Fine-Tuning Llama-3.1-8B for Drug Interaction Extraction Using DrugBank Data

Mohammad Mohsen Hajari Taheri
PSU ID: 902241477
Fatemeh Rahbari
PSU ID: 979836976

Abstract

Drug interactions occur when one medication influences another's activity, potentially causing unexpected side effects, increased severity, or diminished therapeutic effects. These interactions, categorized as pharmacokinetic or pharmacodynamic, can also involve drugs interacting with foods or beverages, impacting efficacy or safety. While most interactions are minor, some pose significant risks if not managed properly. This project focuses on fine-tuning the "meta-llama/Llama-3.1-8B" large language model using the DrugBank database to extract drug entities and their inter-entity relationships from drug interaction descriptions, outputting them as triplets in the format <drug entity1, relationship, drug entity2>. The dataset was split into training (80%), validation (10%), and test (10%) sets, with instruction tuning applied to align the model with the triplet extraction task. Preprocessing involved tokenizing prompts with a maximum length of 400 tokens, ensuring compatibility with the dataset. The project evaluates the model's performance using metrics tailored to triplet extraction accuracy, providing insights into the challenges and learnings from applying large language models to biomedical text processing.

1 Problem Definition and Dataset Creation

1.1 Problem Definition

Drug interactions, where one drug affects another's activity or interacts with foods/beverages, can cause side effects, reduced efficacy, or serious health risks, necessitating accurate identification for patient safety. Manual extraction of drug entities and their relationships from complex biomedical texts is labor-intensive and error-prone. This project aims to automate the extraction of drug interaction triplets by fine-tuning the "meta-llama/Llama-3.1-8B" model on the DrugBank dataset, enabling easy extraction of drug interaction for clinical or research applications.

1.2 Dataset Creation

Instruction tuning involves formatting data into prompts containing an instruction, input context, and expected output, allowing the model to learn both the task and the format of the response.

1.2.1 Data Composition and Instruction Tuning

Each data point in the dataset was formatted into a prompt consisting of three parts:

- **Instruction:** A fixed instruction applied uniformly across all data points: "Extract drug entities and inter-entity relationships from input and output them in the form of triplets <drug entity1, relationship, drug entity2>." This instruction guides the model to perform the specific task of triplet extraction.

- **Input:** The drug interaction text extracted from the DrugBank dataset, describing interactions between drugs or between drugs and other substances (e.g., food or beverages). These texts provide the context from which the model must identify relevant drug entities and their relationships.
- **Output:** The ground truth triplets corresponding to the input, formatted as <drug entity1, relationship, drug entity2>. These triplets serve as the target output for training, validation, and testing, enabling the model to learn the correct structure and content of the extracted relationships.

1.2.2 Data Splitting

To support model training and evaluation, the dataset was divided into:

- **80% Training Set:** Used to fine-tune the LLM via instruction tuning.
- **10% Validation Set:** Used to monitor performance and adjust hyperparameters during training.
- **10% Test Set:** Reserved for final evaluation of the model's triplet extraction accuracy.

1.2.3 Data Preprocessing

To prepare the dataset for instruction tuning, prompts combining instruction, input, and output. We selected a maximum sequence length of 400 tokens based on histogram analysis, which showed most prompts were shorter than this limit. Prompts exceeding 400 tokens were truncated, and shorter ones padded, ensuring computational efficiency and uniform batch processing.

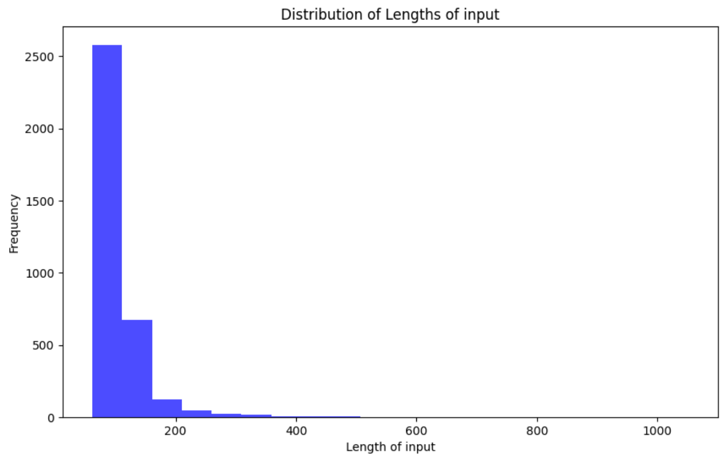

Figure 1: Histogram of length of text consisting instruction, input, and output.

