

**Jenkins**

**Enhancing an Existing LLM Model with Domain-specific Jenkins knowledge**

**Provided To: Kris Stern, Harsh Pratap Singh, Shivay Lamba, Bruno Verachten, Laura Lopez Real**

**Submitted By: Nour Ziad Almulhem**

**Contacts:** [**nouralmulhem@gmail.com**](mailto:nouralmulhem@gmail.com) **| +201123309818**

**Github Handle:** [**/nouralmulhem**](https://github.com/nouralmulhem)

Contents

[**● Abstract: 3**](#_heading=h.gjdgxs)

[**● Project Description: 3**](#_heading=h.tmk9j53amhs)

[**● Proposed TimeLine: 8**](#_heading=h.1fob9te)

[**● Deliverables: 11**](#_heading=h.3znysh7)

[**● Participation: 12**](#_heading=h.2et92p0)

[**● Past Participation with Jenkins: 12**](#_heading=h.1i0kfsgrkpat)

[**● Personal Information: 12**](#_heading=h.30j0zll)

[**● About me: 12**](#_heading=h.tyjcwt)

[**● Skills Set: 13**](#_heading=h.3dy6vkm)

[**● Past projects: 13**](#_heading=h.cye3px88bale)

[**● Certificates: 15**](#_heading=h.4jns7f7og3mr)

### Abstract:

This project aims to improve the performance of an existing model, LLAMA2, by fine-tuning it specifically for Jenkins data. Raw data from Jenkins will be provided to developers for processing and understanding. The objective is to train LLAMA2 on this data to achieve better performance. A small UI will also be provided to allow users to interact with the model. MLops engineering will be involved in deploying the model and making it accessible to users.

From this we can conclude the proposed contributions into groups which might be:

* **Conduct data preprocessing and refining to extract meaningful features.**
* **Provide comprehensive documentation on the rationale behind LLAMA's selection and the enhancements we are implementing.**
* **Fine-tune the LLAMA model using our dataset, exploring optimal parameters through.**
* **Evaluate the fine-tuned model on hidden test cases to ensure robustness and accuracy.**

**Some common metrics are going to be used at this point. What we have in the following sections is 3 metrics (BLEU, Rouge, and Perplexity) which are usually used together to estimate the performance of the model.**

* **Integrate the refined model into the system and conduct thorough testing to verify its functionality.**
* **Design a simple and user-friendly graphical user interface to facilitate interactions with the system.**
* **Gather feedback to evaluate the system's performance and user experience, making necessary adjustments as needed. (By conducting both alpha and beta testing.)**
* **Using of Ollama to set up and run LLM locally**

### Project Description:

This project will be a standalone atomic unit, which will give users greater accessibility to Jenkins knowledge in a simple and user-friendly way.

Having a robust and well-maintained atomic unit adds a lot of value to this community. Contained pieces are usually simpler to maintain and facilitate collaborative development and the addition of new features

The most important part is that it increases the accessibility of data, improving the way that the Jenkins community can interact with the software. These goals provide me with strong motivation to pursue this project.

#### LLAMA2 and data flow:

The release of LLAMA-2 by Meta is a significant advance in the field of artificial intelligence. LLAMA-2 is offered under an open license to facilitate both research and commercial applications; LLAMA-2 outperforms existing benchmarks, including OpenAI models, in terms of performance and security, it has also improvements such as Grouper query attention, Ghost Attention, and In-Context Temperature rescaling; it is available in various sizes and versions including command-tuned variants such as LLaMA-Chat, and significantly outperforms other open source models.

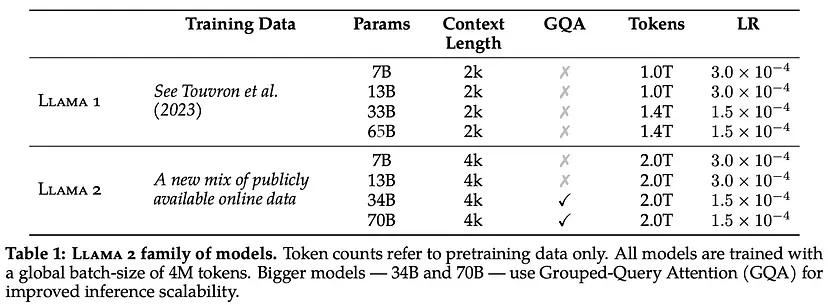


Figure 1: LLAMA 2 family of models

The training process includes supervised fine-tuning and reinforcement learning with human feedback, resulting in two reward models optimized for security and usefulness Meta also addresses bias and security concerns through in-depth data cleaning and model refinement.

This process begins with the pretraining of LLAMA 2 using publicly available online sources. Following this, we create an initial version of LLAMA 2-CHAt through the application of supervised fine-tuning. Subsequently, the model is iteratively refined using Reinforcement Learning with Human Feedback (RLHF) methodologies (this won’t be in our case).

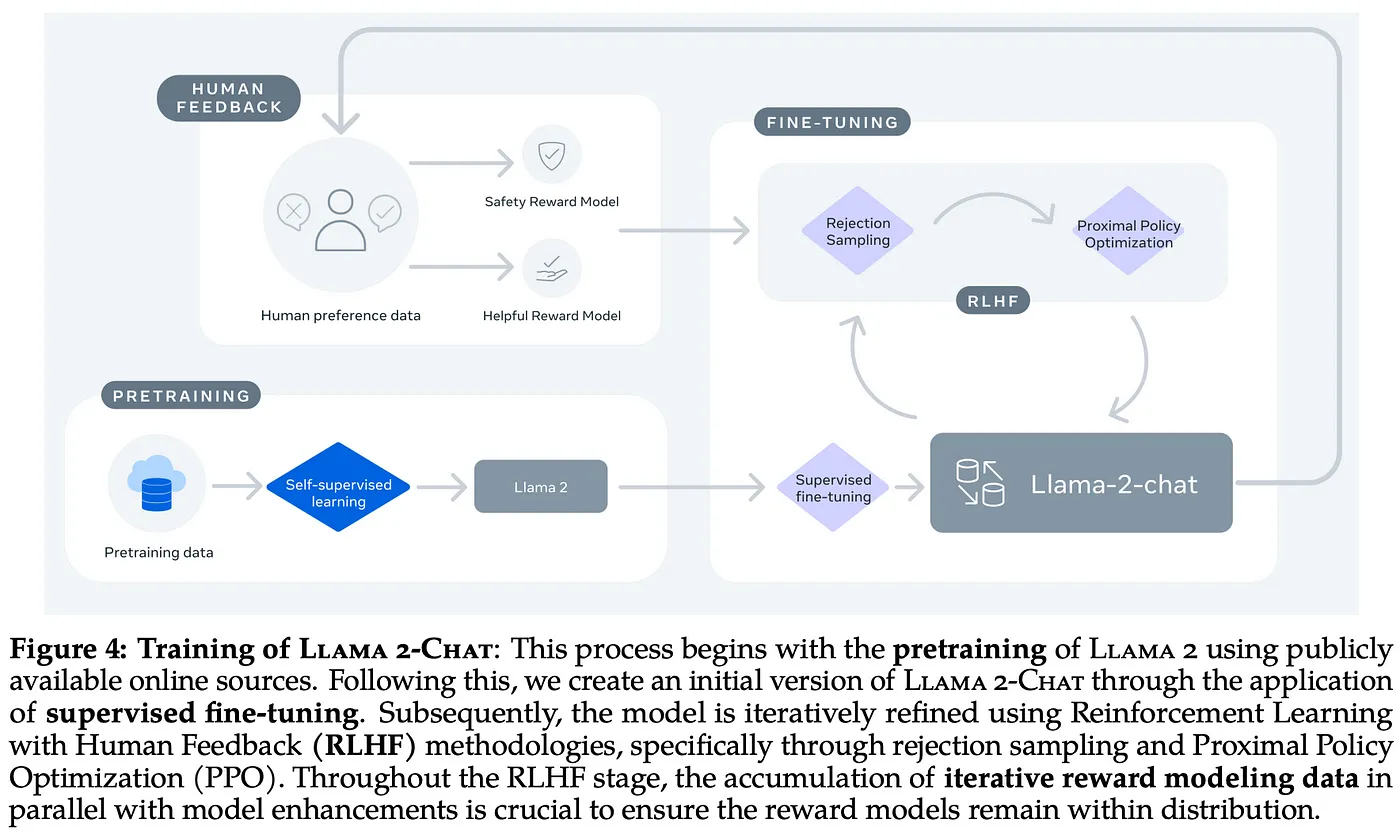


Figure 2: Training of LLAMA2-chat

#### Dataset:

some raw data is provided through:

