# Jenkins

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

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

**Submitted By:** Nour Ziad Almulhem

**Contacts:** nouralmulhem@gmail.com | +201123309818

**Github Handle:** /nouralmulhem

# Contents

## ● 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.

## ● 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 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.

| | Training Data | Params | Context Length | GQA | Tokens | LR |
|---|---|---|---|---|---|---|
| LLAMA 1 | *See Touvron et al. (2023)* | 7B | 2k | ✗ | 1.0T | $3.0 \times 10^{-4}$ |
| | | 13B | 2k | ✗ | 1.0T | $3.0 \times 10^{-4}$ |
| | | 33B | 2k | ✗ | 1.4T | $1.5 \times 10^{-4}$ |
| | | 65B | 2k | ✗ | 1.4T | $1.5 \times 10^{-4}$ |
| LLAMA 2 | *A new mix of publicly available online data* | 7B | 4k | ✗ | 2.0T | $3.0 \times 10^{-4}$ |
| | | 13B | 4k | ✗ | 2.0T | $3.0 \times 10^{-4}$ |
| | | 34B | 4k | ✓ | 2.0T | $1.5 \times 10^{-4}$ |
| | | 70B | 4k | ✓ | 2.0T | $1.5 \times 10^{-4}$ |

Figure 1: LLAMA 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, the initial version of LLAMA 2-CHAt is created through the application of supervised fine-tuning. Subsequently, the model is iteratively refined using Reinforcement Learning with Human Feedback (RLHF) methodologies.

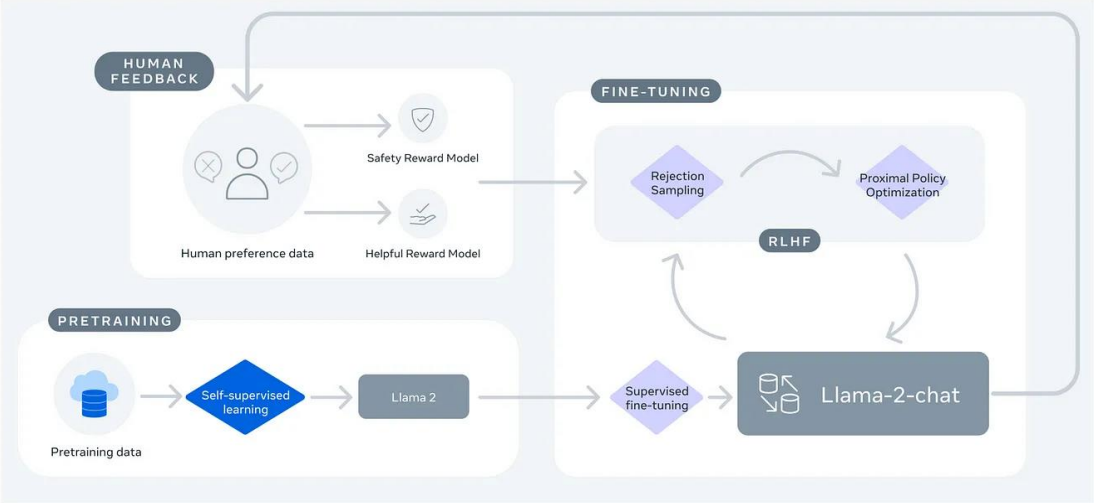



Figure 2: Training of LLAMA2-chat

## Dataset:

some raw data is provided through:

- Jenkins Blog posts and Documentation provided at [jenkins-infra/jenkins.io: A static site for the Jenkins automation server (github.com)]

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

- Latest community questions and answers provided at [Latest Using Jenkins topics - Jenkins]

One of the useful ideas was using [Discourse API Docs] to get questions asked on Jenkins community and along with their reliable answers by Jenkins experts, the endpoints returned Json objects that contains all data about each post and all their related answers with the correct solution marked.