1.2.4 Tokenization

The text data was processed by breaking down prompts into tokens, marking the start and end of each sequence clearly. Each prompt included an instruction, input context, and expected output, but the training specifically focused on the output part. To achieve this, **the tokens related to the instruction and input were ignored during training, ensuring the model learned only from generating the desired outputs**. This process allowed the model to efficiently learn the task of extracting drug interaction triplets.

1.2.5 Dataset Design

By combining a consistent instruction with varied input texts and ground truth triplets, the dataset enabled the model to learn both the **semantic understanding of drug interactions and the precise**

formatting of triplet outputs. This construction process ensured the dataset was well-suited for fine-tuning the LLM, supporting robust training, validation, and testing for the triplet extraction task in biomedical applications.

2 LLM Selection and Training Details

2.1 Base Model

The fine-tuning was conducted using the **meta-llama/Llama-3.1-8B** model, selected for its demonstrated capability in handling complex natural language tasks.

2.2 Fine-Tuning Method

Parameter-Efficient Fine-Tuning (PEFT) was employed via the **Low-Rank Adaptation (LoRA)** method, enabling computationally efficient adaptation of the model parameters.

2.2.1 LoRA Configuration

The LoRA setup included the following parameters:

- **Rank ($r = 8$)**: Defines the dimensionality of low-rank adaptation matrices, significantly reducing the number of trainable parameters.
- **LoRA Alpha ($\alpha = 16$)**: Controls the scale of parameter updates; larger values relative to the rank allow greater flexibility during fine-tuning.
- **Target Modules**: Adaptations were applied specifically to:
  - `q_proj` (Query projection): Generates queries for attention mechanisms.
  - `k_proj` (Key projection): Produces keys used in attention computations.
  - `v_proj` (Value projection): Computes values within attention layers.
  - `o_proj` (Output projection): Integrates attention outputs for subsequent layers.
  - `gate_proj` (Gate projection): Controls information flow in feed-forward layers.
  - `up_proj` (Up projection): Projects representations into higher-dimensional spaces.
  - `down_proj` (Down projection): Compresses representations post-expansion.
  - `lm_head` (Language model head): Converts hidden states into token predictions.
- **LoRA Dropout (5%)**: Reduces potential overfitting by randomly dropping 5% of activations during training.

2.3 Tokenization and Input Formatting

- **Maximum Token Length**: Set at 400 tokens per prompt to ensure uniform input size.
- **Label Masking**: Tokens corresponding to instruction and input portions were masked (set to ignore index), ensuring the training loss was computed only for the model-generated outputs (triplets).

2.4 Training Hyperparameters

The training was executed with these hyperparameters:

- **Batch Size**: 2 (per device)
- **Gradient Accumulation Steps**: 4 (resulting in an effective batch size of 8)
- **Learning Rate**: 2.5×10^{-5}
- **Warm-up Steps**: 5 steps to stabilize initial training phases
- **Total Training Steps**: 2000

2.5 Optimization and Precision

- **Optimizer**: Paged AdamW optimizer using 8-bit precision (`paged_adamw_8bit`)
- **Precision**: BFloat16 precision enabled for computational efficiency and reduced memory usage.

2.6 Efficiency Enhancements

- **Gradient Checkpointing**: Activated to lower GPU memory usage during training.

