

Fine-Tuning trong hệ thống RAG

Fine-tuning đóng vai trò rất quan trọng trong việc **nâng cao chất lượng** của hệ thống **RAG (Retrieval-Augmented Generation)**. Dưới đây là cách **ứng dụng Fine-Tuning** cho từng phần trong hệ thống RAG, kèm theo ví dụ và hướng dẫn triển khai.

Tổng quan về hệ thống RAG

Một hệ thống RAG gồm 2 thành phần chính:

1. **Retriever**: Truy xuất các đoạn văn bản liên quan từ kho dữ liệu.
2. **Generator (LLM)**: Sinh câu trả lời dựa trên văn bản đã truy xuất.

Fine-tuning có thể được áp dụng cho **retriever**, **generator**, hoặc cả hai để cải thiện độ chính xác và tính hữu ích của câu trả lời.

1. Fine-Tuning Retriever

Mục tiêu:

- Giúp retriever tìm đúng các đoạn văn bản **liên quan hơn** đến câu hỏi của người dùng.

Cách làm:

- Sử dụng các cặp dữ liệu `query <-> relevant passage`.
- Huấn luyện mô hình bi-encoder (ví dụ: `sentence-transformers`) để embedding câu hỏi và tài liệu gần nhau trong không gian vector.

Công cụ:

- `sentence-transformers` (Hugging Face)
- Datasets: Haystack, BEIR, custom Q&A pairs

Ví dụ:

```
from sentence_transformers import SentenceTransformer, InputExample, losses
from torch.utils.data import DataLoader
```

```

train_examples = [
    InputExample(texts=["What is the capital of France?", "Paris is the capital of France."]),
    ...
]
model = SentenceTransformer('sentence-transformers/all-MiniLM-L6-v2')
train_dataloader = DataLoader(train_examples, batch_size=16)
train_loss = losses.MultipleNegativesRankingLoss(model)

model.fit(train_objectives=[(train_dataloader, train_loss)], epochs=1)

```

2. Fine-Tuning Generator (LLM)

Mục tiêu:

- Giúp LLM sinh ra câu trả lời chính xác hơn **dựa trên context truy xuất được**.

Cách làm:

- Sử dụng tập dữ liệu gồm: `context`, `question`, và `answer`.
- Fine-tune LLM như `T5`, `GPT-2`, `Mistral`, `LLaMA` trên các cặp `input: context + question → output: answer`.

Công cụ:

- Hugging Face Transformers
- PEFT / LoRA / QLoRA để fine-tune mô hình lớn nhẹ hơn

Ví dụ:

```

from transformers import T5Tokenizer, T5ForConditionalGeneration, Trainer, TrainingArguments

tokenizer = T5Tokenizer.from_pretrained("t5-small")
model = T5ForConditionalGeneration.from_pretrained("t5-small")

def preprocess(example):
    input_text = f"question: {example['question']} context: {example['context']}"
    target_text = example["answer"]
    return tokenizer(input_text, truncation=True, padding="max_length", max_length=512), \
        tokenizer(target_text, truncation=True, padding="max_length", max_length=64)

# Sử dụng Trainer API để fine-tune

```

3. Fine-Tuning kết hợp Retriever + Generator

- Một số framework (như [Haystack](#), [RAG](#)) hỗ trợ fine-tune **toàn bộ pipeline** end-to-end.
- Tuy nhiên, việc này **đắt đỏ** và phức tạp hơn → nên ưu tiên fine-tune từng phần riêng trước.

Khi nào nên Fine-Tune trong RAG?

Dấu hiệu	Giải pháp
Truy xuất sai tài liệu	Fine-tune retriever
Sinh câu trả lời sai / không khớp	Fine-tune generator
Cần tối ưu cho domain cụ thể	Fine-tune cả hai

Tools hỗ trợ Fine-tuning RAG

- [LangChain](#) + [OpenAI](#) / [HuggingFace](#)
- [Haystack](#)
- [Hugging Face Transformers](#) + [PEFT](#)
- [TruLens](#) hoặc [Promptfoo](#) để đánh giá chất lượng sau fine-tuning

Thành phần	Fine-tuning giúp gì?
Retriever	Tìm đúng đoạn tài liệu liên quan
Generator	Sinh câu trả lời chính xác hơn từ context
Toàn hệ thống	Tối ưu hóa end-to-end, tăng chất lượng RAG

Tóm tắt:

Thành phần	Fine-tuning giúp gì?
Retriever	Tìm đúng đoạn tài liệu liên quan
Generator	Sinh câu trả lời chính xác hơn từ context
Toàn hệ thống	Tối ưu hóa end-to-end, tăng chất lượng RAG