch Du lịch



☆ **Kinh nghiệm phỏng vấn**

Hiểu vòng lặp Agent và cách gọi công cụ.

⚠ **Sai lầm thường gặp**

Không xử lý vòng lặp vô hạn.

✓ **Thực hành tốt**

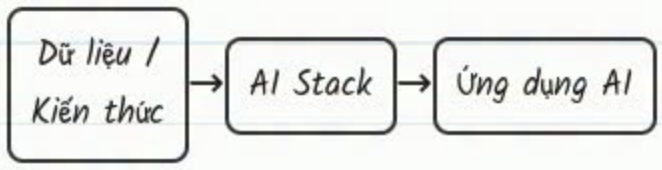
Giới hạn số bước & log từng hành động.

📌 **Ghi nhớ**

Agent mạnh nhờ công cụ tốt, không chỉ nhờ LLM giỏi.

⑥ AI Engineering Stack

Các công nghệ và công cụ cốt lõi dùng để xây dựng, triển khai và mở rộng ứng dụng AI.



1. Nhà cung cấp mô hình

- OpenAI (GPT-4o, GPT-4, GPT-3.5)
- Anthropic (Claude 3)
- Google (Gemini 1.5)
- Meta (Llama 3)
- Mistral AI
- Cohere

→ Cung cấp LLM qua API hoặc mô hình tự lưu trữ (self-hosted)

2. Framework & Thư viện

LangChain	→ Xây ứng dụng LLM dễ dàng
LangGraph	→ Xây agent AI có trạng thái
LlamaIndex	→ Kết nối LLM với dữ liệu
Hugging Face	→ Mô hình, dữ liệu, công cụ
OpenAI SDK	→ SDK chính thức của OpenAI
LiteLLM	→ API thống nhất cho 100+ LLM

3. Tầng Dữ liệu

Vector DB	→ Pinecone, Weaviate, Qdrant, Milvus
Database	→ PostgreSQL, MongoDB, MySQL
Storage	→ S3, GCS, Azure Blob
Xử lý dữ liệu	→ Pandas, NumPy, Spark
ETL / Ingestion	→ Airbyte, Prefect, Dagster

4. Hosting & Suy luận mô hình

Cloud	→ AWS (Bedrock, SageMaker), GCP (Vertex AI), Azure AI
Inference Server	→ vLLM, TGI, Ollama
Container	→ Docker, Kubernetes
Tối ưu mô hình	→ Quantization, ONNX, TensorRT

5. Công cụ & Tích hợp

API	→ Stripe, Twilio, Maps, Weather
Giao tiếp	→ Slack, Email, Discord
Năng suất	→ Notion, Google Sheets, Airtable
Tìm kiếm	→ Tavily, Serper, Brave Search
Thực thi code	→ Python REPL, Sandbox

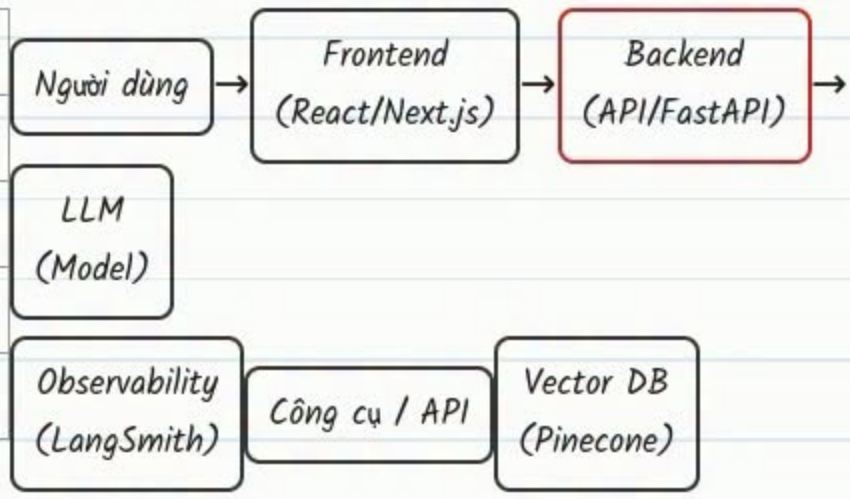
6. Observability & Đánh giá

Giám sát	→ LangSmith, Prometheus, Grafana
Logging	→ Logfire, ELK Stack
Đánh giá	→ Ragas, DeepEval, TruLens
Tracing	→ OpenTelemetry, LangSmith
Guardrails	→ Rebuff, NeMo Guardrails