* [https://www.jenkins.io/](https://www.jenkins.io/doc/)
* <https://www.tutorialspoint.com/jenkins/jenkins_continuous_deployment.htm>
* <https://stackoverflow.com/questions/tagged/jenkins?tab=active&page=3&pagesize=15>

**Extract Text:** we have 3 types of data

* **Jenkins knowledge through Jenkins documentations**

This data should pass through a processing phase to generate contextually relevant questions. This involved employing OpenAI engines for generating questions; in this use case it was the GPT-4’s plugin code interpreter that stood out, producing results that were more coherent, we can use other open source engines to achieve the same results

* **Other sources of knowledge like stackoverflow and stackexchange**

for many websites like [stackoverflow](https://stackoverflow.com/), [ask ubuntu](https://askubuntu.com/questions) and [Software Quality Assurance & Testing](https://sqa.stackexchange.com/) we have [data explorer](https://data.stackexchange.com/) by stack exchange which enables developers to run some queries and get the data from this websites as a csv file this data will contain the question and the accepted answer of this question also to ensure the correctness of the data collected we can specify the query to return only accepted answers with some threshold for the score it can be something like



Figure 3: stackexchange query on jenkins tagged questions

running this query will return the questions titles and body also the accepted answers and scores for all questions tagged with Jenkins and have a score more than 5 and the creation day is after 2000 Sept (can be more updated)

this returned the top 1000 questions and answers as a html elements which can be preprocessed and refined to the format we will use training llama (specified below)

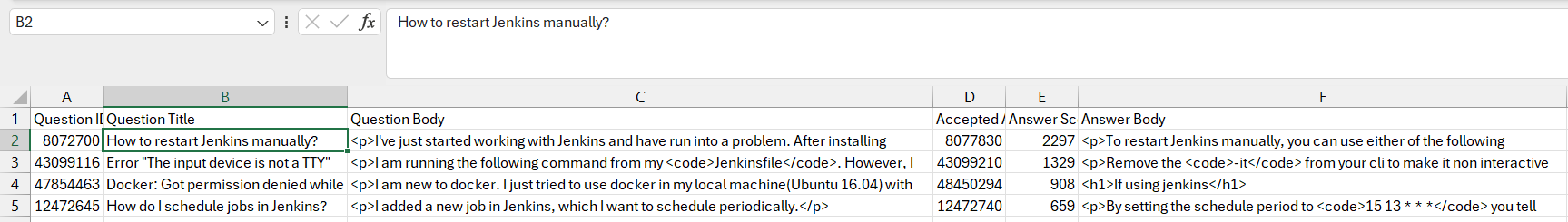


Figure 4: data set retrieved

* **Other sources like tutorialspoint**

if we are willing to get more data to train on it can be done using some tutorials like the documentation of tutorialspoint this can be done using web scraping and question generation on the documents we will get (just as specified in the first type of Jenkins documentation), in that case we will use the following libraries

**requests** - for making HTTP requests to the website

**BeautifulSoup** - for parsing the HTML code

**pandas** - for storing the scraped data in a data frame

**time** - for adding a delay between requests to avoid overwhelming the website with requests

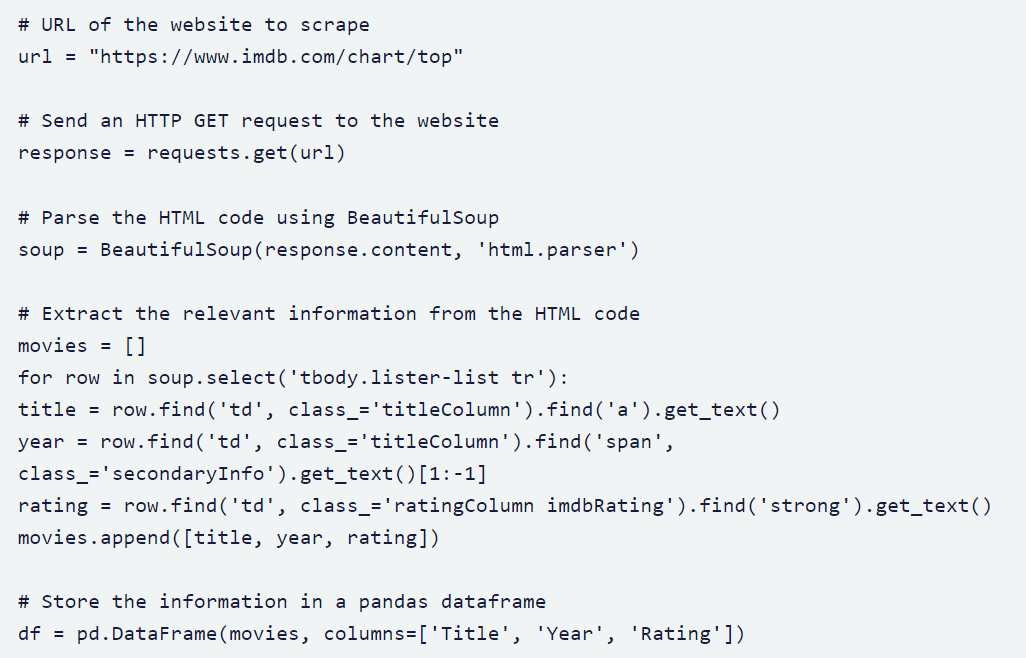


Figure 5: web scraping example

for example we will be using the h2 from each page and the div right below it to get our data from the web page

