

# INTEL UNNATI INDUSTRIAL TRAINING PROGRAM – 2024



## A Report on

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

GitHub Repository: <https://github.com/JoyM268/Intel-Unnati-Industrial-Training-Program-2024>

Submitted By:

**Tech Wizards**



<b>Institute Name</b>	Shri Dharmasthala Manjunatheshwara College of Engineering and Technology		
<b>Team Members</b>	Joy Mascarenhas	<b>Faculty Mentor</b>	Prof. Pratap Kumar
	AbdulBasith A Mulla	<b>Industry Mentor</b>	Ms. Vasudha Kumari
	Kodeganti Bhanu Shankar	<b>External Mentor</b>	Mr. Abhishek Nandy

## **Problem Statement:**

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

## **Problem Description:**

This problem statement is designed to introduce beginners to the exciting field of Generative Artificial Intelligence (GenAI) through a series of hands-on exercises. Participants will learn the basics of GenAI, perform simple Large Language Model (LLM) inference on a CPU, and explore the process of fine-tuning an LLM model to create a custom Chatbot.

## **Major Challenges:**

1. Pre-trained Language Models can have large file sizes, which may require significant storage space and memory to load and run.
2. Learn LLM inference on CPU.
3. Understanding the concept of fine-tuning and its importance in customizing LLMs.
4. Create a Custom Chatbot with fine-tuned pre-trained Large Language Models (LLMs) using Intel AI Tools.

## **Tools:**

- **Intel Developer Cloud (IDC):** IDC is used for developing and deploying AI projects. This platform not only offers the advantage of high-performance GPU and enterprise-grade CPU but also provides access to the latest Intel hardware and software capabilities.
- **Intel Extension For Transformers:** Intel® Extension for Transformers is an innovative toolkit designed to accelerate GenAI/LLM everywhere with the optimal performance of Transformer-based models on various Intel platforms, including Intel Gaudi2, Intel CPU, and Intel GPU.
- **Hugging Face Transformer:** It APIs and tools to easily download and train state-of-the-art pretrained models.

- **Transformers Library:** The Transformers library is a popular Python library for working with state-of-the-art natural language processing (NLP) models, especially those based on the transformer architecture. It's developed by Hugging Face, a company focused on machine learning tools.
- **Llama2 Model:** LLaMA2 is a powerful large language model by Meta, excelling in tasks like text generation and code comprehension, with several sizes available for research and commercial use.
- **Kaggle:** Kaggle enables users to find and publish datasets, explore and build models in a web-based data science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges.
- **TinyLlama Model:** TinyLlama is a compact 1.1B language model pretrained on around 1 trillion tokens for approximately 3 epochs. It builds upon the architecture and tokenizer of Llama 2. The architecture of Llama 2 consists of 24 transformer layers with 16 attention heads and a hidden size of 307223. The tokenizer used is a byte pair encoding (BPE), allowing the model to handle rare or unknown words effectively.

## Building a Chatbot:

In the session 1, we were introduced to the fundamentals of Large Language Models (LLMs) and Transformers, key technologies powering modern AI applications. A practical demonstration involved using the Intel Extension for Transformers to build a chatbot capable of providing informative responses to user queries.

we were provided with a Jupyter Notebook ([build chatbot on spr.ipynb](#)) that facilitated the creation of an AI chatbot. We used 'intel-extension-for-transformers' library for creating the chatbot. First we installed 'intel-extension-for-transformers' library and then installed the necessary requirements given in the file [requirements\\_cpu.txt](#). The chatbot used intel's Neural Chat Model and gave the desired output on running it. We need to login into the Hugging Face using access tokens before executing the code.

