Intel® Unnati
Data-Centric Labs in Emerging Technologies

Project Report on

Introduction to GenAI and Simple LLM Inference on CPU and fine-tuning of LLM Model to create a Custom Chatbot

Submitted for the Intel Unnati Industrial Training Program 2024

Team Tech Crafters

Aneesh G B (1NT22EC016)
Auchitya Jain (1NT22EC029)
Harshitha M P (1NT22EC061)
Kruthi Sridhar (1NT22EC081)

Under the Guidance of

Dr. Rajesh N
Professor
Dept. of Electronics and Communication Engineering

NITTE EDUCATION TRUST | NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

YELAHANKA, BENGALURU- 560064

Contents

Introduction

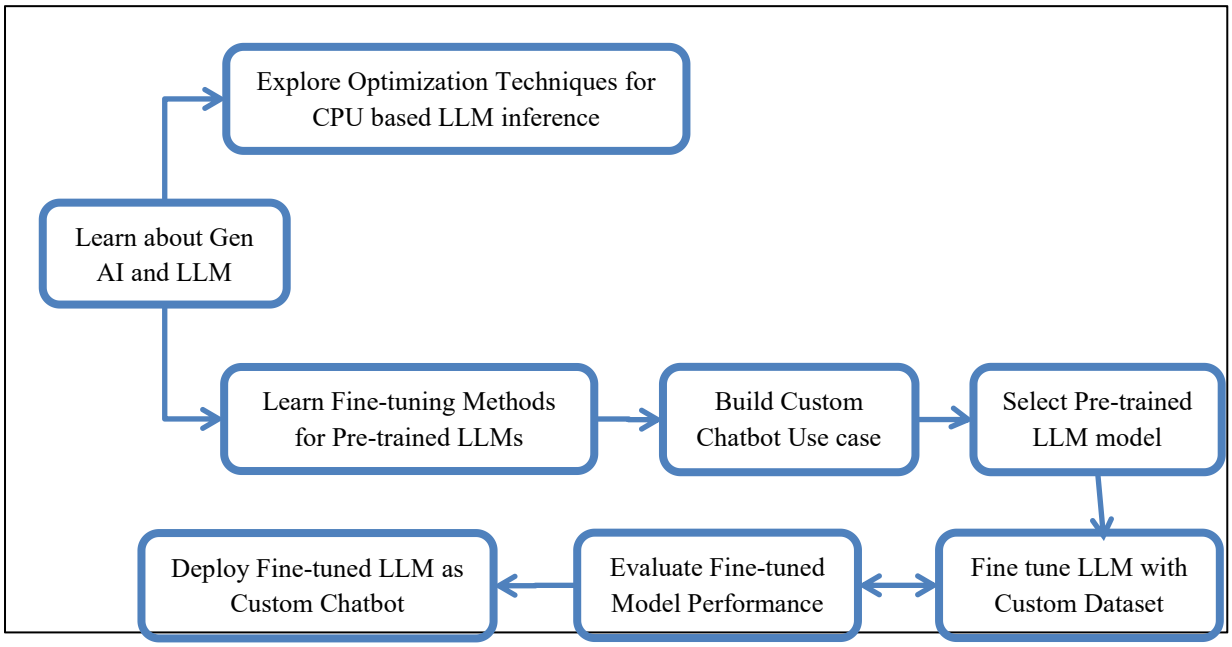
Generative Artificial Intelligence (Gen AI) refers to the use of AI to generate new content like images,text, videos, often in response to prompts. Generative AI model analyzes existing data to learn patterns and use that knowledge to generate novel content. Its application includes image generation, music composition, and code creation.

Large Language Models (LLMs) are complex algorithms trained on enormous datasets of text and code. This training data can include books, articles, code repositories, websites, and even social media conversations.By analyzing these vast amounts of text, LLMs learn the patterns and relationships between words, allowing them to understand the context and meaning of language.

Gen AI technology fuels Large Language Model (LLM) chatbots, which are trained on massive datasets of text to understand human language and respond in a natural way. By fine-tuning LLMs with GenAI, we can create custom chatbots tailored to specific purposes, enabling them to act as expert virtual assistants or informative customer service representatives. This powerful combination unlocks a vast potential for applications that can transform communication and content creation across various industries.

Building upon the foundation of Generative Artificial Intelligence (GenAI) and Large Language Models (LLMs), this project aimed to push the boundaries of accessibility and functionality. While LLMs power the impressive content creation capabilities of GenAI, their typical reliance on powerful GPUs can be a limiting factor. Our team tackled this challenge head-on, mastering optimization techniques that enabled successful LLM inference on CPUs. This breakthrough expands the reach of GenAI by making it attainable on less powerful hardware.