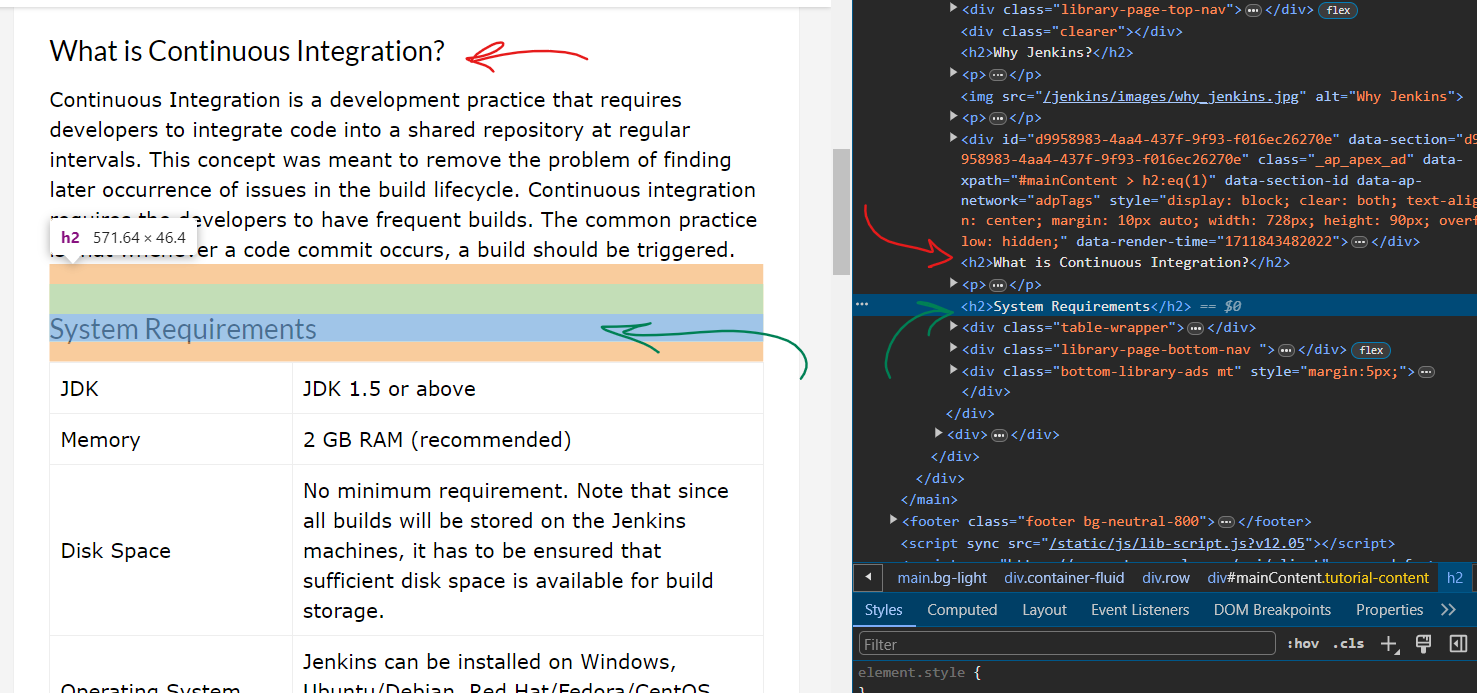


Figure 6: tutorialspoint example

**Clean, Preprocess, and Organize Data:** Clean the extracted text data to remove any unnecessary characters, formatting artifacts, or noise. Preprocessing steps might include removing special characters, normalizing text, tokenization, and removing stopwords. This will be the first step after getting accepted in GSoC

As dataset are not usually something to change and can be appended easily so it can be stored in 2 ways after being created and refined:

* A common format for storing chat data in text files is comma-separated values (CSV). we can refer to something like [this](https://huggingface.co/datasets/timdettmers/openassistant-guanaco) which can also be appended and updated easily.
* In JSON format it can be put in a database as JSON objects ready for training.

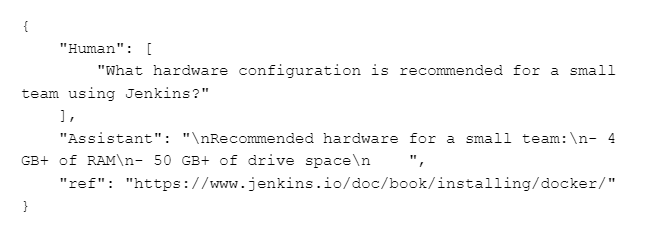


Figure 7: Dataset example

**Augment Data:** Depending on the size and diversity of the dataset, we may consider augmenting it to improve the robustness and generalization of your model.

The article entitled "Fine-Tuning Llama 2 with LoRA for Question Answering" presents a subset extracted from the timdettmers/openassistant-guanaco dataset. These samples have been processed to align with the prompt format required for compatibility with Llama 2.

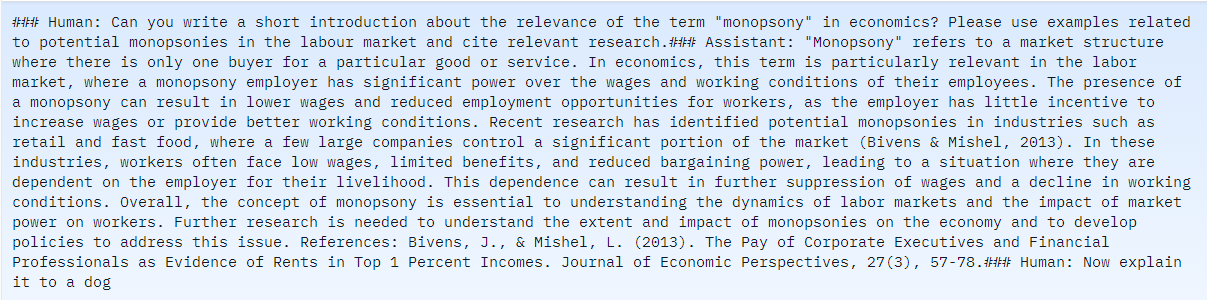


Figure 8: Reference dataset example

**This is a row in a sheet so data can basically be stored in a similar format just like any other dataset.** What I’m trying to get at the trials drive in the last section is a starting point with Jenkins data that exists in the same format as provided in the dataset in the link above. This could be achieved through Python scripts.

**Future work:** The sections wrapped within an ( \’\’\’ ) will be formatted slightly differently from the response itself; this will be provided in the chatbot response as markdown and will be parsed in the UI to give the feel of GPT responses such as

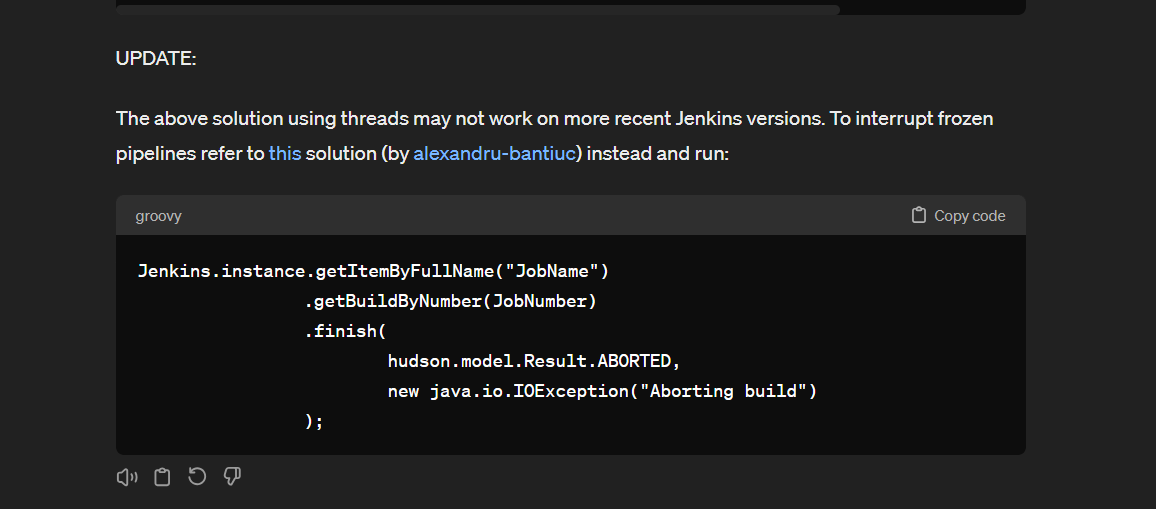


Figure 9: Sample of markdown replies

The [react-markdown package](https://www.npmjs.com/package/react-markdown) will be helpful in that case. It would be a nice addition to have if things went well with fine-tuning the model, to give markdown in the response.

