Intel Unnati Industrial Training Program (20th May, 24 - 15th July, 24)

PS - 04 - Introduction to GenAI and Simple LLM Inference on CPU and finetuning of LLM Model to create a Custom Chatbot

TEAM MEMBERS
Aruniya Asokan
Yusra Inam
Srimeenakshi KS

MENTOR'S NAME
Dr. Shridevi S

Introduction

Within artificial intelligence, the field of generative AI (GenAI) focuses on building models that can produce new content. These models are capable of producing text, graphics, audio, and other types of data, frequently with a great degree of originality and coherence. GenAI models are intended to produce outputs that were not specifically included in their training data, in contrast to typical AI models that concentrate on classification or prediction. The creative industries as well as practical automation tools can benefit greatly from this capacity to produce innovative and distinctive material.

A subtype of GenAI called Large Language Models (LLMs) is created expressly to tackle tasks related to natural language processing. These models may produce writing that resembles that of a human being when given a prompt since they have been trained on large datasets with a variety of text formats. Deep neural networks are usually used in the design, and transformers are frequently used because of how well they handle sequential data.

A trained model is used to produce outputs based on new inputs through a process called inference. According to a given prompt, LLMs are required to generate text. Inference can be carried out on a CPU, although these models are typically operated on GPUs for efficiency.

A pre-trained LLM is fine-tuned by retraining it on a more specialized, smaller dataset. Through this procedure, the parameters of the model are modified to better suit the particular tasks or domain of the customized chatbot.

Project Planning and Objective Setting

Our team is passionate about mental health, and we've developed a chatbot named AMY, which stands for the initials of our team members. We've integrated artificial intelligence and machine learning into AMY to enhance its capabilities. Understanding the significant influence mental health has on people and communities, our goal is to create an advanced tool that offers guidance, assistance, and information on mental health-related issues.

AMY is intended to provide users with fact-based information, sympathetic reactions, and encouraging messages. We guarantee that AMY provides precise, considerate, and beneficial interactions by utilizing our ML and AI capabilities, ultimately leading to improved mental health outcomes. This initiative demonstrates our commitment to addressing one of the most important facets of human well-being using cutting-edge technology.

Model Used

The model "meta-llama/Llama-2-7b-chat-hf" is an advanced large language model developed by Meta (formerly Facebook) AI, designed to facilitate sophisticated natural language processing tasks. With 7 billion parameters, this model exemplifies a significant leap in the ability to understand and generate human-like text, making it particularly adept for conversational agents and chatbots. The extensive parameter count enables it to capture nuanced language patterns, allowing for more accurate and contextually appropriate responses.

The "meta-llama/Llama-2-7b-chat-hf" model is built on the transformer architecture, which has become the cornerstone of modern NLP due to its ability to handle large-scale data and complex language understanding tasks. This architecture allows the model to manage dependencies across different parts of a sentence, leading to more coherent and contextually relevant text generation. Its training involves vast amounts of text data, which enhances its capability to generate human-like responses in a wide variety of scenarios.

One of the standout features of this model is its balance between computational efficiency and performance. While it boasts a substantial number of parameters, it is optimized to run on both CPUs and GPUs, making it accessible for a broad range of applications. This efficiency ensures that developers can deploy high-quality NLP solutions without the need for extensive computational resources, thereby broadening its usability across different sectors.

Moreover, the model's design is particularly well-suited for fine-tuning. This means it can be adapted to specific domains or tasks, improving its performance based on the context in which it is used. For instance, when fine-tuned for a customer service chatbot, it can learn to handle inquiries specific to that domain, providing more accurate and helpful responses. This flexibility makes "meta-llama/Llama-2-7b-chat-hf" an invaluable tool for developing customized NLP applications that meet specific business or research needs.

The AMY Dataset

Our team assembled a large and diversified dataset to fully cover a range of mental health themes and scenarios to construct an advanced mental health chatbot.

Thousands of conversation threads from a variety of scenarios were collected to better understand the real-world issues raised by people looking for mental health care. Questions, coping mechanisms, personal experiences, and support-seeking behavior are all covered in these threads.

Firsthand information on mental health experiences is provided by the dataset, which focuses on symptoms, coping strategies, treatment experiences, and emotional

well-being. The collection also includes dialogues from supportive counseling, motivational interviewing, and cognitive behavioral therapy.