7. Công cụ phát triển & Hạ tầng

Version Control	→ Git, GitHub
CI/CD	→ GitHub Actions, CircleCI
IaC	→ Terraform, Pulumi
Secrets	→ AWS Secrets Manager, Vault
Môi trường dev	→ VS Code, Cursor, Docker

8. Kiến trúc ứng dụng AI điển hình



☆ Điều cốt lõi

AI Engineering không chỉ là về mô hình.

⚠ Sai lầm thường gặp

✗ Dùng sai mô hình cho nhiệm vụ

✗ Bỏ qua chất lượng dữ liệu

💡 Mẹo hay

Bắt đầu đơn giản, đo lường kết quả, rồi lặp lại.

📌 Ghi nhớ

Ứng dụng AI tốt = Dữ liệu tốt + Stack phù hợp.

⑦ Fine-Tuning vs RAG

1. Fine-Tuning (Tinh chỉnh)

- Huấn luyện mô hình trên dữ liệu riêng của bạn.
- Mô hình học kiến thức mới.
- Phù hợp hơn cho giọng văn, phong cách, chuyên môn ngành.
- Tốn kém và mất nhiều thời gian.
- Cần GPU để huấn luyện.
- Kiến thức cố định sau khi huấn luyện.

2. RAG (Truy xuất tăng cường sinh)

- Truy xuất thông tin liên quan từ nguồn bên ngoài.
- LLM dùng thông tin đó để tạo câu trả lời.
- Kiến thức luôn cập nhật.
- Không cần huấn luyện.
- Dễ triển khai và rẻ hơn.
- Phụ thuộc vào chất lượng truy xuất.

💡 **Khi nào dùng cái nào?**

Fine-Tuning → Khi bạn cần hành vi tùy chỉnh, giọng văn hoặc chuyên môn theo lĩnh vực cụ thể.

RAG → Khi bạn cần thông tin thời gian thực, cập nhật và có tính xác thực.

3. LoRA (Low-Rank Adaptation)

- Kỹ thuật fine-tuning tiết kiệm tham số.
- Thêm các ma trận nhỏ có thể huấn luyện (adapter).
- Giảm bộ nhớ GPU và chi phí huấn luyện.
- Hoạt động rất tốt với mô hình lớn.

↓ Chỉ adapter được huấn luyện, mô hình gốc giữ nguyên (đóng băng).

4. PEFT (Fine-tuning tiết kiệm tham số)

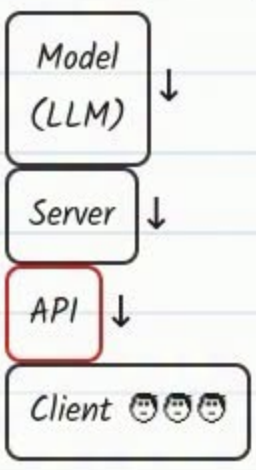
- Nhóm kỹ thuật giúp fine-tune LLM hiệu quả.
- Bao gồm LoRA, Prefix-Tuning, Prompt-Tuning, Adapter,...
- Ít tính toán, ít bộ nhớ, huấn luyện nhanh hơn.
- Lý tưởng cho môi trường production.

5. Quantization (Lượng tử hóa)

- Giảm kích thước mô hình bằng cách dùng độ chính xác thấp hơn (VD: 8-bit, 4-bit).
- Ít bộ nhớ hơn, suy luận nhanh hơn, triển khai rẻ hơn.
- Ví dụ: GPTQ, AWQ, QLoRA, GGUF.

6. Model Serving (Phục vụ mô hình)

- Triển khai mô hình để phục vụ dự đoán.
- Lựa chọn: vLLM (thông lượng cao), TensorRT-LLM (tối ưu NVIDIA), FastAPI + Uvicorn, SageMaker/Vertex AI/Azure ML, Hugging Face Inference Endpoints.