**References:** [**llama 2 with lora question answering**](https://deci.ai/blog/fine-tune-llama-2-with-lora-for-question-answering/)**,** [**openassistant guanaco dataset**](https://huggingface.co/datasets/timdettmers/openassistant-guanaco)

#### Training procedure:

Fine-tuning is the process of adjusting the weights and parameters of a pre-trained model on new data to enhance its performance for a specific task. In our case, we’ll fine-tune llama2-chat using the Jenkins-specific knowledge dataset provided in the Jenkins document and verified data from other open sources.

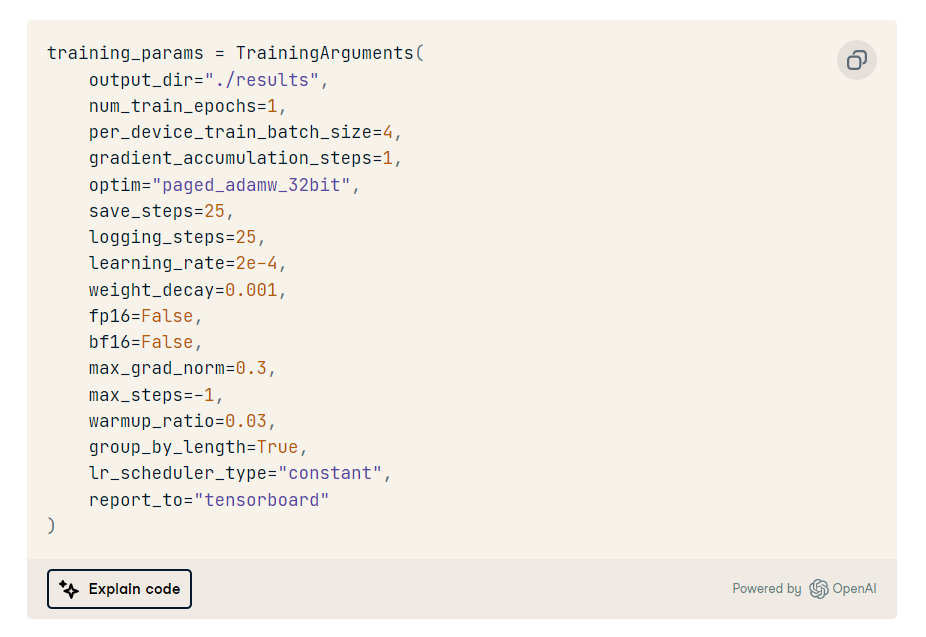


Figure 10: [LLaMA-2 training parameters](https://www.datacamp.com/tutorial/fine-tuning-llama-2)

Also, reading and finding more previous experiments on llama2 will be helpful.

[This experiment](https://github.com/brevdev/notebooks/blob/main/llama2-finetune.ipynb) mentioned a note that **advises** initially setting a high value for **max\_steps** during model training and observing the point where the model's performance starts declining.

This serves as a guide to finding the optimal number of steps to execute.

For example, if, **at 500 steps**, the model begins to overfit, with the validation loss increasing while the training loss decreasing significantly, it suggests that the model is learning the training data well but struggles to generalize to new data.

⇒ Therefore, 500 steps would be deemed the optimal point. As a result, in step 6 of the process, the checkpoint-500 model repository in the output directory (llama2-7b-finetune-viggo) would be used as the final model.

That can give a nice indication of how the parameters will be modified depending on the model performance after training and iterate over the training again until I can find some stable ground. (max\_steps = 1000), also iterating on the whole dataset multiple times can result in a more robust model as well.

#### Model Performance:

1. **BLEU score:**

A metric used to automatically evaluate the quality of machine-translated text by comparing it to a set of high-quality reference translations. It ranges from 0 to 1, where 0 indicates no overlap with the reference translation (low quality) and 1 indicates perfect overlap (high quality). AutoML often expresses BLEU scores as percentages rather than decimals.

Interpreting BLEU scores:

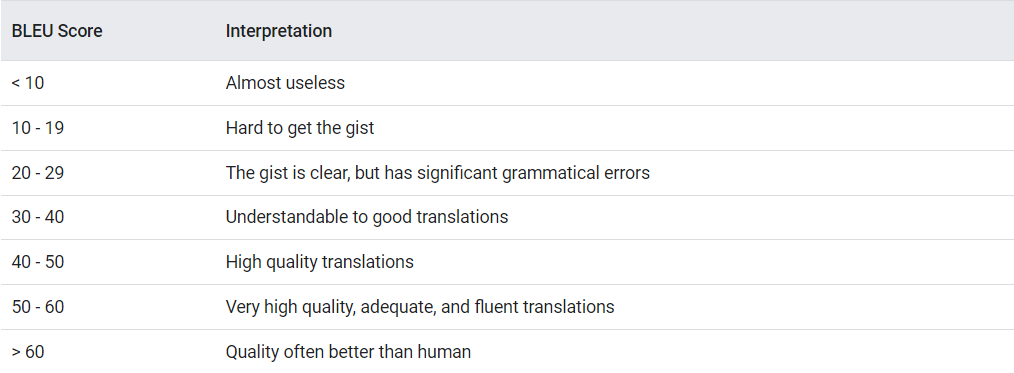


Figure 11: The interpretation of BLEU score

The article provided in the references of this section used the BLEU metric to measure the performance of LLAMA2 trained on translation problem

When we design a LLM system, it’s important to have a set of benchmark questions with “good quality” answers that we can use to evaluate our LLM. These benchmark questions would serve as our reference data, similar to a test data set in more traditional machine learning tasks like image classification. We can have some indication of how model is performing, the article provided a comparison between different pretrained models based on llama and how they performed on BLEU metric

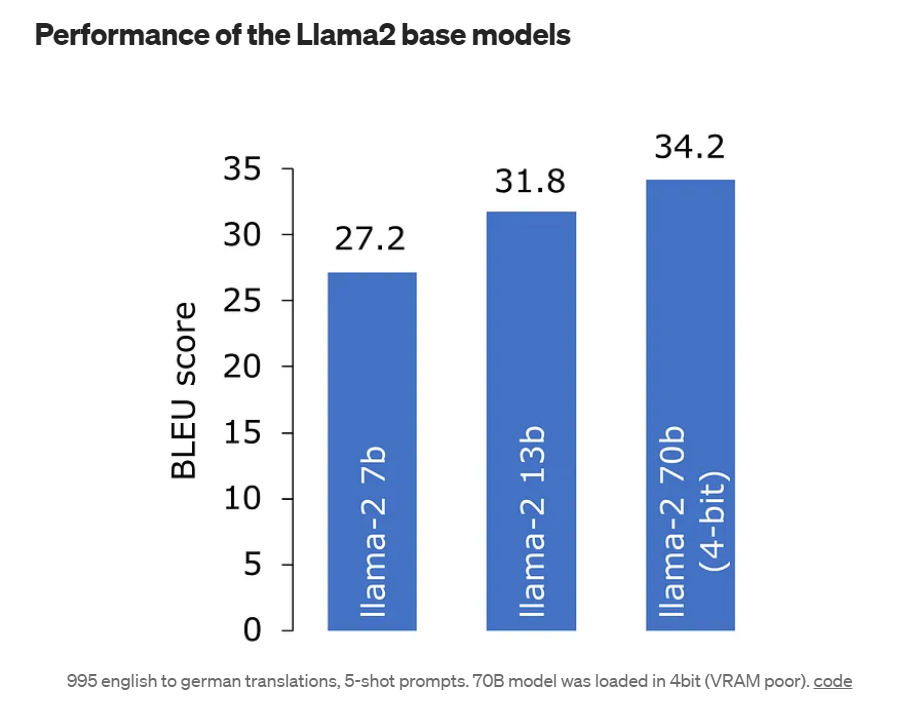


Figure 12: Performance of the Llama-2 base models

To compute the BLEU score, we will utilize the evaluate library from HuggingFace which implements the BLEU score.

