Các phương pháp tiền xử lý dữ liệu

Các bước tiền xử lý dữ liệu tập trung vào việc chuẩn bị dữ liệu văn bản từ các tệp PDF để sử dụng cho việc huấn luyện mô hình ngôn ngữ:

- Trích xuất văn bản từ PDF: Hàm `extract_text_from_pdfs(folder_path)` được sử dụng để đọc tất cả các tệp PDF trong một thư mục được chỉ định và trích xuất toàn bộ nội dung văn bản từ chúng. Toàn bộ văn bản sau khi trích xuất sẽ được nối lại thành một chuỗi lớn.

- Phân đoạn văn bản (Chunking): Hàm `chunk_text(text, chunk_size, chunk_overlap)` được sử dụng để chia chuỗi văn bản lớn đã trích xuất thành các đoạn nhỏ hơn (chunks). Quá trình này có tính năng chồng lấn (overlap) giữa các đoạn để đảm bảo ngữ cảnh không bị mất ở ranh giới các chunk.

  - `CHUNK_SIZE`: Kích thước của mỗi đoạn văn bản (số ký tự) được cấu hình là 5000 ký tự.
  - `CHUNK_OVERLAP`: Số ký tự chồng lấn giữa các đoạn được cấu hình là 500 ký tự.

- Tạo câu hỏi từ các đoạn văn bản: Trước khi fine-tune, sử dụng mô hình Gemini ('gemini-2.5-flash-lite') để tạo các câu hỏi trắc nghiệm dựa trên các đoạn văn bản (chunks) đã được tiền xử lý. Mỗi câu hỏi được tạo ra theo định dạng JSON, bao gồm câu hỏi, 4 lựa chọn, đáp án đúng và giải thích (reasoning).

- Token hóa dữ liệu: Dữ liệu văn bản (có thể là các câu hỏi đã tạo hoặc dữ liệu khác sau khi tạo câu hỏi) được chuyển đổi thành các token (số nguyên) mà mô hình có thể hiểu được bằng cách sử dụng tokenizer. Hàm `tokenize_function` áp dụng tokenizer cho toàn bộ dataset.

Phương pháp finetune

Giải pháp này sử dụng phương pháp Fine-tuning hiệu quả tham số (Parameter-Efficient Fine-Tuning - PEFT), cụ thể là kỹ thuật LoRA (Low-Rank Adaptation of Large Language Models), kết hợp với lượng tử hóa 4-bit (4-bit Quantization).

- Lượng tử hóa 4-bit (4-bit Quantization):
  - Mục đích: Giảm đáng kể lượng bộ nhớ RAM (đặc biệt là VRAM của GPU) cần thiết để tải và huấn luyện các mô hình ngôn ngữ lớn (LLM). Điều này cho phép fine-tuning các mô hình lớn hơn trên phần cứng hạn chế hơn.
  - Cách thức: Các trọng số của mô hình được biểu diễn bằng 4 bit thay vì 16 bit (bfloat16) hoặc 32 bit (float32) thông thường. Khi thực hiện các phép tính, các trọng số này sẽ được giải lượng tử hóa động (on-the-fly) trở lại định dạng có độ chính xác cao hơn, giúp cân bằng giữa hiệu quả bộ nhớ và hiệu suất tính toán.
- LoRA (Low-Rank Adaptation):
  - Mục đích: Khắc phục nhược điểm của fine-tuning toàn bộ mô hình (rất tốn kém về tài nguyên và thời gian) bằng cách chỉ fine-tune một tập hợp nhỏ các tham số bổ sung, trong khi vẫn giữ nguyên các trọng số gốc của mô hình đóng băng..
  - Cách thức: LoRA thêm các ma trận xếp hạng thấp (low-rank matrices) vào các lớp Attention của mô hình. Trong quá trình fine-tuning, chỉ các ma trận xếp hạng thấp này được cập nhật, giảm đáng kể số lượng tham số có thể huấn luyện. Khi suy luận, các ma trận này được tích hợp lại vào mô hình gốc.
  - Lợi ích: Giảm số lượng tham số có thể huấn luyện, tiết kiệm bộ nhớ và thời gian tính toán. Tăng tốc độ huấn luyện. Giảm rủi ro "quên" kiến thức ban

đầu (catastrophic forgetting) của mô hình nền tảng. Cho phép lưu trữ các "adapter" nhỏ gọn thay vì lưu toàn bộ mô hình đã fine-tune, dễ dàng chuyển đổi giữa các tác vụ.

Cấu hình finetune

Cấu hình finetune có các thông số như sau:

- Mô hình nền tảng (Base Model): meta-llama/Llama-3.2-1B
- Chiến lược lượng tử hóa: Lượng tử hóa 4-bit (NF4, bfloat16 cho tính toán, double quantization).
- Cấu hình LoRA:
  - r: 32
  - `lora_alpha`: 64
  - `target_modules`:["`q_proj` ", "`k_proj`", "`v_proj`", "`o_proj`"]
  - `lora_dropout`: 0.05
  - bias: "none"
  - `task_type`: "`CAUSAL_LM`"
- Tham số huấn luyện (Training Arguments):
  - `output_dir`:đường dẫn lưu trữ mô hình và log
  - `num_train_epochs`: 3 epoch
  - `per_device_train_batch_size`: 1 (kích thước batch trên mỗi thiết bị)
  - `gradient_accumulation_step`: 4 (tổng kích thước batch hiệu quả là 1 * 4 = 4)
  - `learning_rate`: 2e-4

  - fp16: True (sử dụng floating point 16-bit cho huấn luyện để tiết kiệm bộ nhớ và tăng tốc)
  - `logging_steps`: 20 (ghi log sau mỗi 20 bước)
  - `save_strategy`: "epoch" (lưu mô hình sau mỗi epoch)
  - optim: "`paged_adamw_8bit`" (thuật toán tối ưu AdamW được tối ưu hóa bộ nhớ)

- Bộ tập hợp dữ liệu (Data Collator): DataCollatorForLanguageModeling với mlm=False (cho tác vụ Causal Language Modeling).

- Tham số có thể huấn luyện (Trainable Parameters): trainable params: 6,815,744 || all params: 1,242,630,144 || trainable%: 0.5485

Kết quả trước và sau khi finetune, đánh giá trên bộ B1

Trước khi finetune

- Tổng số câu hỏi: 150
- Số câu trả lời đúng: 27
- Độ chính xác (Baseline): 18.00

Sau khi finetune:

- Tổng số câu hỏi: 150
- Số câu trả lời đúng: 61
- Độ chính xác: 40.67

Hướng dẫn sử dụng

1. Upload toàn bộ thư mục `CEH_project` lên Google Drive
2. Mở file source.ipynb trên Google Colab
3. Thêm các khóa bí mật ở thanh công cụ bên trái:
    - `GEMINI_API_KEY`: là Gemini api key của bạn
    - `HF_TOKEN`: token Hugging Face của bạn
4. Chạy lần lượt từng cell