8. Kiến thức cơ bản về GPU

- GPU = Graphics Processing Unit
- Dùng cho tính toán song song.
- Cần biết: VRAM, CUDA Cores, Tensor Cores (cho ML), Batch Size ảnh hưởng bộ nhớ, Mixed Precision (FP16, BF16)

GPU	VRAM
T4	16 GB
A10G	24 GB
L4	24 GB
A100	40 / 80 GB
H100	80 GB

Nhiều VRAM hơn → mô hình lớn hơn & batch size lớn hơn

9. Chọn đúng hướng tiếp cận

Trường hợp	Cách tốt nhất	Vì sao?
Hành vi/giọng văn tùy chỉnh	Fine-Tuning/LoRA	Học đúng mẫu hình
Kiến thức cập nhật	RAG	Luôn mới
Hạn chế tài nguyên	LoRA/PEFT	Huấn luyện hiệu quả
Suy luận production	Quantization+vLLM	Nhanh & rẻ hơn

10. Điều cốt lõi

☑ Hiểu thế mạnh của từng kỹ thuật.
☑ RAG + Dữ liệu tốt = Kết quả mạnh mẽ.
☑ LoRA/PEFT = Fine-tune thông minh hơn, không cực nhọc hơn.
☑ Quantization = Triển khai mô hình hiệu quả.
☑ Đúng công cụ + Đúng cách tiếp cận = AI sẵn sàng production.

☆ **Kinh nghiệm phỏng vấn**

So sánh chi phí, tốc độ, độ chính xác giữa RAG và Fine-tuning.

⚠ **Sai lầm thường gặp**

Fine-tune khi RAG là đủ, hoặc ngược lại.

💡 **Mẹo hay**

Kết hợp cả hai khi cần thiết.

📌 **Ghi nhớ**

Không có giải pháp nào là duy nhất phù hợp mọi trường hợp.

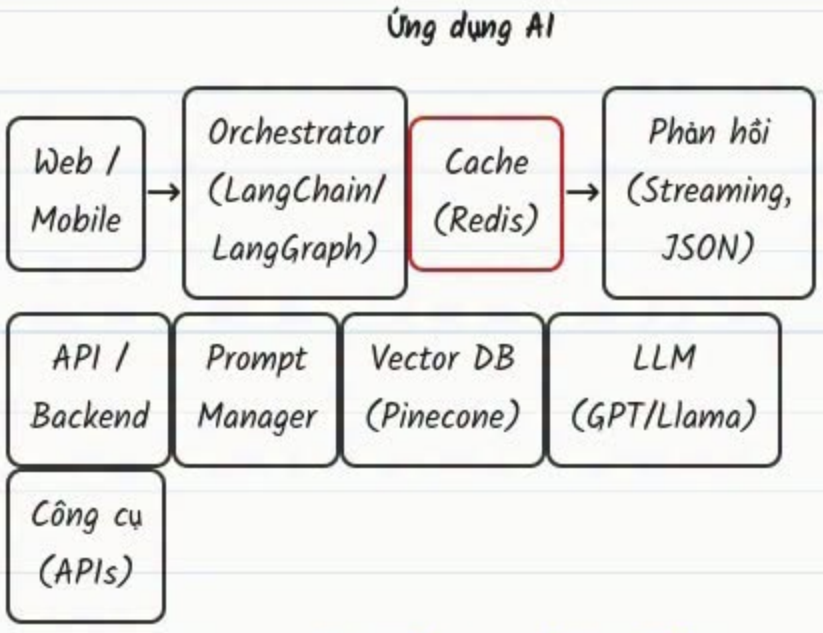
⑧ Thiết kế Hệ thống AI (Sẵn sàng Production)

Thiết kế các hệ thống AI có khả năng mở rộng, đáng tin cậy và sẵn sàng đưa vào production.