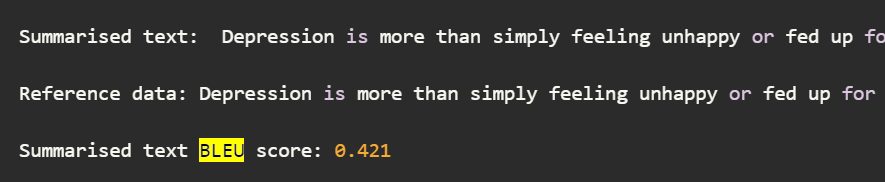
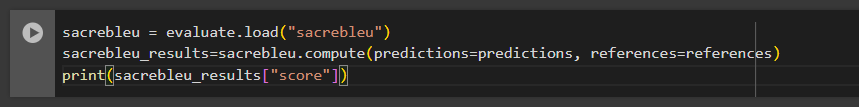


Figure 13 - 14: BLEU metric calculation and Example results

The BLEU score is 0.421, which is good: a score between 0.4-0.5 represents a high quality output. As a reference, a score above 0.6 is often considered to be better than a human.

If we were to compare a random sentence against our reference, we would expect a score of 0, meaning that there is no overlap between the sentences.

**References:** [**llama 2 the bleu score**](https://medium.com/@geronimo7/evaluating-language-competence-of-llama-2-based-models-the-bleu-score-d44c651a5e58)

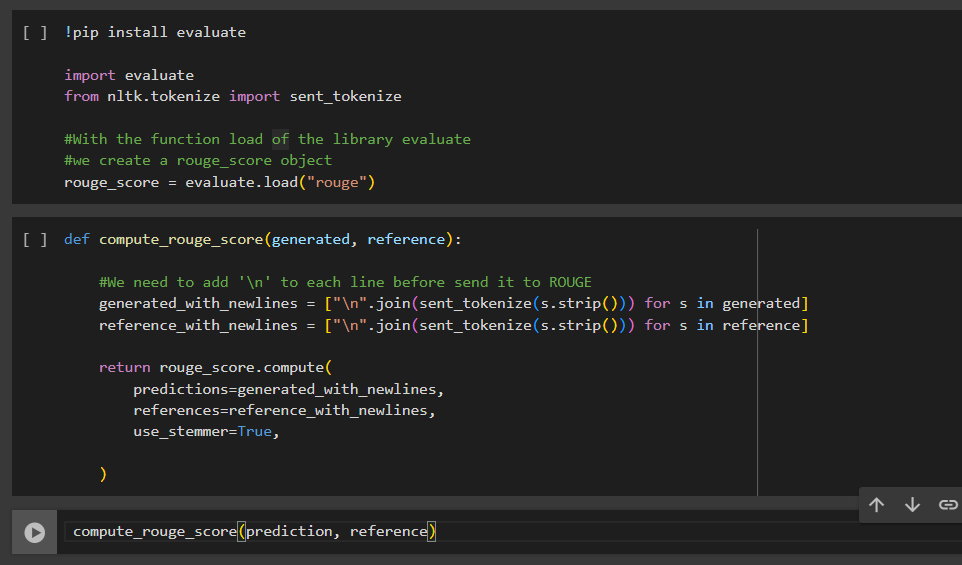
1. **Rouge score:**

A metric used to assess how closely a language model's generated text resembles reference texts, commonly employed in tasks like summarization and paraphrasing (this can be adjusted also to our problem). Its calculation involves various methods such as ROUGE-N and ROUGE-L which compare the generated text to one or more reference texts, determining a score based on their overlap.

ROUGE is a set of metrics used primarily for evaluating the quality of text summarization. It measures the overlap between the generated summary and the reference summaries or documents. but from my understanding of this article, it could help us with the question-answering problem. It can still provide some insights into their performance, particularly in tasks involving summarization or generation of responses.

In the context of question-answering chatbots, ROUGE can be used as an indication of how well the generated responses capture the key information or salient points from the input question or context.

We'll use an extra tool called the evaluate module, which is part of a library. It's like having a group activity to see how well sentences match up. Here's how it goes:



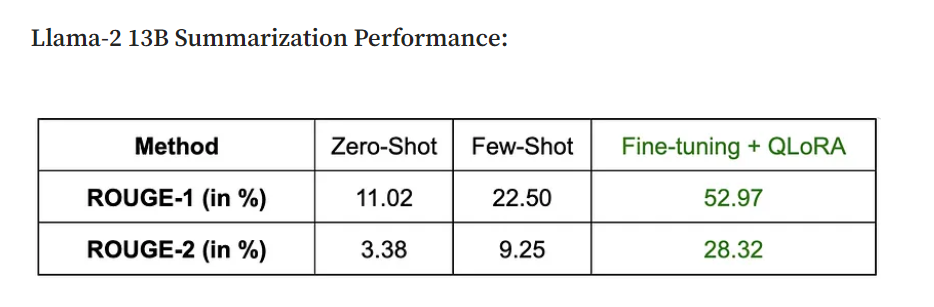


Figure 15 - 16: Rouge metric calculation and Llama-2 13B Summarization Performance

While ROUGE metrics offer valuable insights into language evaluation, they come with their quirks. Imagine a computer repeating the same word over and over, like "fix, fix, fix, fix." ROUGE might **SUPPORT** this, but that's not great writing.

To figure this out, we use the clipping function "rule" that can be used to limit the counting of repeated words "Don't count the same word too much.". This keeps things fair and accurate, making sure our evaluation truly reflects the quality of the generated text.

[**ROUGE metrics have their own limitations**](https://dev.to/aws-builders/mastering-rouge-matrix-your-guide-to-large-language-model-evaluation-for-summarization-with-examples-jjg)**.** Let's look at some of these:

* ROUGE loves comparing words, but it's not so great at telling if words are like cousins – different on the outside but similar on the inside. This can be a problem when words have different faces but mean similar things. **"happy"** and **"joyful"** – they're like word twins, but ROUGE doesn't always see that.
* ROUGE checks if words like to be together, but it doesn't care much about where they stand. This can be tricky. Imagine shuffling the words in a sentence – ROUGE might still think it's all good even if the meaning changes a lot.

so we usually use rouge with other metrics to evaluate the LLM model performance.

**References:** [**the practical guide to LLMs llama 2**](https://medium.com/georgian-impact-blog/the-practical-guide-to-llms-llama-2-cdf21d540ce3)

1. **Perplexity:**

It measures how well a language model predicts text. It's calculated as the inverse probability of a test set normalized by the number of words. In simpler terms, lower perplexity indicates better prediction accuracy.

For code snippets and insights on how to get performance measured using perplexity:

we can simply pass the input\_ids as the labels to our model, and the average negative log-likelihood for each token is returned as the loss. With our sliding window approach (summing the probability based on all past sequences).

However, there is an overlap in the tokens we pass to the model at each iteration. We don’t want the log-likelihood for the tokens we’re just treating as context to be included in our loss, so we can set these targets to -100 so that they are ignored. The following is an example of how we could do this with a stride of 512. This means that the model will have at least 512 tokens for context when calculating the conditional likelihood of any one token (provided there are 512 preceding tokens available to condition on).