- Questions asked on different platforms like stack overflow, ask ubuntu and etc. provided at [Hot Questions - Stack Exchange]

for many websites like [stackoverflow], [ask ubuntu] and [Software Quality Assurance & Testing ] we have [data explorer] 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

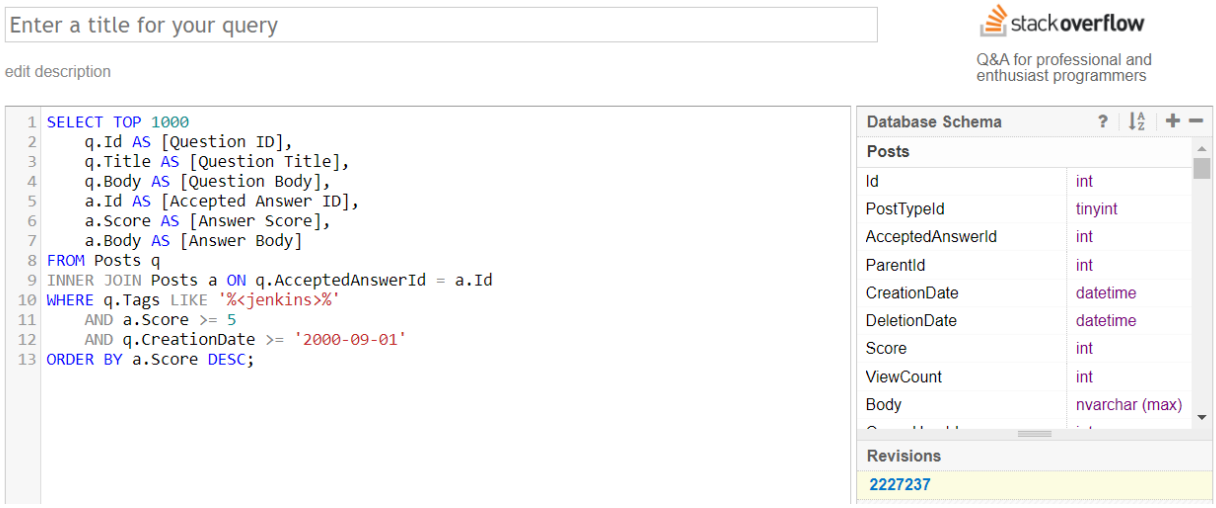
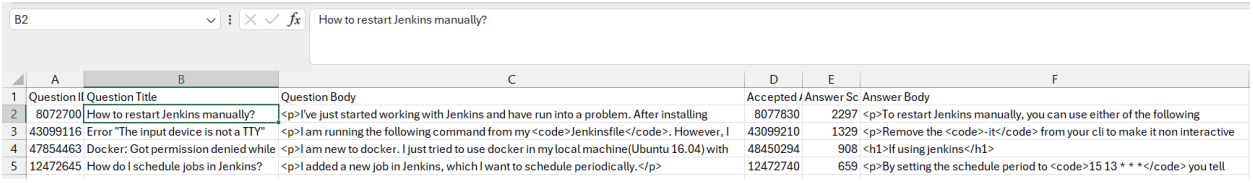



Figure 3: stackexchange query on jenkins tagged questions

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)




| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | Question Id | Question Title | Question Body | Accepted A | Answer Sc | Answer Body |
| 2 | 8072700 | How to restart Jenkins manually? | I've just started working with Jenkins and have run into a problem. After installing | 8077830 | 2297 | To restart Jenkins manually, you can use either of the following |
| 3 | 43099116 | Error "The input device is not a TTY" | I am running the following command from my <code>Jenkinsfile</code>. However, I | 43099210 | 1329 | Remove the <code>-it</code> from your cli to make it non interactive |
| 4 | 47854463 | Docker: Got permission denied while | I am new to docker. I just tried to use docker in my local machine(Ubuntu 16.04) with | 48450294 | 908 | <h1>If using jenkins</h1> |
| 5 | 12472645 | How do I schedule jobs in Jenkins? | I added a new job in Jenkins, which I want to schedule periodically. | 12472740 | 659 | By setting the schedule period to <code>15 13 * * *</code> you tell |

Figure 4: data set retrieved

**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, html tags, and removing tuples with code syntax, to ensure fine-tuning with text only.