1. Mục tiêu thiết kế

- ☑ Độ trễ thấp
- ☑ Tính sẵn sàng cao
- ☑ Khả năng mở rộng
- ☑ Tối ưu chi phí
- ☑ Câu trả lời chính xác
- ☑ An toàn & Bảo mật
- ☑ Có thể quan sát (Observable)

2. Kiến trúc tổng quan



3. Thành phần chính

1. Tiền xử lý → Làm sạch, chia nhỏ, embedding
2. Vector Store → Lưu embedding
3. LLM → Sinh câu trả lời
4. Caching → Giảm chi phí & độ trễ
5. Guardrails → An toàn & giới hạn
6. Monitoring → Theo dõi & cải thiện

4. Chiến lược mở rộng

- ☑ Dùng caching (câu hỏi thường gặp)
- ☑ Xử lý bất đồng bộ cho tác vụ dài
- ☑ Định tuyến mô hình (nhỏ → lớn)
- ☑ Gộp yêu cầu (Batch requests)
- ☑ Mở rộng theo chiều ngang
- ☑ Dùng mô hình lượng tử hóa/nhỏ hơn
- ☑ CDN cho tài nguyên tĩnh

5. Tối ưu độ trễ

- → Phản hồi dạng streaming
- → Giữ ngữ cảnh nhỏ gọn
- → Dùng mô hình nhanh cho câu hỏi đơn giản
- → Tính trước embedding
- → Cache câu trả lời thường gặp
- → Thực thi công cụ song song

💡 **Mục tiêu:** Phản hồi dưới 1 giây để trải nghiệm người dùng tốt hơn.

6. Các tầng Cache

Caching có thể giảm 50-80% chi phí trong ứng dụng AI!

App Cache (In-memory) → Embedding Cache (Tái sử dụng vector) → Response Cache (Câu hỏi thường gặp)

7. Guardrails & Bảo mật

- ☑ Kiểm tra đầu vào
- ☑ Chống prompt injection
- ☑ Kiểm duyệt nội dung
- ☑ Phát hiện / che PII
- ☑ Giới hạn tốc độ (Rate limiting)
- ☑ Kiểm soát & ghi log truy cập

8. Observability

- 🔍 Theo dõi độ trễ
- 🔍 Giám sát lỗi
- 🔍 Log prompt & phản hồi
- 🔍 Đo lượng token dùng
- 🔍 A/B test kết quả mô hình
- 🔍 Dùng LangSmith, Prometheus

9. Mẫu thiết kế phổ biến

- Hệ thống dựa trên RAG (Tài liệu, FAQ)
- Hệ thống Agent AI (Tự động hóa)
- Nền tảng AI đa người thuê
- AI streaming thời gian thực
- Trợ lý AI (công cụ nội bộ)
- Phân tích dữ liệu với LLM

10. Dự án thực tế

- ☑ Chatbot tri thức công ty
- ☑ Sàng lọc hồ sơ ứng viên
- ☑ Trợ lý review code
- ☑ Chăm sóc khách hàng bằng AI
- ☑ Tóm tắt biên bản họp
- ☑ Trợ lý nghiên cứu

11. Checklist thiết kế

- ☑ Xác định rõ use case
- ☑ Đúng mô hình & cách tiếp cận
- ☑ Lên kế hoạch mở rộng
- ☑ Đã triển khai caching
- ☑ Đã thêm guardrails
- ☑ Có giám sát
- ☑ Đã tối ưu chi phí

⑨ Câu hỏi Phỏng vấn AI

☆ *Top 50 câu hỏi phỏng vấn AI Engineering*

1. Kiến thức cơ bản LLM

1. Mô hình ngôn ngữ lớn là gì?
2. LLM hoạt động như thế nào?
3. Token là gì? Chúng được tạo ra sao?
4. Context window là gì?
5. Khác biệt giữa LLM và ML truyền thống?
6. Cơ chế attention là gì?
7. Kiến trúc Transformer hoạt động ra sao?
8. Positional encoding là gì?
9. Khác biệt giữa mô hình decoder-only và encoder-decoder?
10. Temperature, top-k, top-p là gì?