The following code is used to create the chatbot:

```
from intel_extension_for_transformers.neural_chat import build_chatbot, PipelineConfig
from intel_extension_for_transformers.transformers import MixedPrecisionConfig

config = PipelineConfig(optimization_config=MixedPrecisionConfig())
chatbot = build_chatbot(config)
```

Once the chatbot is created we can prompt it as shown below:

```
response = chatbot.predict(query="Tell me about Intel Xeon Scalable Processors.")
print(response)
```

Which gives us the output:

The Intel Xeon Scalable Processors represent a family of high-performance central processing units (CPUs) designed for data centers, cloud computing, and other demanding workloads. These processors offer significant improvements in performance, efficiency, and scalability compared to their predecessors.

Key features include:

1. **Advanced architecture:** The Intel Xeon Scalable Processors utilize the latest microarchitecture, which enables better performance and power efficiency.
2. **Enhanced security:** With built-in hardware-level security features like Intel Software Guard Extensions (SGX), these processors provide robust protection against potential threats.
3. **Scalability:** The modular design allows for flexible configurations, enabling users to choose the right combination of cores, memory capacity, and I/O capabilities to meet specific needs.
4. **Optimized for virtualization:** The Intel Xeon Scalable Processors are optimized for virtual environments, supporting multiple operating systems and applications on a single server.
5. **High-speed connectivity:** Integrated support for high-speed interconnects such as Intel Omni-Path Architecture (OPA) and Intel QuickAssist Technology (QAT) ensures efficient communication between servers and devices

We experimented with different prompts to test the chatbot's capabilities:

### **Example 1:**

Prompt:

```
response1 = chatbot.predict(query="What is difference between GPU and CPU.")  
print(response1)
```

Output:

The main difference between a Graphics Processing Unit (GPU) and a Central Processing Unit (CPU) lies in their primary functions and architectures.

A CPU is the brain of a computer, responsible for executing instructions and managing various tasks. It's designed to handle general-purpose computing tasks like processing data, running programs, and controlling peripheral devices. CPUs have a single instruction stream and can perform multiple tasks simultaneously through multithreading or multiprocessing.

On the other hand, a GPU is specialized hardware optimized for handling complex mathematical calculations and rendering visuals. GPUs excel at parallel processing, which means they can execute many tasks simultaneously using thousands of small processing units called CUDA cores. This makes them ideal for graphics-intensive applications such as gaming, video editing, and scientific simulations.

In summary, while both CPUs and GPUs are essential components of modern computers, they serve different purposes. CPUs focus on general-purpose computing tasks, whereas GPUs specialize in high-performance graphics and parallel processing.

### **Example 2:**

Prompt:

```
response2 = chatbot.predict(query="Explain finetuning of a ai model.")  
print(response)
```

## Output:

Finetuning of an AI model is a process where you take an existing pre-trained model and further adapt it for a specific task or domain. This technique helps improve the performance of the model on new data sets or tasks without starting from scratch.

Imagine you have a smart robot that can perform various tasks like cooking, cleaning, and playing music. At first, this robot was designed to only cook. However, as time went by, you wanted it to learn how to clean your house too. Instead of building a completely new robot for cleaning, you decide to teach the existing cooking robot some additional skills. That's what we call finetuning - adapting the original model (robot) to perform better in a different area (cleaning).

To finetune an AI model, you typically follow these steps:

1. Choose a pre-trained model: Start with a model that has already been trained on a large dataset for a particular task. For example, if you want to create a model that can recognize images, you might use a pre-trained image recognition model.
2. Prepare the new dataset: Collect a new set of data related to the desired task or domain.

## Example 3:

### Prompt:

```
response4 = chatbot.predict(query="What is a Transformer")  
print(response4)
```

## Output:

A Transformer is a type of neural network architecture designed for natural language processing tasks. It was introduced in 2017 by researchers at Google AI. The model has shown impressive results in various NLP applications such as machine translation, text summarization, and question answering. Transformers learn to represent words and sentences through self-attention mechanisms, which allow them to understand the context and relationships between different parts of the input data. This makes them highly effective in understanding complex linguistic structures and generating meaningful outputs.