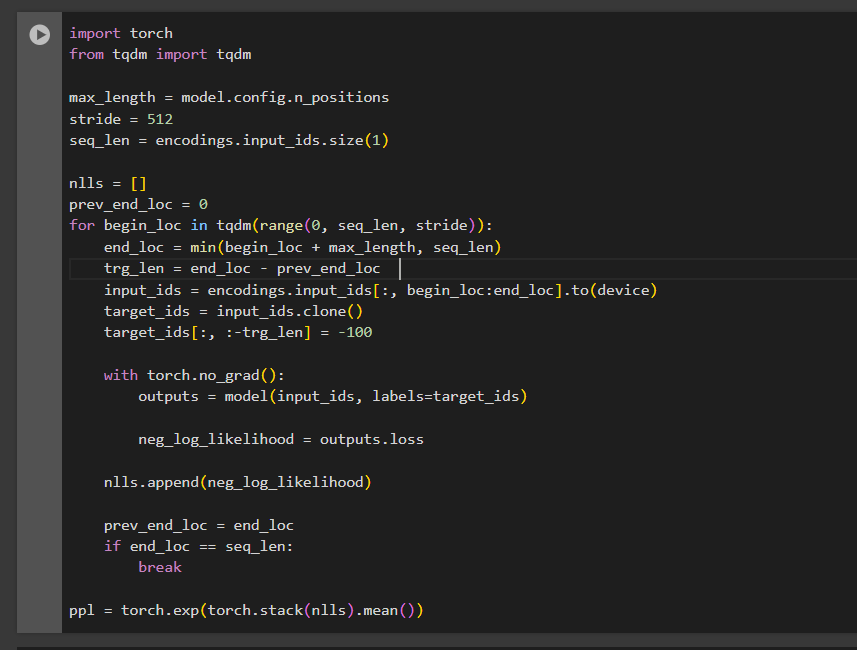


Figure 17: Perplexity metric calculation

through old experiments, we found that Perplexity as a function of context size for the LLaMA-1 (black) and LLaMA-2 (red) 13B models can be visualized as the following

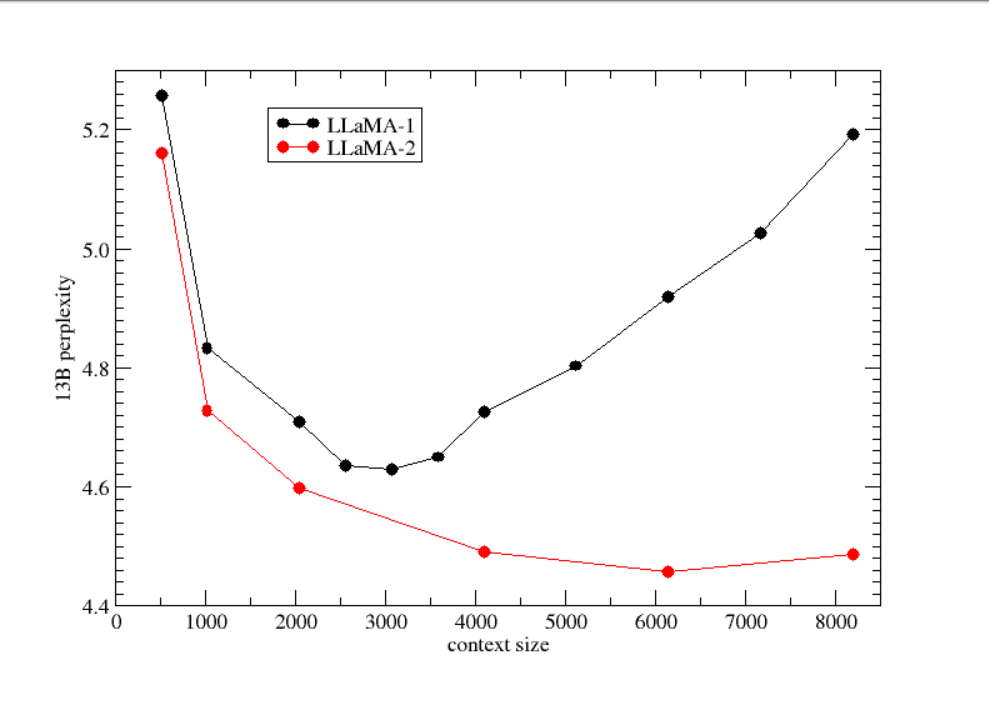


Figure 18: Perplexity as a function of context size for the LLaMA-1 and LLaMA-2 13B models.

**A few key points to note:**

* A model with a vocabulary of 10,000 words and a perplexity of 2.71 is much better than a model with a vocabulary of 100 words and the same perplexity score of 2.71.  
  We can reference some old experiment used this score to evaluate their LLM and reference the vocab size they have and what we have
* Lower perplexity results in higher consistency. As we know, LLMs are non-deterministic, i.e., the same inputs can result in two different outputs; a lower perplexity means that the model is more likely to produce the same output over multiple runs.
* Perplexity is the inverse of the geometric mean of the probability of each word. Hence, the inverse of average probability (2.3077 in the above case) can be considered a good proxy for quick calculations.
* This calculation happens in the tokens space (compared to the word space), but the core principle remains the same.

**References:** [**calculating perplexity of llama 2**](https://www.kaggle.com/code/philculliton/calculating-the-perplexity-of-4-bit-llama-2)**,** [**llama 2 perplexity visualizations**](https://github.com/ggerganov/llama.cpp/discussions/2352)**,** [**perplexity HF**](https://huggingface.co/docs/transformers/en/perplexity)

**Improving the score of an LLM:**

It is a case were we can find the score of our fine-tuned model to be somehow low at first, to enhance the score we can think of different approaches:

* **Comparison:** Benchmark our model's BLEU score against existing pre-trained models based on LLAMA, as discussed in the provided references. This comparative analysis will provide insights into the relative performance of our model.

This GitHub project, found at [MU-LlaMA](https://github.com/shansongliu/MU-LLaMA) , offers insights into how the BLEU and Rouge scores can be applied within llama-specific tasks. By exploring various pretrained models available through open-source projects, valuable references can be gathered.

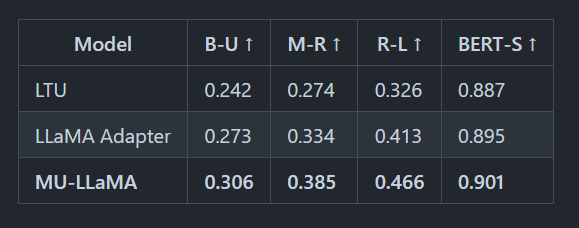


Figure 19: Obtained results using different evaluation metrics on Llama2 performance.

* **Data Augmentation:** Increase the diversity and volume of training data by augmenting it with additional text. This can help the model capture a wider range of language patterns and improve answering quality.

This can be done by using some paraphrasing pretrained model as a preprocessing step on the data example: [bart-paraphrasing](https://huggingface.co/eugenesiow/bart-paraphrase) so we can make sure we can catch different patterns.

* **Data Balancing:** Ensure that the training data is balanced across different language pairs, domains, and sentence lengths. Imbalanced data can lead to biases in the model's training and affect answering quality.

Making sure that the training data is balanced if we found a low BLEU score

* **Utilize External Resources:** Leverage external resources such as bilingual dictionaries, parallel corpora, or domain-specific terminology lists to improve the accuracy and vocabulary coverage.

In our case other sources of Jenkins knowledge can help in this point

#### Pipeline:

Now let’s put all previous knowledge together to achieve the project objective in meaningful and actionable steps.