Intel Developers Cloud

Intel Developer Cloud is a comprehensive platform designed to provide developers with access to Intel's advanced hardware and software tools for building, testing, and optimizing applications. It offers a suite of resources including virtual machines equipped with the latest Intel processors, AI and machine learning frameworks, and development environments. This cloud-based platform facilitates the development of high-performance computing solutions by allowing developers to leverage Intel's cutting-edge technology, ensuring efficient and effective application deployment and scalability. Whether for AI, IoT, or enterprise applications, Intel Developer Cloud supports innovation and accelerates time-to-market for new technologies.

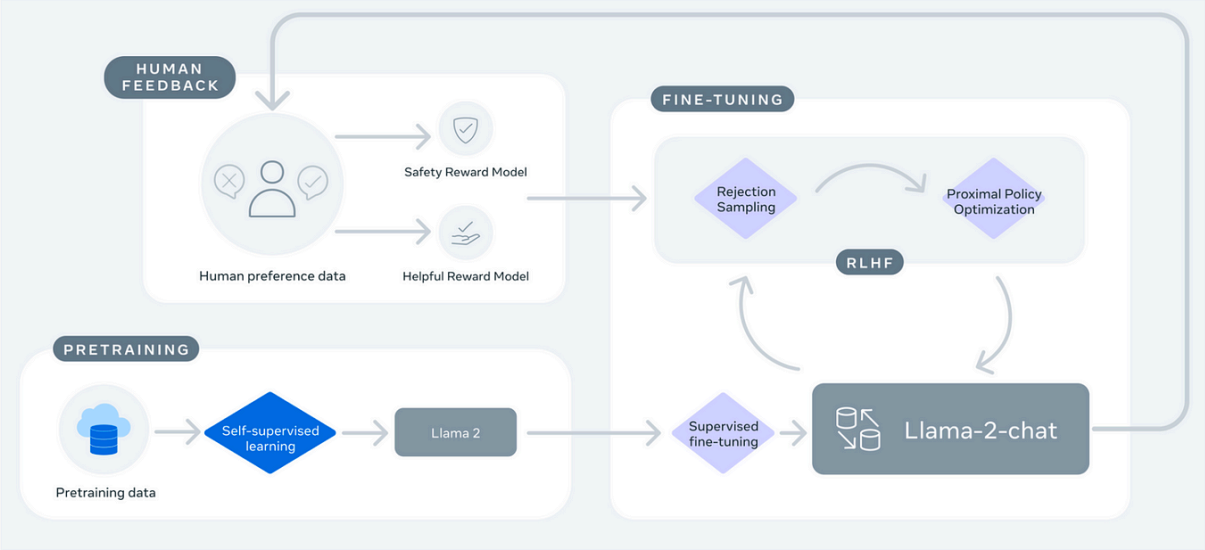
Intel Extension for Transformers

The Intel Extension for Transformers, available on GitHub, is an advanced library designed to optimize the use of transformer models on Intel platforms. This extension allows developers to build chatbots and other natural language processing (NLP) applications efficiently. It provides state-of-the-art (SOTA) compression techniques for large language models (LLMs), ensuring that these models can be run effectively even on devices with limited resources.

By leveraging Intel's hardware capabilities, the extension significantly enhances the performance and speed of transformer models, making it possible to deploy sophisticated NLP solutions quickly and reliably. The library is particularly useful for those looking to maximize the capabilities of Intel CPUs and GPUs, offering robust support for model training and inference. This makes the Intel Extension for Transformers a valuable tool for developers aiming to implement high-performance, scalable NLP applications on Intel hardware.

Model	Architecture	Parameters (Billion)	Training Data (Tokens, Billion)	Training Time (Days)	Use Cases
LLaMA	Transformer	13	1	30	General-purpose
GPT-3	Transformer	175	300	30	General-purpose
OPT	Transformer	175	1000	60	General-purpose
T5	Transformer	11	750	60	Text Generation, Translation

System Architecture



Tools Used

In our project, we've utilized several tools and libraries to preprocess the dataset, fine-tune the language model, and set up the chatbot for interaction. Here are the primary tools and libraries used:

1. Transformers: A library by Hugging Face that provides state-of-the-art general-purpose architectures for NLP, including tokenizers and models.
    - AutoTokenizer: For tokenizing the input text.
    - AutoModelForCausalLM: For loading the causal language model.
    - TrainingArguments and Trainer: For setting up and running the training loop.
    - pipeline: For creating a pipeline for text generation.

2. Datasets: Another library by Hugging Face for loading and processing datasets.
    - Dataset: For creating and handling datasets.
    - load_dataset: For loading datasets from various sources.

3. Data Collator:
    - DataCollatorWithPadding: For dynamically padding the sequences to the maximum length within each batch.

