Các công cụ

- LangSmith, Promptfoo, và TruLens
- Hugging Face Transformers, PEFT, LoRA, và QLoRA
- Thực hành LangSmith
- Bài thực hành Promptfoo cơ bản
- Bài Thực Hành: Đánh Giá Hệ Thống RAG với TruLens và LangChain
- Bài thực hành PEFT: Phân loại phản hồi khách hàng (Feedback)

LangSmith, Promptfoo, và TruLens

1. **LangSmith** – Giám sát và kiểm thử pipelines trong LangChain

LangSmith là một nền tảng **được phát triển bởi LangChain** giúp bạn:

- Ghi lại và giám sát các pipeline tương tác với LLM.
- Kiểm tra và đánh giá chất lượng các lời gọi đến LLM.
- Phát hiện lỗi, theo dõi hiệu suất và so sánh prompt/agent chains.

Tính năng chính:

- **Trace** toàn bộ luồng hoạt động trong LangChain (gồm các agent, tool, retriever…).
- **Compare** giữa các phiên bản prompt hoặc mô hình.
- **Test Suites**: tạo và chạy bộ test trên các prompt.
- **Feedback System**: thêm đánh giá thủ công hoặc tự động.

Dùng khi:

- Bạn đang dùng LangChain để xây dựng app dùng LLM.
- Muốn kiểm tra, debug hoặc theo dõi các phiên bản của mô hình/prompt.

2. **Promptfoo** – Kiểm thử và benchmark các prompt

Promptfoo là một **công cụ dòng lệnh và dashboard** giúp bạn **kiểm thử (test), so sánh (benchmark)** và đánh giá hiệu suất của **prompt**.

Tính năng chính:

- Viết **test cases** giống như unit tests cho prompt.
- So sánh nhiều mô hình (GPT-4, Claude, Mistral…) với cùng một prompt.
- Đo hiệu suất (latency, độ dài, token usage, v.v).
- Hỗ trợ tích hợp CI/CD – kiểm thử prompt tự động mỗi lần đẩy mã.

Ví dụ:

Bạn có thể viết một test YAML:

```
prompts:
  - "Summarize: {{input}}"
tests:
  - input: "This is a very long article about..."
    expected_output: "A short summary"
```

```
promptfoo test
```

Dùng khi:

- Muốn so sánh đầu ra từ nhiều mô hình hoặc nhiều phiên bản prompt.
- Muốn đảm bảo chất lượng prompt trước khi đưa vào production.

3. TruLens – Giám sát và đánh giá đạo đức, độ tin cậy, tính đúng đắn của LLM

TruLens là một framework mã nguồn mở giúp bạn:

- **Đánh giá chất lượng đầu ra LLM** (như factuality, relevance, toxicity…).
- **Tích hợp feedback tự động** (qua các đánh giá rule-based hoặc LLM-based).
- Ghi lại lịch sử lời gọi API và visual hóa qua dashboard.

Tính năng chính:

- **Instrumenting**: thêm ghi chú (instrumentation) vào app sử dụng LLM (OpenAI, LangChain...).
- **Evaluation**: cung cấp thước đo sẵn như:
  - Groundedness (tính gắn với dữ liệu truy xuất)
  - Harmfulness
  - Answer relevance
- **TruLens App**: Giao diện trực quan để duyệt và phân tích.

Dùng khi:

- Muốn **theo dõi độ đúng đắn** và **đạo đức** của LLM app.
- Cần đo lường LLM có sinh ra phản hồi sai, lệch, gây hiểu nhầm không.

So sánh nhanh:

Công cụ	Mục tiêu chính	Điểm mạnh	Khi nào dùng?