#### **Example 4:**

Prompt:

```
response3 = chatbot.predict(query="What are TPU, are they better than GPU")  
print(response3)
```

Output:

TPU stands for Tensor Processing Unit, which is a specialized hardware designed for machine learning tasks. It's optimized for running deep neural networks efficiently. GPUs (Graphics Processing Units) were initially developed for rendering graphics in video games but have since been repurposed for various computing tasks, including machine learning.

Comparing TPUs and GPUs can be complex as it depends on the specific application and workload. Generally, TPUs offer higher performance for certain machine learning tasks like training large models or performing inference at scale. However, GPUs still excel in other areas such as general-purpose computing, gaming, and handling diverse workloads. So, while TPUs may be better suited for some AI applications, GPUs remain versatile and valuable across multiple domains.

#### **Example 5:**

Prompt:

```
response5 = chatbot.predict(query="What is a AGI")  
print(response5)
```

Output:

A General Artificial Intelligence (AGI) refers to a hypothetical future AI system that can perform any intellectual task as well as or better than a human being. It would possess a high level of intelligence, creativity, problem-solving abilities, and adaptability across various domains. AGIs are still in the realm of science fiction, with current AI systems focusing on specific tasks and limited capabilities.

## Finetuning the Chatbot:

Fine-tuning is the process of taking a pretrained machine learning model and further training it on a smaller, targeted data set. The aim of fine-tuning is to maintain the original capabilities of a pretrained model while adapting it to suit more specialized use cases.

In the session 2, we were introduced to the concept of fine-tuning of Large Language Models (LLMs). A practical demonstration involved using the Intel Extension for Transformers to fine-tune the [Llama 2](#) model on the [Alpaca Dataset](#). We used [TinyLlama\\_v1.1](#) model instead of Llama2 model.

A Jupyter Notebook ([single\\_node\\_finetuning\\_on\\_spr.ipynb](#)) was provided that facilitated the fine-tuning of a ai model. Similar to the notebook in session 1 we used 'intel-extension-for-transformers' library and installed the requirements given in [requirements.txt](#).

**Dataset:** We are using the [Alpaca dataset](#) from Stanford University as the general domain dataset to fine-tune the model. This dataset is provided in the form of a JSON file, [alpaca\\_data.json](#). In Alpaca, researchers have manually crafted 175 seed tasks to guide text-davinci-003 in generating 52K instruction data for diverse tasks.

**Model:** [TinyLlama\\_v1.1](#) adopts exactly the same architecture and tokenizer as Llama 2. This means TinyLlama can be plugged and played in many open-source projects built upon Llama. Besides, TinyLlama is compact with only 1.1B parameters. This compactness allows it to cater to a multitude of applications demanding a restricted computation and memory footprint.

Before we fine-tune the TinyLlama\_v1.1 model on Alpaca dataset, we have to login into the Hugging Face using access tokens. We have to execute the following command:

```
huggingface-cli login
```

After executing the command enter the Hugging Face Access Token, the access token can be generated at the following link:

<https://huggingface.co/settings/tokens>



The following code is used to fine-tune the TinyLlama\_v1.1 model:

```
from transformers import TrainingArguments
from intel_extension_for_transformers.neural_chat.config import (
    ModelArguments,
    DataArguments,
    FinetuningArguments,
    TextGenerationFinetuningConfig,
)
from intel_extension_for_transformers.neural_chat.chatbot import finetune_model
model_args = ModelArguments(model_name_or_path="TinyLlama/TinyLlama_v1.1")
data_args = DataArguments(train_file="alpaca_data.json",
validation_split_percentage=1)
training_args = TrainingArguments(
    output_dir='./tmp',
    do_train=True,
    do_eval=True,
    num_train_epochs=3,
    overwrite_output_dir=True,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    gradient_accumulation_steps=2,
    save_strategy="no",
    log_level="info",
    save_total_limit=2,
    bf16=True,
)
finetune_args = FinetuningArguments()
finetune_cfg = TextGenerationFinetuningConfig(
    model_args=model_args,
    data_args=data_args,
    training_args=training_args,
    finetune_args=finetune_args,
)
finetune_model(finetune_cfg)
```

Fine-tuning the TinyLlama\_v1.1 model on the Alpaca dataset took approximately 8 hours as shown in Fig 1 and had a Training loss of 1.2865 as shown in Fig 2. The Eval Metrics are shown in Fig 3.