2. RAG & Embedding

1. RAG là gì?
2. RAG cải thiện LLM như thế nào?
3. Embedding hoạt động ra sao?
4. Vector database là gì?
5. Bạn chia nhỏ (chunk) tài liệu như thế nào?
6. Cosine similarity là gì?
7. Quy trình truy xuất diễn ra ra sao?
8. Xử lý ảo giác (hallucination) trong RAG thế nào?
9. Đánh giá chất lượng RAG ra sao?
10. Thách thức trong RAG là gì?

3. Agent & Công cụ

1. AI Agent là gì?
2. Tool calling là gì?
3. Function calling là gì?
4. MCP là gì?
5. Vòng lặp agent hoạt động ra sao?
6. Khác biệt giữa single-agent và multi-agent?
7. Agent ghi nhớ thông tin như thế nào?
8. Làm sao ngăn vòng lặp vô hạn ở agent?
9. Đánh giá hiệu năng agent ra sao?
10. Các use case thực tế của agent là gì?

4. Fine-Tuning & Mô hình

1. Fine-tuning là gì?
2. LoRA là gì?
3. PEFT là gì?
4. Full fine-tuning vs LoRA?
5. Quantization là gì?
6. Khác biệt giữa lượng tử hóa 8-bit và 4-bit?
7. Khi nào dùng fine-tuning thay vì RAG?
8. Overfitting trong LLM là gì?
9. Đánh giá mô hình như thế nào?
10. Các chỉ số đánh giá LLM là gì?

5. Thiết kế Hệ thống

1. Thiết kế một hệ thống RAG.
2. Thiết kế một chatbot AI.
3. Thiết kế một hệ thống multi-agent.
4. Thiết kế hệ thống hỏi-đáp tài liệu.
5. Xử lý traffic lớn cho LLM API ra sao?
6. Lưu lịch sử hội thoại như thế nào?
7. Stream phản hồi ra sao?
8. Xử lý lỗi và retry thế nào?
9. Đảm bảo bảo mật prompt ra sao?
10. Giảm chi phí production như thế nào?

6. Thực hành & Khác

1. Xử lý ảo giác như thế nào?
2. Giám sát ứng dụng LLM ra sao?
3. Prompt injection là gì?
4. Bảo mật ứng dụng AI ra sao?
5. Kiểm thử đầu ra LLM thế nào?
6. Quản lý phiên bản prompt ra sao?
7. Guardrails là gì?
8. Log và debug ứng dụng LLM ra sao?
9. Các hạn chế thường gặp của LLM?
10. Tương lai của AI Engineering?

✓ Thực hành tốt trong Production

- Luôn kiểm tra & làm sạch đầu vào người dùng.
- Dùng guardrails & lọc đầu ra.
- Cache kết quả thường gặp.
- Giám sát độ trễ, lỗi & lượng token dùng.
- Dùng streaming để UX tốt hơn.
- Quản lý phiên bản prompt.
- Đánh giá mô hình liên tục.
- Có mô hình dự phòng (fallback).
- Ghi log mọi thứ. Rất hữu ích!
- Tối ưu chi phí (batching, caching, chọn mô hình).

✗ Sai lầm thường gặp

- Dùng sai mô hình cho nhiệm vụ.
- Prompt quá dài, không nén.
- Bỏ qua giới hạn context window.
- Không đánh giá đầu ra.
- Chỉ dựa vào mô hình (không có guardrails).
- Chunking kém trong RAG.
- Không cache embedding hay kết quả.
- Lộ secrets trong prompt/code.
- Không giám sát khi lên production.
- Cho rằng LLM chính xác 100%.

9. Dự án AI thực tế

- Trợ lý hỏi-đáp tài liệu - Tải tài liệu và đặt câu hỏi.
- Chatbot dựa trên RAG - Trợ lý tri thức công ty.
- Sàng lọc hồ sơ ứng viên - Trích xuất & xếp hạng CV.
- Tóm tắt biên bản họp - Chuyển lời & tóm tắt cuộc họp.

☆ Mẹo phỏng vấn

Khi trả lời:

☑ Suy nghĩ thành tiếng
☑ Giải thích sự đánh đổi (trade-off)
☑ Bắt đầu từ góc nhìn tổng quan

⑩ Lộ trình AI cho Kỹ sư

Lộ trình từng bước để đi từ Người học tò mò đến AI Engineer !