LangSmith	Giám sát & kiểm thử pipeline LLM (LangChain)	Giao diện mạnh, có trace	Khi dùng LangChain
Promptfoo	Benchmark & test prompt	CLI, CI/CD, so sánh nhiều mô hình	Khi muốn kiểm thử prompt
TruLens	Đánh giá đầu ra LLM (relevance, safety)	Tích hợp đánh giá đạo đức, factual	Khi cần đo lường chất lượng LLM

Hugging Face Transformers, PEFT, LoRA, và QLoRA

Hugging Face Transformers

Hugging Face Transformers là một thư viện mã nguồn mở nổi tiếng cung cấp các mô hình ngôn ngữ hiện đại (LLMs) đã được huấn luyện sẵn như BERT, GPT, T5, RoBERTa, BLOOM, v.v.
Thư viện này hỗ trợ nhiều tác vụ NLP như: phân loại văn bản, sinh văn bản, dịch, hỏi đáp, v.v.

Ưu điểm:

- Dễ sử dụng với API thống nhất.
- Có kho mô hình (model hub) phong phú trên https://huggingface.co/models.
- Tương thích với PyTorch, TensorFlow, JAX.

PEFT (Parameter-Efficient Fine-Tuning)

PEFT là viết tắt của *Parameter-Efficient Fine-Tuning*, tức là kỹ thuật **tinh chỉnh mô hình một cách tiết kiệm tham số**.
Thay vì tinh chỉnh **toàn bộ** mô hình (gồm hàng trăm triệu đến hàng tỷ tham số), PEFT chỉ cập nhật **một phần nhỏ**, giúp:

- Tiết kiệm bộ nhớ.
- Nhanh hơn khi huấn luyện.
- Dễ dàng áp dụng với các mô hình lớn.

PEFT phổ biến trong các trường hợp bạn muốn cá nhân hóa mô hình hoặc áp dụng mô hình vào một domain cụ thể mà không cần tốn quá nhiều tài nguyên.

LoRA (Low-Rank Adaptation)

LoRA là một kỹ thuật cụ thể trong PEFT, được dùng để **thêm các ma trận học nhỏ (low-rank matrices)** vào một số lớp của mô hình.
Thay vì cập nhật toàn bộ ma trận trọng số, LoRA chỉ học một phần nhỏ thay thế.

Cách hoạt động:

- Chèn thêm hai ma trận A và B nhỏ (low-rank) vào trong quá trình huấn luyện.
- Trọng số gốc không thay đổi, chỉ có A và B được học.

- Khi suy luận (inference), các ma trận này sẽ kết hợp lại để mô phỏng hành vi đã tinh chỉnh.

Ưu điểm:

- Ít tham số cần huấn luyện (ví dụ chỉ 1-2% so với full fine-tuning).
- Có thể chia sẻ hoặc swap các phần LoRA như plugin.

QLoRA (Quantized LoRA)

QLoRA là sự kết hợp giữa:

- **LoRA** (để tinh chỉnh một phần nhỏ).
- **Quantization (lượng tử hóa)** – giảm số bit dùng để lưu trọng số mô hình (ví dụ từ float32 xuống int4).

QLoRA cho phép bạn:

- Tinh chỉnh các mô hình **rất lớn** (7B, 13B, 65B tham số) **trên GPU thương mại như 1 x 24GB VRAM**.
- Sử dụng ít RAM, ít GPU.
- Đạt hiệu suất gần như tinh chỉnh đầy đủ.

QLoRA đã được dùng trong nhiều mô hình hiệu suất cao như Guanaco, RedPajama, v.v.

Tổng kết:

Thuật ngữ	Ý nghĩa	Lợi ích chính
Transformers	Thư viện mô hình NLP mạnh mẽ	Dễ sử dụng, nhiều mô hình sẵn
PEFT	Tinh chỉnh tiết kiệm tham số	Nhanh, tiết kiệm tài nguyên
LoRA	Cách tinh chỉnh trong PEFT	Chỉ học ma trận nhỏ, hiệu quả
QLoRA	LoRA + lượng tử hóa mô hình	Tinh chỉnh mô hình lớn trên máy nhỏ

Thực hành LangSmith

LangSmith là gì?