[19305/19305 8:03:20, Epoch 2/3]

**Fig 1: Training Time Taken**

Step	Training Loss				
		7000	1.286600		
500	1.362100	7500	1.289800		
1000	1.311200	8000	1.278600		
1500	1.291400	8500	1.273900	14000	1.289300
2000	1.305900	9000	1.270700	14500	1.281300
2500	1.297400	9500	1.287000	15000	1.275600
3000	1.307700	10000	1.278500	15500	1.282300
3500	1.303200	10500	1.281300	16000	1.288200
4000	1.289000	11000	1.294800	16500	1.291100
4500	1.288000	11500	1.303800	17000	1.271300
5000	1.290300	12000	1.286300	17500	1.289600
5500	1.285500	12500	1.279800	18000	1.293200
6000	1.290600	13000	1.286300	18500	1.296000
6500	1.292900	13500	1.281800	19000	1.286500

**Fig 2:** Training Loss at each step.

```

***** eval metrics *****
epoch                =      2.9998
eval_loss             =      1.2866
eval_ppl              =      3.6205
eval_runtime          = 0:00:43.64
eval_samples          =      520
eval_samples_per_second =     11.914
eval_steps_per_second  =      2.978

```

**Fig 3:** Eval Metrics.

## Project (Finance Chatbot):

Leveraging the knowledge gained from the two sessions conducted by the Intel Unnati team, and through the execution of the provided Jupyter Notebooks, we have developed a another chatbot. This chatbot has been fine-tuned on the [finance alpaca.json](#) dataset, specifically designed to address user inquiries which are finance related. The underlying architecture utilizes the TinyLlama pre-trained model. The finetuning of the model took approximately one and half hour as shown in Fig 4 and the training loss was 0.829500 as Shown in Fig 5. The eval metrics are shown in Fig 6. The code and its output are present in the Jupyter Notebook [finance\\_chatbot.ipynb](#).

[1401/1401 1:23:57, Epoch 3/3]

**Fig 4: Training Time.**

Step	Training Loss
500	0.851400
1000	0.829500

**Fig 5: Training Loss at each step.**

```
**** eval metrics ****
epoch                =      3.0
eval_loss            =      1.0709
eval_ppl             =      2.918
eval_runtime         = 0:00:08.20
eval_samples         =      38
eval_samples_per_second =      4.629
eval_steps_per_second  =      1.218
```

**Fig 6: Eval Metrics.**

For prompt:

what is meant by cost of equity?

We get the response:

Cost of equity is a financial metric used to evaluate the cost of capital for a company. It is calculated by dividing the total return on equity (ROE) by the cost of equity (COE). The COE is a measure of the cost of capital for a company, which is the amount of money required to finance the company's operations.

The cost of equity is important for investors because it provides a measure of the risk associated with investing in a company. It takes into account the cost of capital, which includes interest rates, taxes, and other expenses associated with investing in the company. By comparing the cost of equity to the ROE, investors can determine whether the company's investment is profitable or not.

**Video Link:**

[https://drive.google.com/file/d/1y0zHbecpfFLH25lg24\\_Wu-sAX7mu-kjY/view?usp=sharing](https://drive.google.com/file/d/1y0zHbecpfFLH25lg24_Wu-sAX7mu-kjY/view?usp=sharing)

**Conclusion:**

The program and its conducted sessions proved highly beneficial in imparting valuable insights into the creation of chatbots, model fine-tuning, and other AI-related topics. These sessions have effectively initiated our journey into the expansive realm of generative AI and machine learning.