Proposed Architecture



Software Description

IDE: Intel Developer Cloud

Dataset: Alpaca Dataset from Standford University as the general domain dataset.

Model: Intel® Extension for Transformer's Neural Chat, Llama 2

Language: Python

Technology: Generative AI

Intel Libraries Used

Intel® Extension for Transformers (Intel® IET) is a toolkit designed to accelerate the performance of Transformer-based models on Intel platforms, particularly for servers running on Xeon Scalable processors and Habana Gaudi accelerators.

- **Extends Hugging Face Transformers API:** Leverages existing popular APIs for Transformer models, minimizing code changes for developers familiar with these frameworks.
- **Model Compression Techniques:** Integrates with Intel® Neural Compressor to offer various model compression techniques like quantization and pruning. This allows for smaller model sizes without sacrificing accuracy, leading to faster inference and deployment on resource-constrained devices.

Intel® Extension for Transformers empowers developers to:

- Deploy Transformer models efficiently on Intel platforms.
- Reduce model size and improve inference speed through compression techniques.
- Simplify chatbot development and deployment with the NeuralChat framework.
- Leverage optimized packages and workflows for common tasks.

Dataset Description

The Alpaca dataset refers to a dataset of instructions and demonstrations used for fine-tuning language models in the area of instruction following.It contains 52,000 instruction-demonstration pairs which are likely text-based, describing a specific task. It is used to fine-tune language models to better understand and follow instructions. Aims to improve the capability of LLMs to perform actions in the real world based on textual instructions.

Project Details

Steps followed

- ✓ Created an environment

  !python -m venv intel1

- ✓ Cloned the intel repository
- ✓ Ran the requirement files
- ✓ Executed the file

Github Link: https://github.com/AneeshGB/Intel-Unnati-Industrial-Training-2024

1. build_chatbot_on_spr.ipynb:

```
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)
response = chatbot.predict(query="tell me about intel transformer extensions")
print(response)
```

The above code snippet uses the Intel® Extension for Transformers (Intel® IET) to create a chatbot and query it.

Importing Libraries:

- `from intel_extension_for_transformers.neural_chat import build_chatbot, PipelineConfig`: This line imports the `build_chatbot` function and `PipelineConfig` class from the `neural_chat` module within the `intel_extension_for_transformers` library.
- `from intel_extension_for_transformers.transformers import MixedPrecisionConfig`: This line imports the `MixedPrecisionConfig` class from the `transformers` submodule within the `intel_extension_for_transformers` library.

Configuring Mixed Precision:

- `config = PipelineConfig(optimization_config = MixedPrecisionConfig())`: This line creates a `PipelineConfig` object named `config`. It sets the `optimization_config` attribute of this object to a new `MixedPrecisionConfig` object. This enables mixed precision training, which can potentially improve performance and reduce memory usage during chatbot training.

Building the Chatbot:

- `chatbot = build_chatbot(config)`: This line calls the `build_chatbot` function, passing the `config` object as an argument. This function creates a chatbot instance based on the provided configuration.