LangSmith là một nền tảng giúp bạn:LangSmith

- **Theo dõi (tracing)**: Ghi lại toàn bộ quá trình thực thi của ứng dụng LLM.
- **Đánh giá (evaluation)**: Đánh giá chất lượng đầu ra của mô hình.
- **Kỹ thuật prompt (prompt engineering)**: Quản lý và tối ưu hóa các prompt.
- **Giám sát (observability)**: Theo dõi hiệu suất và hành vi của ứng dụng.LangSmith

LangSmith có thể được sử dụng độc lập hoặc tích hợp với **LangChain** để xây dựng và triển khai các ứng dụng LLM chất lượng cao.

Bước 1: Cài đặt và cấu hình

1.1 Cài đặt thư viện cần thiết

```
pip install langsmith openai
```

1.2 Tạo tài khoản và API Key

1. Truy cập LangSmith và đăng ký tài khoản.
2. Sau khi đăng nhập, vào phần **Settings** và tạo một **API Key**.Introduction | □□□□ LangChain +5LangSmith+5YouTube+5LangSmith+1Medium+1

1.3 Thiết lập biến môi trường

```
export LANGSMITH_TRACING=true
export LANGSMITH_API_KEY="your-langsmith-api-key"
export OPENAI_API_KEY="your-openai-api-key"
```

Bước 2: Tạo ứng dụng mẫu và ghi lại quá trình thực thi

2.1 Định nghĩa ứng dụng mẫu

```
from langsmith import traceable
from langsmith.wrappers import wrap_openai
from openai import OpenAI

# Bọc OpenAI client để ghi lại các cuộc gọi
openai_client = wrap_openai(OpenAI())

# Hàm truy xuất tài liệu (giả lập)
def retriever(query: str):
    return ["Harrison worked at Kensho"]

# Hàm chính thực hiện RAG
@traceable
def rag(question: str):
    docs = retriever(question)
    system_message = f"Answer the user's question using only the provided information below:\n\n{docs[0]}"
    response = openai_client.chat.completions.create(
        messages=[
            {"role": "system", "content": system_message},
            {"role": "user", "content": question},
        ],
        model="gpt-4o-mini",
    )
    return response.choices[0].message.content

# Gọi hàm và in kết quả
answer = rag("Where did Harrison work?")
print("Answer:", answer)
```

2.2 Xem trace trên LangSmith

Sau khi chạy mã, truy cập LangSmith để xem chi tiết trace của ứng dụng.LangSmith+3LangSmith+3YouTube+3

Bước 3: Đánh giá ứng dụng với LangSmith

3.1 Tạo tập dữ liệu đánh giá

```
from langsmith import Client
```

```
client = Client()
dataset = client.create_dataset(
    name="agent-qa-demo",
    description="Dataset for evaluating the RAG application."
)

# Thêm dữ liệu vào tập
examples = [
    {"input": {"question": "Where did Harrison work?"}, "output": "Harrison worked at Kensho."},
    {"input": {"question": "What company did Harrison work for?"}, "output": "Kensho"},
]

for example in examples:
    client.create_example(
        inputs=example["input"],
        outputs={"answer": example["output"]},
        dataset_id=dataset.id
    )
```

3.2 Chạy đánh giá trên tập dữ liệu

```
from langsmith.evaluation import run_on_dataset
import functools

# Định nghĩa hàm tạo ứng dụng
def create_rag_app():
    return functools.partial(rag)

# Chạy đánh giá
results = run_on_dataset(
    dataset_name="agent-qa-demo",
    llm_or_chain_factory=create_rag_app(),
    evaluation="qa",
    client=client,
    project_name="rag-evaluation-demo"
)
```

3.3 Phân tích kết quả

Sau khi đánh giá hoàn tất, bạn có thể xem kết quả chi tiết trên giao diện LangSmith, bao gồm:

- Các trace của từng lần chạy.
- Điểm số đánh giá.
- Phản hồi từ người dùng (nếu có).Introduction | □□□□ LangChainLearn R, Python & Data Science Online+1YouTube+1

Bước 4: Tùy chỉnh và mở rộng

- **Tích hợp với LangChain**: Nếu bạn sử dụng LangChain, LangSmith có thể tích hợp trực tiếp để ghi lại trace của các chain và agent.
- **Tạo dashboard tùy chỉnh**: LangSmith cho phép bạn tạo các dashboard để giám sát các chỉ số quan trọng như chi phí, độ trễ và chất lượng phản hồi.
- **Thu thập phản hồi từ người dùng**: Bạn có thể thu thập phản hồi từ người dùng cuối để cải thiện ứng dụng.

Bài thực hành Promptfoo cơ bản

Dưới đây là **bài thực hành Promptfoo cơ bản** để giúp bạn bắt đầu đánh giá và so sánh các prompt sử dụng mô hình ngôn ngữ (LLM). `promptfoo` rất hữu ích trong việc tối ưu prompt, benchmark nhiều model khác nhau, và kiểm thử các thay đổi trong ứng dụng sử dụng LLM.

MỤC TIÊU

- Sử dụng Promptfoo để benchmark các prompt.
- So sánh output giữa nhiều mô hình như `gpt-3.5-turbo` và `openrouter/mistral`.
- Tùy chỉnh tiêu chí đánh giá output.
- Chạy test bằng CLI và hiển thị kết quả.

BƯỚC 1: CÀI ĐẶT PROMPTFOO

```
npm install -g promptfoo
```

BƯỚC 2: TẠO CẤU TRÚC THƯ MỤC DỰ ÁN

```
mkdir promptfoo-demo && cd promptfoo-demo
touch config.yaml
mkdir evals
```

BƯỚC 3: TẠO TẬP TIN CẤU HÌNH `config.yaml`

```
prompts:
  - "Translate to French: {{input}}"
  - "Please provide the French translation for: {{input}}"
```

```
providers:
  - id: openai:gpt-3.5-turbo
    config:
      apiKey: $OPENAI_API_KEY
  - id: openrouter:mistralai/mistral-7b-instruct
    config:
      apiKey: $OPENROUTER_API_KEY

tests:
  - vars:
      input: "Hello, how are you?"
    assert:
      includes:
        - "Bonjour"

  - vars:
      input: "I am going to the market."
    assert:
      includes:
        - "marché"
```

Bạn cần lấy API Key từ OpenAI và OpenRouter.

BƯỚC 4: CHẠY ĐÁNH GIÁ

Chạy benchmark từ CLI:

```
promptfoo eval config.yaml
```

Sau đó, hiển thị dashboard kết quả:

```
promptfoo web
```

Mở trình duyệt tại địa chỉ: http://localhost:3000 để xem kết quả trực quan.

Bài Thực Hành: Đánh Giá Hệ Thống RAG với TruLens và LangChain

Mục Tiêu

- Xây dựng một hệ thống RAG đơn giản sử dụng LangChain.
- Tích hợp TruLens để theo dõi và đánh giá hiệu suất của hệ thống.
- Áp dụng các hàm phản hồi (feedback functions) để đo lường các chỉ số như độ liên quan, tính chính xác và sự phù hợp của ngữ cảnh.docs.pinecone.io+1TruEra+1

Bước 1: Cài Đặt Môi Trường

Cài đặt các thư viện cần thiết:

```
pip install trulens trulens-apps-langchain trulens-providers-openai openai langchain langchainhub langchain-openai langchain_community faiss-cpu bs4 tiktoken
```

Thiết lập khóa API cho OpenAI:

```
import os
os.environ["OPENAI_API_KEY"] = "sk-..."  # Thay thế bằng khóa API của bạn
```