2.7 Logging

Training and evaluation processes were systematically logged and monitored:

- **Checkpoint Frequency**: Saved every 25 training steps
- **Evaluation Frequency**: Conducted every 50 training steps

3 Evaluation Metric and Experiments

3.1 Experiments

The `meta-llama/Llama-3.1-8B` model was fine-tuned on the DrugBank dataset for a total of 2000 training steps using Hugging Face's `transformers.Trainer` on an A100 GPU. The training configuration included:

- **Effective Batch Size:** 8 (batch size of 2 with gradient accumulation over 4 steps)
- **Learning Rate:** 2.5×10^{-5} with 5 warm-up steps
- **Optimizer:** `paged_adamw_8bit` with `bfloat16` (bf16) precision
- **Hardware:** Single GPU (`A100`)
- **LoRA Configuration:** Rank 8, applied to key modules (query, key, value, and output projections) to reduce trainable parameters while preserving performance
- **Training Logs and Evaluation:**
  - Logs recorded every 50 steps
  - Model checkpoints saved every 25 steps
  - Evaluation conducted every 50 steps
- **Experiment Tracking:** Training progress monitored using `Weights & Biases (W&B)` for real-time insights

3.1.1 Epoch:

Figure 2 confirms that training progressed as expected, reaching 2000 steps and approximately 5 epochs, aligning with the dataset size and batch configuration.

3.1.2 Learning Rate:

Figure 3 shows that learning rate decreased linearly from 2.5e-5 to 1.2e-8 over the 2000 steps, following the scheduler to ensure stable convergence.

3.2 Evaluation Metric

The decreasing training and evaluation losses suggest that the model effectively learned to extract drug interaction triplets. The stable gradient norms and linear learning rate decay further confirm that the training process was well-controlled, allowing the model to converge effectively.

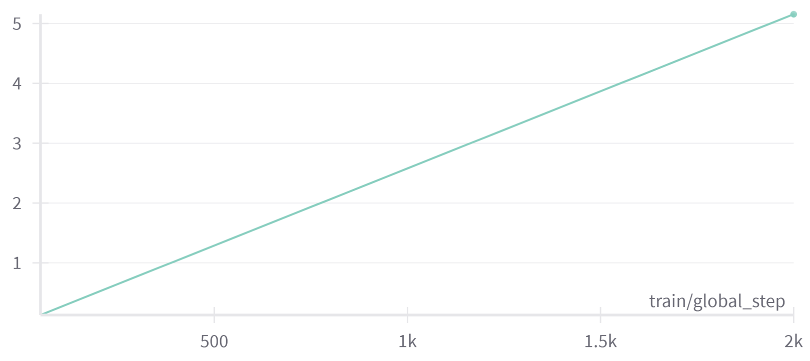

Figure 2: Number of iteration over training set.

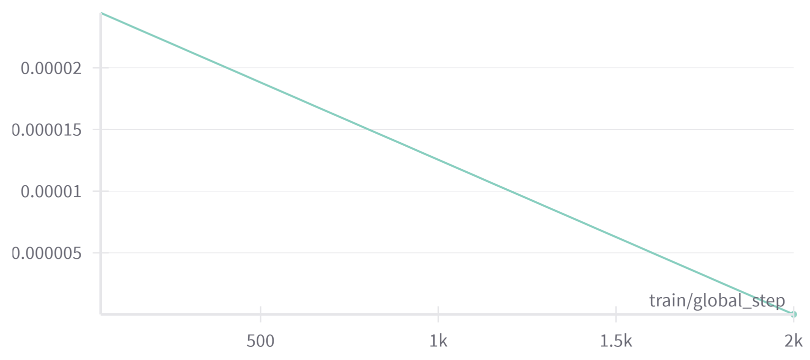

Figure 3: Learning rate changes over 2000 steps.

3.2.1 Training Loss:

Figure 4 shows a consistent decrease in training loss over 2000 steps, dropping from approximately 1.4 to below 0.2, indicating that the model successfully learned to generate the target triplets. This decline suggests effective optimization and convergence on the training data.