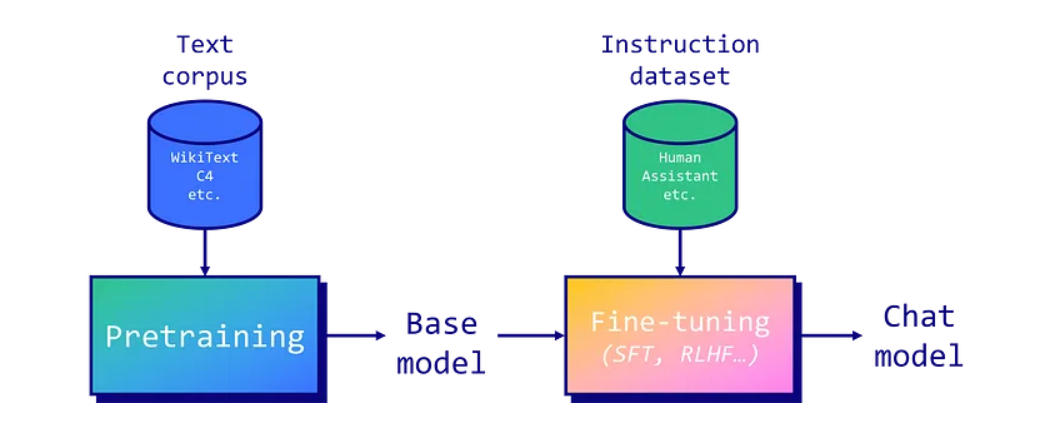


Figure 20: Fine-tuning LLM model pipeline

**Preknowledge:**

1. **LLM Pretraining:**
   * Large Language Models (LLMs) are pre trained on extensive text corpora.
   * Llama 2 was pretrained on a dataset of 2 trillion tokens.
2. **Auto-Regressive Prediction:**
   * Llama 2, an auto-regressive model, predicts the next token in a sequence.
   * Auto-regressive models lack usefulness in providing instructions, leading to the need for instruction tuning.
3. **Fine-Tuning Techniques:**

* Instruction tuning uses two main fine-tuning techniques: a. Supervised Fine-Tuning (SFT): Trained on instruction-response datasets, minimizing differences between generated and actual responses. b. Reinforcement Learning from Human Feedback (RLHF): Trained to maximize rewards based on human evaluations.

1. **RLHF vs. SFT:**
   * RLHF captures complex human preferences but requires careful reward system design and consistent human feedback.
   * Direct Preference Optimization (DPO) might be a future alternative to RLHF.
   * SFT can be highly effective when the model hasn't encountered specific data during pretraining.
2. **Effective SFT Example:**
   * LIMA paper showed improved performance of LLaMA v1 model over GPT-3 by fine-tuning on a small high-quality dataset.
   * Data quality and model size (e.g., 65b parameters) are crucial for successful fine-tuning.
3. **Importance of Prompt Templates:**
   * Prompt templates structure inputs: system prompt, user prompt, additional inputs, and model answer.
   * Different templates (e.g., Alpaca, Vicuna) have varying impacts.
4. **Reformatting for Llama 2:**

* Converting the instruction dataset to Llama 2's template is important.

1. **Base Llama 2 Model vs. Chat Version:**

* Specific prompt templates are not necessary for the base Llama 2 model, unlike the chat version.

(Note: LLMs = Large Language Models, SFT = Supervised Fine-Tuning, RLHF = Reinforcement Learning from Human Feedback, DPO = Direct Preference Optimization)

**Fine-Tuning Llama 2 (7 billion parameters) with VRAM Limitations and QLoRA:**

The goal is to fine-tune a Llama 2 model with 7 billion parameters using a T4 GPU with 16 GB of VRAM. Given the VRAM limitations, traditional fine-tuning is not feasible, necessitating parameter-efficient fine-tuning (PEFT) techniques like LoRA or QLoRA. The chosen approach is QLoRA, which employs 4-bit precision to drastically reduce VRAM usage.

**The following steps will be executed:**

1. **Loading the Dataset:**

* The first step involves loading the preprocessed dataset. This dataset will be used for fine-tuning. Preprocessing might involve reformatting prompts, filtering out low-quality text, and combining multiple datasets if needed.
* Reformatting is a process that requires a training dataset tailored specifically to match the model’s template, samples can be found through [[example](https://huggingface.co/datasets/kelSidenna/softwareReq-data?row=3)]



Figure 21: Sample of model Input

1. **Launching the fine-tuning**

**Parameters to tune**

* Load a llama-2-7b-chat-hf model
* specify the TrainingArguments as mentioned before

**QLoRA parameters**

* QLoRA will use a rank of 64 with a scaling parameter of 16.
* The Llama 2 model will be loaded directly in 4-bit precision using the NF4 type.
* The model will be trained for one epoch.

**Other parameters**

* cam through [PeftModel](https://huggingface.co/docs/peft/package_reference/peft_model) and [SFTTrainer](https://huggingface.co/docs/trl/main/en/sft_trainer)  
  It is worth mentioning that the PEFT (Parameter Efficiency Fine-Tuning) framework integrated in the LLaMA 2 models family allows advanced training techniques, such as k-bit quantization, low-rank approximation, and gradient checkpointing, resulting in more efficient and resource-friendly models. Compared with original models, quantized language models stake a smaller memory footprint.

*# Maximum sequence length to use*

max\_seq\_length = None

*# Pack multiple short examples in the same input sequence to increase efficiency*

packing = False

*# Load the entire model on the GPU 0*

device\_map = {"": 0}

1. **Configuring BitsAndBytes for 4-bit Quantization:** The BitsAndBytesConfig is set up to enable 4-bit quantization. This configuration is crucial for reducing the memory usage during fine-tuning.
2. **Loading Llama 2 Model and Tokenizer in 4-bit Precision:** The Llama 2 model is loaded with 4-bit precision, which significantly reduces the memory footprint. The corresponding tokenizer is also loaded to preprocess the text data.
3. **Loading Configurations and Initializing SFTTrainer:**

* The configurations needed for QLoRA, which is a parameter-efficient fine-tuning technique, are loaded.
* Regular training parameters are set up.
* The SFTTrainer is initialized with all the loaded configurations and parameters. This trainer will manage the supervised fine-tuning process.

1. **Start of Training:** After all the necessary components are loaded and configured, the training process begins. The SFTTrainer takes care of fine-tuning the Llama 2 model using the specified dataset, configurations, and parameters.
2. **Measure model performance:** using the metrics provided above we can measure model performance on a golden test set and act according to the results as described in the model performance section

#### Customizing a conversational agent:

LLMs exhibit strong capacities to understand natural language and solve complex tasks via text generation. In order to mold text generation for a conversational chatbot we can apply LangChain framework rather than specifying a conversational task (as the pipeline’s input) as this alternative provides the memory feature able to store and retrieve information during a conversation, we request memorizing up to 5 past conversations (setting k=5).

1. **Defining prompts:**

A carefully-crafted prompt acts as a navigational tool, guiding the model to produce accurate and coherent outputs. Without appropriate prompting, users might receive vague or off-target answers.

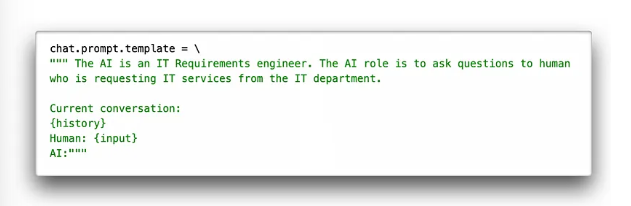


Figure 22: Sample of input prompt using user question and the history

1. **Testing the agent:**

In this step the chatbot was tested to engage in a focused conversation. Applying the ‘chat\_trim’ function may be necessary in order to execute additional refinements by looking for specific suffixes like ‘AI:’, ‘\nHuman:’ and ‘[]’. If found at the end of the response, these suffixes are trimmed off. This step ensures that any residual conversational artifacts or unwanted tokens are effectively removed.