Bước 2: Tạo Hệ Thống RAG Đơn Giản

Tạo một ứng dụng RAG đơn giản sử dụng LangChain:

```
from langchain.chains import RetrievalQA
from langchain.vectorstores import FAISS
from langchain.embeddings import OpenAIEmbeddings
from langchain.chat_models import ChatOpenAI
from langchain.document_loaders import TextLoader

# Tải dữ liệu
loader = TextLoader("data.txt")
```

```
documents = loader.load()

# Tạo vector store
embeddings = OpenAIEmbeddings()
vectorstore = FAISS.from_documents(documents, embeddings)

# Tạo hệ thống RAG
qa_chain = RetrievalQA.from_chain_type(
    llm=ChatOpenAI(temperature=0),
    retriever=vectorstore.as_retriever()
)
```

Bước 3: Tích Hợp TruLens

Sử dụng TruLens để theo dõi và đánh giá hệ thống:YouTube+5TruLens+5YouTube+5

```
from trulens_eval import Tru
from trulens_eval.feedback import Feedback
from trulens_eval.feedback.provider.openai import OpenAI as OpenAIFeedbackProvider
from trulens_eval.apps.langchain import instrument_langchain

# Khởi tạo Tru
tru = Tru()

# Thiết lập các hàm phản hồi
openai_provider = OpenAIFeedbackProvider()
f_qa_relevance = Feedback(openai_provider.relevance).on_input_output()
f_context_relevance = Feedback(openai_provider.relevance).on_input().on_retrieved_context()
f_groundedness = Feedback(openai_provider.groundedness).on_input().on_retrieved_context().on_output()

# Tích hợp TruLens vào hệ thống
qa_chain_recorder = instrument_langchain(
    qa_chain,
    app_id="langchain_app",
    feedbacks=[f_qa_relevance, f_context_relevance, f_groundedness]
)
```

Bước 4: Gửi Truy Vấn và Đánh Giá

Gửi truy vấn và đánh giá phản hồi:

```
with qa_chain_recorder as recorder:
    response = qa_chain_recorder.query("What is the capital of France?")
    print(response)
```

Bước 5: Khám Phá Kết Quả trong Dashboard

Khởi chạy dashboard của TruLens để xem kết quả đánh giá:docs.pinecone.io

```
tru.run_dashboard()
```

Dashboard sẽ hiển thị các chỉ số như:

- **QA Relevance**: Độ liên quan giữa câu hỏi và câu trả lời.
- **Context Relevance**: Độ liên quan giữa truy vấn và ngữ cảnh được truy xuất.
- **Groundedness**: Mức độ mà câu trả lời dựa trên ngữ cảnh được cung cấp.docs.pinecone.io Lab Lab+2Analytics Vidhya+2Zilliz+2

Bước 6: Tối Ưu Hệ Thống

Dựa trên các phản hồi và chỉ số từ TruLens, bạn có thể:

- Điều chỉnh kích thước chunk hoặc chiến lược chunking.
- Thay đổi mô hình embedding hoặc vector store.
- Tối ưu prompt để cải thiện độ chính xác và tính phù hợp của câu trả lời.Zilliz+1TruLens+1

Bài thực hành PEFT: Phân loại phản hồi khách hàng (Feedback)

Dưới đây là **một bài thực hành PEFT gắn liền với thực tế**, cụ thể là bài toán **phân loại phản hồi khách hàng (Feedback) của người dùng tiếng Việt thành các nhóm: Tích cực, Tiêu cực, Trung lập**. Bài này áp dụng PEFT (LoRA) để fine-tune một mô hình nhỏ và dễ triển khai thực tế trong doanh nghiệp hoặc startup.

"Phân loại phản hồi khách hàng tiếng Việt bằng PEFT + LoRA"

Mục tiêu:

- Hiểu cách áp dụng PEFT với LoRA để fine-tune mô hình pre-trained cho phân loại văn bản.
- Làm việc với dữ liệu phản hồi khách hàng (giả lập từ thực tế).
- Tối ưu hoá chi phí và thời gian train bằng cách chỉ fine-tune một phần nhỏ mô hình.