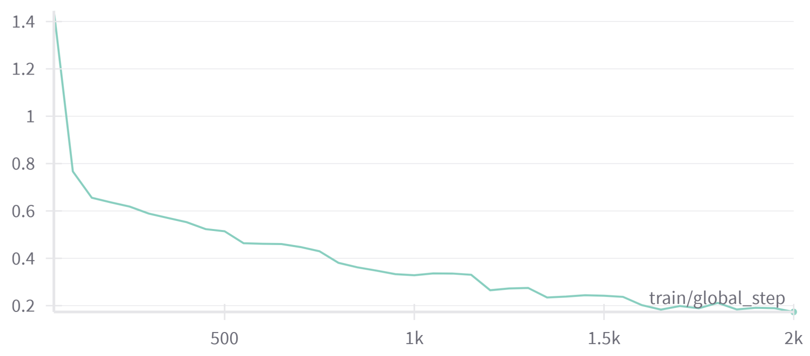

Figure 4: Training loss over 2000 steps.

3.2.2 Validation Loss:

Figure 5 shows a decreasing trend, stabilizing below 0.4, which aligns with the training loss and suggests that the model generalizes well to the validation set without significant overfitting.

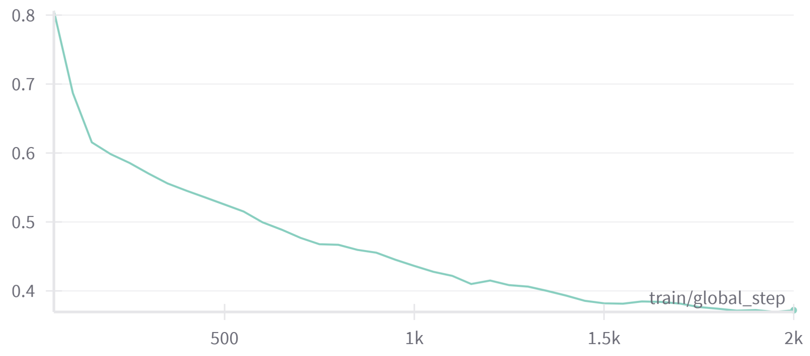

Figure 5: Validation loss over 2000 steps.

3.2.3 Test evaluation loss:

The fine-tuned model was evaluated on the test dataset, achieving a test loss of 0.3947. This low loss indicates that the model effectively learned to generate the expected output triplets from the input texts, demonstrating strong generalization performance on unseen data.

4 Challenges Encountered and Lessons Learned

4.1 Hands-on experience

Through this project, we gained valuable hands-on experience in fine-tuning large language models (LLMs) using parameter-efficient techniques, specifically Low-Rank Adaptation (LoRA).

4.2 Impact of Proper Label Masking on Model Performance

In our initial attempt, we did not mask the instruction and input parts during loss computation, which caused the model to waste capacity learning irrelevant parts of the prompt. After adjusting the dataset formatting to correctly mask the instruction and input tokens, we observed a notable improvement: both the validation loss and test evaluation loss decreased, and the model converged faster during training. This experience taught us the importance of precise label preparation for instruction tuning, ensuring that the model focuses solely on generating the correct outputs.

4.3 Low-Rank Adaptation

In this project, we learned how to apply Low-Rank Adaptation (LoRA), one of the state-of-the-art parameter-efficient fine-tuning techniques for large language models. LoRA enables efficient training by introducing trainable low-rank matrices into specific parts of the model, significantly reducing the number of trainable parameters while maintaining or even improving performance. Instead of updating all weights of the base model, LoRA modifies only small, strategically placed low-rank updates, which makes fine-tuning feasible on limited computational resources. We also learned how hyperparameters like the rank and scaling factor influence the adaptation process, and how LoRA can help balance model adaptability, training speed, and memory efficiency in real-world applications.

4.4 Determining Optimal Token Length for Efficient Training

One challenge we encountered was setting an appropriate maximum token length to avoid excessive truncation while maintaining efficient batch processing, which we addressed through data preprocessing section.

4.5 Managing Computational Challenges in Resource-Intensive Models

Through this project, we realized the critical role of quantization techniques, such as 4-bit quantization and bf16 precision, in enabling the fine-tuning of large, resource-intensive models like Llama-3.1-8B.

By significantly reducing memory requirements and computational load, these methods made it feasible to train the model on available hardware while maintaining stability and training efficiency.