As dataset are not usually something to change and can be appended easily so it can be stored in CSV files, refined dataset is also provided in our repository at [Dataset Folder]

```
{
    "Human": [
        "What hardware configuration is recommended for a small
team using Jenkins?"
    ],
    "Assistant": "\nRecommended hardware for a small team:\n- 4
GB+ of RAM\n- 50 GB+ of drive space\n    ",
    "ref": "https://www.jenkins.io/doc/book/installing/docker/"
}
```

Figure 5: Dataset example

**Future work:** after including code in the training data, the sections wrapped within an ( \'\'\' ) can 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

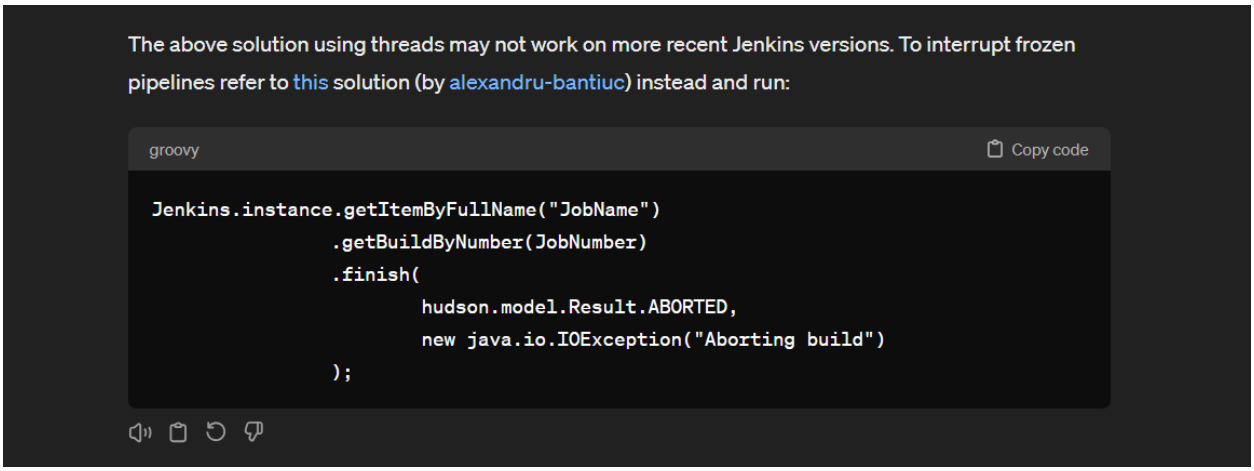



Figure 6: Sample of markdown replies

[react-markdown package] 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.

## 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.

```
training_params = TrainingArguments(
    output_dir="./results",
    num_train_epochs=1,
    per_device_train_batch_size=4,
    gradient_accumulation_steps=1,
    optim="paged_adamw_32bit",
    save_steps=25,
    logging_steps=25,
    learning_rate=2e-4,
    weight_decay=0.001,
    fp16=False,
    bf16=False,
    max_grad_norm=0.3,
    max_steps=-1,
    warmup_ratio=0.03,
    group_by_length=True,
    lr_scheduler_type="constant",
    report_to="tensorboard"
)
```

Figure 7: LLaMA-2 training parameters

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

This experiment 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.