Bối cảnh thực tế

Một cửa hàng TMĐT muốn tự động **phân loại phản hồi khách hàng** sau mỗi đơn hàng để:

- Gửi báo cáo chất lượng dịch vụ theo tuần/tháng.
- Phân nhóm khiếu nại để ưu tiên xử lý.
- Tự động gắn thẻ sentiment vào feedback để dễ lọc.

Cài đặt thư viện

```
pip install transformers datasets accelerate peft bitsandbytes
```

Dataset giả lập

```
from datasets import Dataset


data = {
    "text": [
        "Giao hàng nhanh, sản phẩm đẹp",  # tích cực
```

```
        "Tôi rất thất vọng vì sản phẩm bị hỏng",  # tiêu cực
        "Bình thường, không có gì đặc biệt",  # trung lập
        "Đóng gói kỹ, tôi sẽ quay lại mua",  # tích cực
        "Hàng giao trễ, đóng gói cẩu thả",  # tiêu cực
        "Ổn, chưa biết có dùng lâu được không"  # trung lập
    ],
    "label": [2, 0, 1, 2, 0, 1]  # 0: tiêu cực, 1: trung lập, 2: tích cực
}

dataset = Dataset.from_dict(data).train_test_split(test_size=0.3)
```

Tiền xử lý

```
from underthesea import word_tokenize

def tokenize_vi(example):
    example["text"] = word_tokenize(example["text"], format="text")
    return example

dataset = dataset.map(tokenize_vi)
```

Dùng PhoBERT + LoRA để fine-tune

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification
from peft import get_peft_model, LoraConfig, TaskType

model_id = "vinai/phobert-base"
tokenizer = AutoTokenizer.from_pretrained(model_id, use_fast=False)
base_model = AutoModelForSequenceClassification.from_pretrained(model_id, num_labels=3)

lora_config = LoraConfig(
    r=8,
    lora_alpha=32,
    target_modules=["encoder.layer.11.output.dense"],
    lora_dropout=0.1,
    bias="none",
    task_type=TaskType.SEQ_CLS
)

peft_model = get_peft_model(base_model, lora_config)
```

```
peft_model.print_trainable_parameters()
```

Huấn luyện mô hình

```
def encode(example):
    return tokenizer(example["text"], truncation=True, padding="max_length", max_length=128)

encoded = dataset.map(encode)

from transformers import TrainingArguments, Trainer
from sklearn.metrics import classification_report
import numpy as np

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=1)
    return {"accuracy": (preds == labels).mean()}

training_args = TrainingArguments(
    output_dir="./sentiment_model",
    per_device_train_batch_size=4,
    num_train_epochs=3,
    evaluation_strategy="epoch",
    logging_dir="./logs"
)

trainer = Trainer(
    model=peft_model,
    args=training_args,
    train_dataset=encoded["train"],
    eval_dataset=encoded["test"],
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)

trainer.train()
```

Dự đoán phản hồi mới (ví dụ thực tế)

```
import torch

label_map = {0: "Tiêu cực", 1: "Trung lập", 2: "Tích cực"}

def predict(text):
    text = word_tokenize(text, format="text")
    inputs = tokenizer(text, return_tensors="pt", truncation=True, padding="max_length", max_length=128)
    with torch.no_grad():
        logits = peft_model(**inputs).logits
    pred = torch.argmax(logits, axis=1).item()
    return label_map[pred]

# Ví dụ thực tế
feedback = "Mình không hài lòng với chất lượng vải"
print("Phân loại:", predict(feedback))
```

Ứng dụng thực tế

- Tích hợp vào backend của trang web TMĐT, nhận phản hồi → phân loại tự động.
- Lưu label sentiment vào cơ sở dữ liệu để thống kê cảm xúc khách hàng theo thời gian.
- Gửi cảnh báo nội bộ nếu số phản hồi tiêu cực tăng vọt.