4. Scikit-learn:
    - train_test_split: For splitting the dataset into training and testing sets.

5. Pandas: For data manipulation and transformation.
    - DataFrame: For handling and processing data in tabular form.

6. JSON: For reading and parsing JSON files.

7. Torch: For utilizing PyTorch functionalities.
    - torch.device: For specifying the device (CPU in this case).

These tools were used together to preprocess the dataset, tokenize the input text, configure the model, train it, and finally interact with the fine-tuned chatbot model. The following is a description of the steps where these tools were used:

1. Preprocessing and Dataset Handling:
    - JSON and Pandas were used to load and preprocess the dataset.
    - Scikit-learn's 'train_test_split' was used to split the dataset into training and testing sets.

2. Tokenization and Transformation:
    - Hugging Face's 'AutoTokenizer' was used to tokenize the input text.
    - Dynamic padding was applied using 'DataCollatorWithPadding'.

3. Model Training:
    - AutoModelForCausalLM was used to load the pre-trained causal language model.
    - 'TrainingArguments' and 'Trainer' from the transformers library were used to configure and run the training loop.

4. Inference:
    - The `pipeline` function from transformers was used to create a text generation pipeline for interacting with the fine-tuned model.

Implementation

Jupyter Notebook, a powerful tool for interactive development and documentation, was used to execute our code. This notebook guided us through the process of preparing a custom dataset, transforming it, and configuring a tokenizer for training a language model using Hugging Face's tools.

Initially, we set up the environment by installing the necessary libraries such as `transformers`, `accelerate`, `bitsandbytes`, and `datasets`. These libraries were

essential for handling model training and data processing. After the installations, we logged into the Hugging Face Hub using an API token, which allowed us access to pre-trained models and datasets.

Next, we loaded our custom dataset, which was in JSON format and contained intents for chatbot training. We extracted relevant data from the JSON file, focusing on tags, patterns, and responses associated with each intent. This extraction process was crucial for transforming the raw data into a structured format that could be utilized for model training.

Following the data extraction, we transformed the extracted data into a structured format using a Pandas DataFrame. This transformation enabled us to split the data into training and test sets, facilitating model evaluation later in the process. Splitting the data ensured that we could effectively train the model and assess its performance on unseen data.

We then initialized a tokenizer from a pre-trained model, specifically "meta-llama/Llama-2-7b-chat-hf". This tokenizer was used to convert the text data into a format suitable for model training. The initialization of the tokenizer was a crucial step in preparing the data for the model.

A function was defined to transform the conversation data into a specific input format expected by the model. We applied this function to both the training and test datasets, ensuring each conversation was formatted correctly. This transformation was vital for maintaining consistency in the data fed into the model during training.

The transformed DataFrames were then converted into Hugging Face Dataset objects, making them compatible with the model training pipeline. This conversion was necessary for leveraging the capabilities of the Hugging Face ecosystem in training the language model.

We then verified the transformed datasets by printing samples from the training and test sets. This step confirmed that the data was correctly formatted. We also set the padding token for the tokenizer to ensure proper handling of input sequences during training. Additionally, further installations and imports of required libraries were carried out to prepare the environment for the subsequent model training and fine-tuning steps.

After that, we import essential libraries such as `torch`, `transformers`, `datasets`, and components like `AutoModelForCausalLM`, `AutoTokenizer`, `TrainingArguments`, `Trainer`, and `DataCollatorWithPadding`. These libraries provide the foundational tools needed for manipulating and training neural network models for natural language processing tasks.

Next, various parameters are configured to tailor the training process. This includes specifying the `model_name`, which determines the base architecture of the pre-trained language model to be fine-tuned. Paths for `new_model` and `output_dir` are defined to store the trained model and results respectively. Parameters such as `num_train_epochs`, `per_device_train_batch_size`, `learning_rate`, `optim`, and others govern the specifics of how the model will be trained, such as the number of epochs, batch sizes, optimizer type, and learning rate scheduling.

Once parameters are set, the tokenizer is initialized using `AutoTokenizer.from_pretrained(model_name)`. This tokenizer is essential for converting raw text inputs into numerical representations that the model can process. A check is performed to ensure the tokenizer has a padding token (`tokenizer.add_special_tokens({'pad_token': tokenizer.eos_token})`), crucial for ensuring all input sequences are of uniform length during training.