## Pipeline:

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

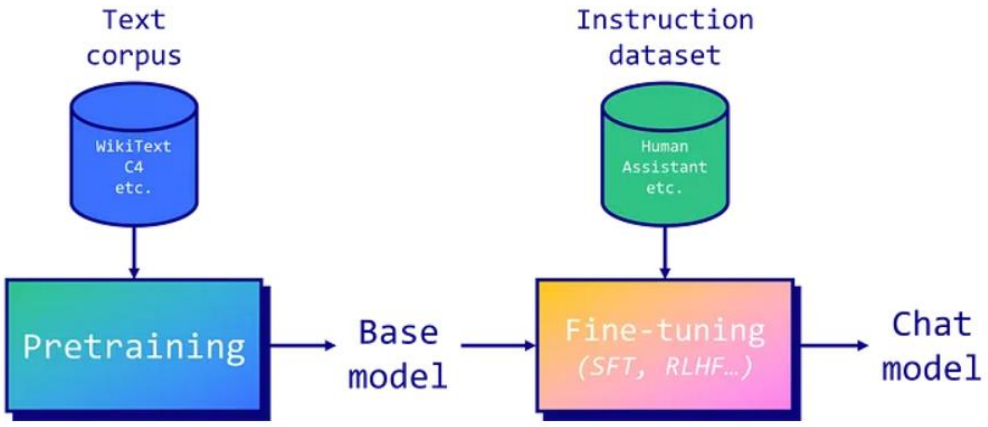



Figure 8: Fine-tuning LLM model pipeline

**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.
4. **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.
5. **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.

6. **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.
7. **Reformatting for Llama 2:**
   - Converting the instruction dataset to Llama 2's template is important.
8. **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]

```
<s>[INST] <<SYS>>
{{ system_prompt }}
<</SYS>>
{{ user_message }} [/INST] </s>
```

Figure 9: Sample of model Input

## 2. Launching the fine-tuning

4-bit quantization via QLoRA allows efficient finetuning of huge LLM models on consumer hardware while retaining high performance. This dramatically improves accessibility and usability for real-world applications.

QLoRA quantizes a pre-trained language model to 4 bits and freezes the parameters. A small number of trainable Low-Rank Adapter layers are then added to the model.

During fine-tuning, gradients are backpropagated through the frozen 4-bit quantized model into only the Low-Rank Adapter layers. So, the entire pretrained model remains fixed at 4 bits while only the adapters are updated. Also, the 4-bit quantization does not hurt model performance.

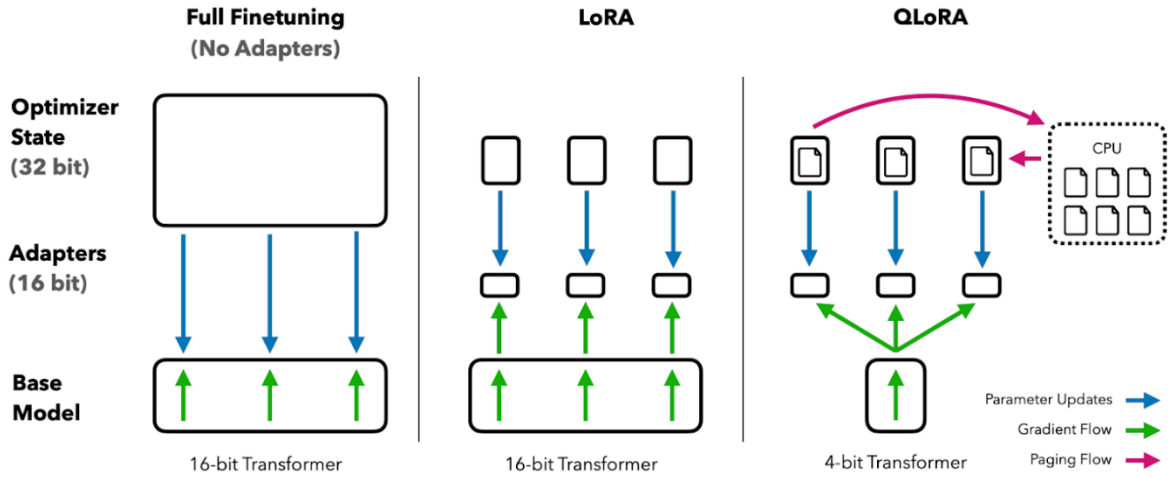



Figure 10: Image from QLoRA paper

**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 five epochs.

**Other parameters**

- came through PeftModel and SFTTrainer
  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.