Figure 23: Testing agent performance

**References**

This is an appendix on what I have reached searching for this problem

* <https://paperswithcode.com/paper/llama-open-and-efficient-foundation-language-1>
* <https://www.datacamp.com/tutorial/fine-tuning-llama-2>
* [full llama documentation be meta] <https://ai.meta.com/research/publications/llama-2-open-foundation-and-fine-tuned-chat-models/> , <https://llama.meta.com/llama2/>
* <https://www.kaggle.com/models/metaresearch/llama-2/code>

LLAMA2 vs Open Orca

* <https://sapling.ai/llm/llama2-vs-orca>
* <https://huggingface.co/docs/transformers/en/model_doc/llama>
* <https://huggingface.co/Open-Orca/Mistral-7B-OpenOrca>

Possible competitors models:

* <https://www.google.com/url?q=https://ollama.com/&sa=D&source=docs&ust=1710184087013225&usg=AOvVaw1rHQvOM-QUKGsqhfP82ZZR>

Model evaluation

* <https://cloud.google.com/translate/automl/docs/evaluate#bleu>
* <https://www.cs.cmu.edu/~roni/papers/eval-metrics-bntuw-9802.pdf>
* evaluation checkpoint: <https://aisera.com/blog/llm-evaluation/>

Fine-tuning LLAMA2 with lora method:

* <https://www.kaggle.com/code/philculliton/fine-tuning-with-llama-2-qlora>
* <https://github.com/ashishpatel26/LLM-Finetuning/blob/main/2.Fine_Tune_Your_Own_Llama_2_Model_in_a_Colab_Notebook.ipynb>
* <https://medium.com/@kel.sidenna/fine-tuning-into-tech-talk-llama-2-langchain-synergy-404c21c6444a>

### Future work:

For this use case fine-tuning was a better option as Fine-tuning tailors the entire model to a specific dataset, Fine-tuned models can often generate responses more quickly, Implementing and managing a fine-tuned model can be simpler than a RAG model

But will the RAG model work in that case? definitely yes and may perform same as the fine tuned model using the right prompt, question and indexed database

LLaMA 2 model is pre trained and fine-tuned with 2 Trillion tokens and 7 to 70 Billion parameters which makes it one of the powerful open source models. Including being trained on 40% more tokens, having a much longer context length (4k tokens).

FAISS (Facebook AI Similarity Search) is a library for efficient similarity search and clustering of dense vectors.

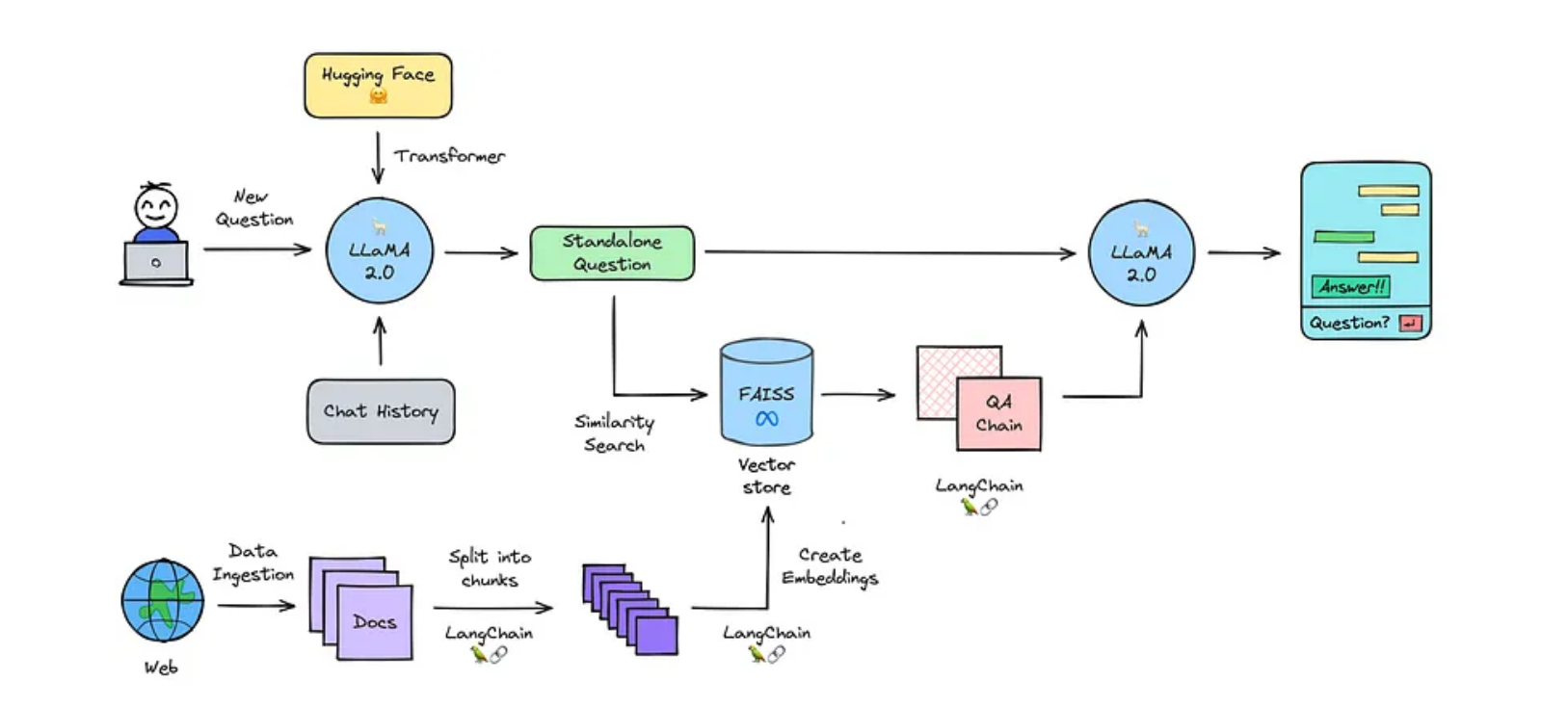


Figure 25: process data flow

* The process starts with initializing the text-generation model in that case it will most probably be Llama-2-7b-chat, Load documents data stored as text in the database.
* split into chunks and this is necessary to create small chunks because language models can handle a limited amount of text.
* create embeddings which are used to search and retrieve relevant documents in large databases.
* loading embedding into vector store FAISS can help in that case those stores perform well in similarity search using text embeddings.
* combining the chat history with the new question and turn them into a standalone question

Here we can find that we may face some context length issues with the model but luckily LangChain solves this problem, LangChain can load these documents and get the text from these documents. As the text is too big to be used for the context it needs to be splitted into several chunks, using **RecursiveCharacterTextSplitter**. We'll have to determine the size of the chunks, so it again won't be too large to be used for the context size. By setting a chunk overlap one can keep context between the chunks.

* searching for relevant information and passing the question to the question answering chain where we can generate an answer.

some mentors of openAi and many other places suggested that this is the best way to think of if you are going to make a chatbot which is document based refer to:

[[fine tuning with massive amount of documents](https://community.openai.com/t/fine-tuning-with-help-of-massive-amount-of-documents/84195)] [[fine tuning on a collection of text documents](https://github.com/meta-llama/llama/issues/730)]