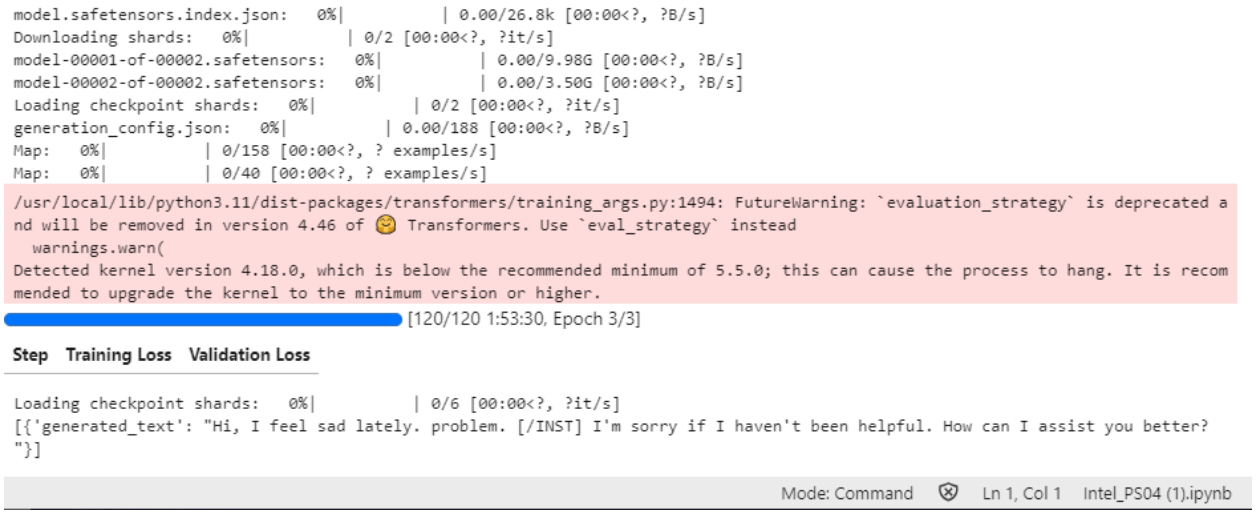
The pre-trained causal language model (`AutoModelForCausalLM.from_pretrained(model_name)`) is then loaded and moved to the specified device (`model.to(device)`), typically a CPU in this case (`device = torch.device("cpu")`). Model configurations may be adjusted (`model.config.use_cache = False`, `model.config.pretraining_tp = 1`) based on specific requirements or characteristics of the task at hand, such as disabling caching or adjusting pretraining parameters.

For tokenizing datasets, a function `tokenize_function` is defined to process input examples. This function utilizes the previously initialized tokenizer to tokenize input texts, padding them to a maximum length (`max_seq_length`) and preparing corresponding labels for the model training.

Actual datasets are then loaded using `Dataset.from_pandas(transformed_train_df)` and `Dataset.from_pandas(transformed_test_df)`, converted into tokenized forms (`tokenized_train_dataset` and `tokenized_test_dataset`) using the `map` function with `tokenize_function`. This step ensures that the data is formatted and prepared in a way that the model can efficiently process during training.

A `DataCollatorWithPadding` instance is created using the tokenizer, which helps in dynamically padding batches of data to ensure consistent input sizes during training (`DataCollatorWithPadding(tokenizer)`).

Training configuration is set up using `TrainingArguments`, specifying various training parameters such as batch sizes, optimizer settings, logging intervals, learning rate scheduling, and evaluation strategies (`evaluation_strategy="steps"`).



Upon completion of training, the fine-tuned model (`trainer.save_model(new_model)`) and tokenizer (`tokenizer.save_pretrained(new_model)`) are saved to disk for future use or deployment.

Finally, an example inference scenario is demonstrated using the `pipeline` function from `transformers`, allowing for quick generation of text based on user input. This showcases the practical application of the fine-tuned model in generating language responses.

Conclusion

We started a project where we used the "meta-llama/Llama-2-7b-chat-hf" model to build a custom chatbot. Our methodical approach involved preprocessing, dataset transformation, and laboriously fine-tuning a powerful transformer-based model. Using cutting-edge technologies and libraries, like Hugging Face's transformers, and datasets, we successfully developed a chatbot that can provide both coherent and contextually appropriate responses.

To create a reliable and effective solution, we overcame difficulties with data preparation, model construction, and CPU resource training along the way. We were able to maximize our chatbot's performance with the incorporation of sophisticated strategies, such as dynamic padding and particular model setups.

The deployment and engagement with our refined model have proven to be effective, demonstrating the ability of contemporary NLP frameworks to create personalized conversational agents for certain domains. This study not only exemplifies the usefulness of fine-tuning and transfer learning but also shows how these ideas may be used to create scalable and efficient NLP systems.

As a first step towards future improvements and applications across other domains, our findings validate the practicality of leveraging big language models for specific tasks. A strong basis for future study and advancement in the field of conversational AI is provided by the insights and techniques acquired throughout this project.