1. Nền tảng → 2. Machine Learning → 3. Deep Learning → 4. LLM & GenAI → 5. Xây & Triển khai → 6. Mở rộng & Phát triển

- Toán (Đại số tuyến tính, Xác suất)
- Python
- Cấu trúc dữ liệu
- Thống kê cơ bản

- Supervised Learning
- Unsupervised Learning
- Đánh giá mô hình
- Scikit-learn

- Mạng nơ-ron
- CNN, RNN
- Transformer
- PyTorch/TensorFlow

- Prompt Engineering
- RAG
- Fine-Tuning
- LangChain/LlamaIndex

- API & Services
- Docker
- FastAPI/Flask
- Triển khai (Cloud)

- Thiết kế hệ thống
- MLOps
- Giám sát
- Học liên tục

☆ Kiên trì + Luyện tập + Dự án = Thành thạo

1. Toán học cần nắm vững

- Đại số tuyến tính (Vector, Ma trận, Eigenvalue)
- Giải tích (Đạo hàm, Gradient)
- Xác suất (Phân phối, Định lý Bayes)
- Thống kê (Trung bình, Phương sai, Kiểm định giả thuyết)

2. Thư viện Python cốt lõi

Nhóm	Thư viện
Số học	NumPy
Xử lý dữ liệu	Pandas
Trực quan hóa	Matplotlib, Seaborn, Plotly
Machine Learning	Scikit-learn
Deep Learning	PyTorch/TensorFlow
NLP/LLM	Transformers, LangChain, LlamaIndex

3. Bộ kỹ năng AI Engineer

- ☑ Kỹ năng lập trình vững
- ☑ Khái niệm ML/DL
- ☑ Xử lý & phân tích dữ liệu
- ☑ Huấn luyện & đánh giá mô hình
- ☑ Kiến thức LLM & GenAI
- ☑ API & Triển khai
- ☑ Tư duy hệ thống & giải quyết vấn đề
- ☑ Giao tiếp & viết tài liệu

4. Ý tưởng dự án (Xây. Triển khai. Học.)

1. Trợ lý AI cá nhân dùng LLM + RAG
2. Chatbot hỏi-đáp PDF/Tài liệu
3. Sàng lọc hồ sơ ứng viên bằng AI
4. Tóm tắt biên bản họp
5. LangChain Agent tìm kiếm web
6. Hệ thống Multi-Agent phân tích cổ phiếu

5. Lộ trình học được đề xuất

📖 Học	Bắt đầu từ cơ bản. Xây nền tảng vững chắc.
💻 Luyện tập	Giải bài tập, làm Kaggle, xây dự án nhỏ.
🚀 Xây dựng	Xây dự án end-to-end. Đừng chỉ làm theo.
☁ Triển khai	Đưa mô hình ra thực tế.
📈 Phát triển	Học chủ đề nâng cao. Không ngừng cải thiện.

6. Tài nguyên hữu ích

- **Khóa học** → Coursera, fast.ai, DeepLearning.AI
- **Tài liệu** → PyTorch, TensorFlow, Hugging Face
- **Luyện tập** → LeetCode, Kaggle, Papers with Code
- **Cộng đồng** → Reddit (r/MachineLearning), Discord, X
- **Bản tin** → The Batch, Import AI, Latent Space
- **Sách** → Hands-On ML, Deep Learning with Python

✓ Thực hành tốt
- Bắt đầu nhỏ, nghĩ lớn.
- Hiểu trước khi triển khai.
- Viết code sạch & module hóa.

✗ Sai lầm thường gặp
- Học mà không xây dựng.
- Copy-paste mà không hiểu.
- Bỏ qua chất lượng dữ liệu.

💡 Mẹo hay
- Đọc bài nghiên cứu.
- Xây dựng công khai.
- Đóng góp mã nguồn mở.

☆ Ghi nhớ

"AI không phải để thay thế Kỹ sư, mà là để trao quyền cho Kỹ sư."