3. **Configuring BitsAndBytes for 4-bit Quantization:** The BitsAndBytesConfig is set up to enable 4-bit quantization. This configuration is crucial for reducing memory usage during fine-tuning.
4. **Loading Llama 2 Model and Tokenizer in 4-bit Precision:** The Llama 2 model is loaded with 4-bit precision with the compute dtype "float16", which significantly reduces the memory footprint. The corresponding tokenizer is also loaded to preprocess the text data.
5. **Loading Tokenizer:** we will load the tokenizer from Hugginface and set padding_side to “right” to fix the issue with fp16.
6. **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.
7. **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.
8. Merge Lora weights with the base model and you now have a fine-tuned version of llama 2 on your data

## Convert fine-tuned to GGML format:

You can load this full model onto the GPU and run it like you would any other hugging face model, but we are here to take it to the next level of running this model on the CPU using llama.cpp.

Llama.cpp has a script called **convert_hf_to_gguf.py** that is used to convert models to the binary GGML format that can be loaded and run on CPU.

## Quantize GGML Model:

Part of the appeal of the GGML library is being able to quantize this binary model into smaller models that can be run even faster. There is a tool called quantize in the Llama.cpp repo that can be used to convert the model to different quantization levels. We used the quantize tool to shrink our model to q8_0

## Customizing a conversational agent:

LLMs exhibit strong capacities to understand natural language and solve complex tasks via text generation. 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.

```
chat.prompt.template = \
""" The AI is an IT Requirements engineer. The AI role is to ask questions to human
who is requesting IT services from the IT department.

Current conversation:
{history}
Human: {input}
AI:"""
```

Figure 11: Sample of input prompt using user question and the history

2. **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.

```
def chat_trim(chat_chain, query):
    chat_chain.predict(input=query)
    cc = chat_chain.memory.chat_memory.messages[-1].content
    cc = cc.split('\n\n')[0]
    cc = cc.strip()

    # Finding the first occurrence of '\nHuman:' and slicing the string accordingly
    human_idx = cc.find('\nHuman:')
    if human_idx != -1:
        cc = cc[:human_idx]

    # Further trimming
    for stop_text in ['AI:', '[]']:
        cc = cc.removesuffix(stop_text)

    return cc.strip()
```

Figure 12: Testing agent performance

## References


- Fine-Tuning LLaMA 2: A Step-by-Step Guide to Customizing the Large Language Model | DataCamp
- LLaMA 2: A Detailed Guide to Fine-Tuning the Large Language Model | by Gobi Shangar | Medium
- How to run Llama-2 on CPU after fine-tuning with LoRA | by OXEN AI. Build World-Class AI Datasets. Together. | Medium


# 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? 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 most 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.

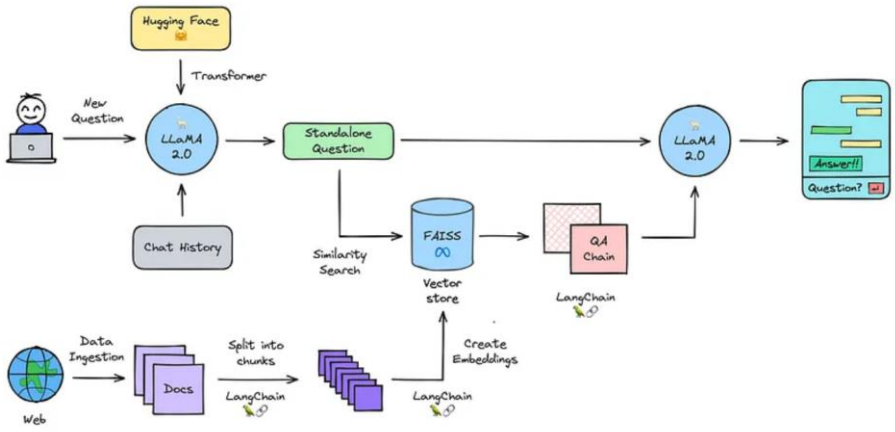



Figure 13: process data flow

- 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 split 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] [fine tuning on a collection of text documents]