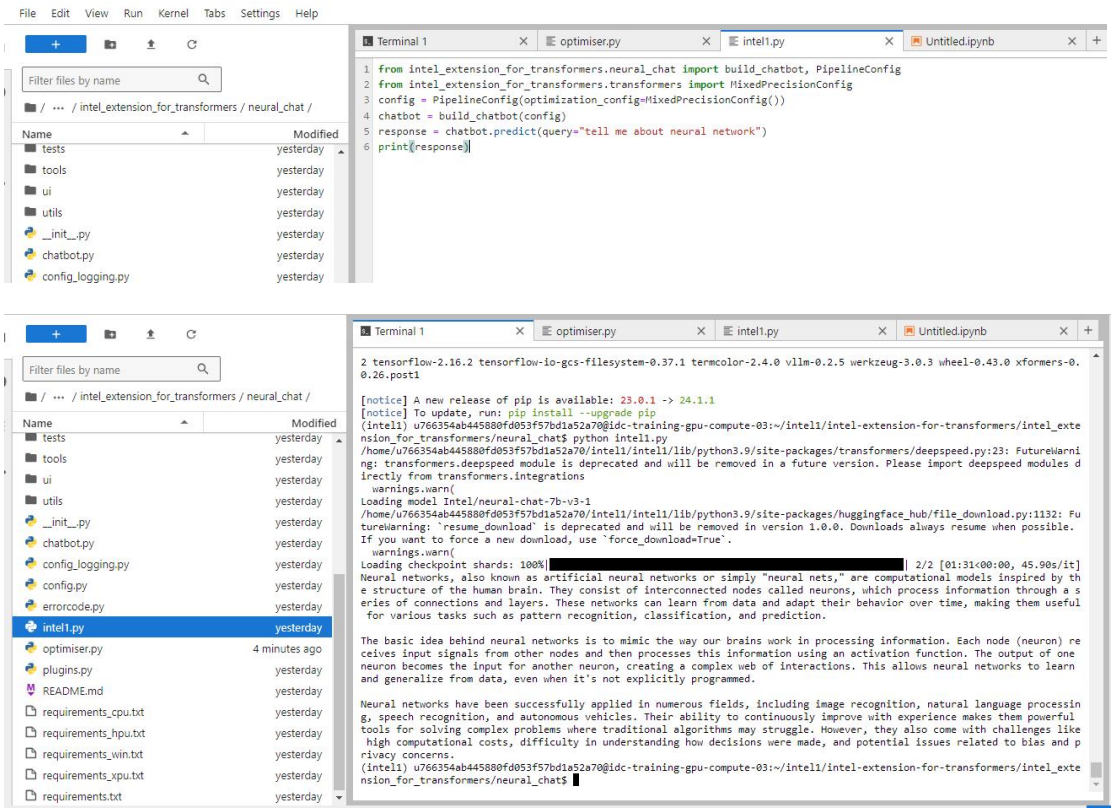
Querying the Chatbot:

- `response = chatbot.predict(query="tell me about intel transformer extensions")`: This line calls the `predict` method on the `chatbot` object. It passes a query string "tell me about intel transformer extensions" as an argument. The `predict` method queries the underlying LLM (Large Language Model) and returns a response based on its knowledge and training data.

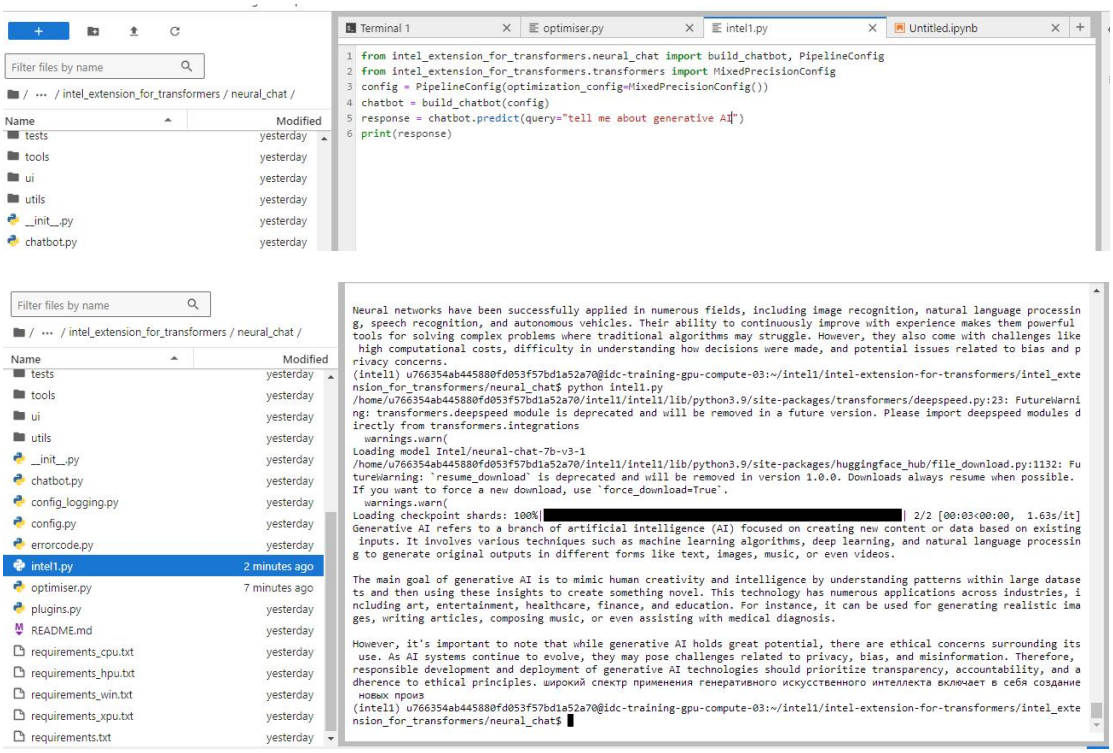
Printing the Response:

- `print(response)`: This line simply prints the response received from the chatbot to the console.

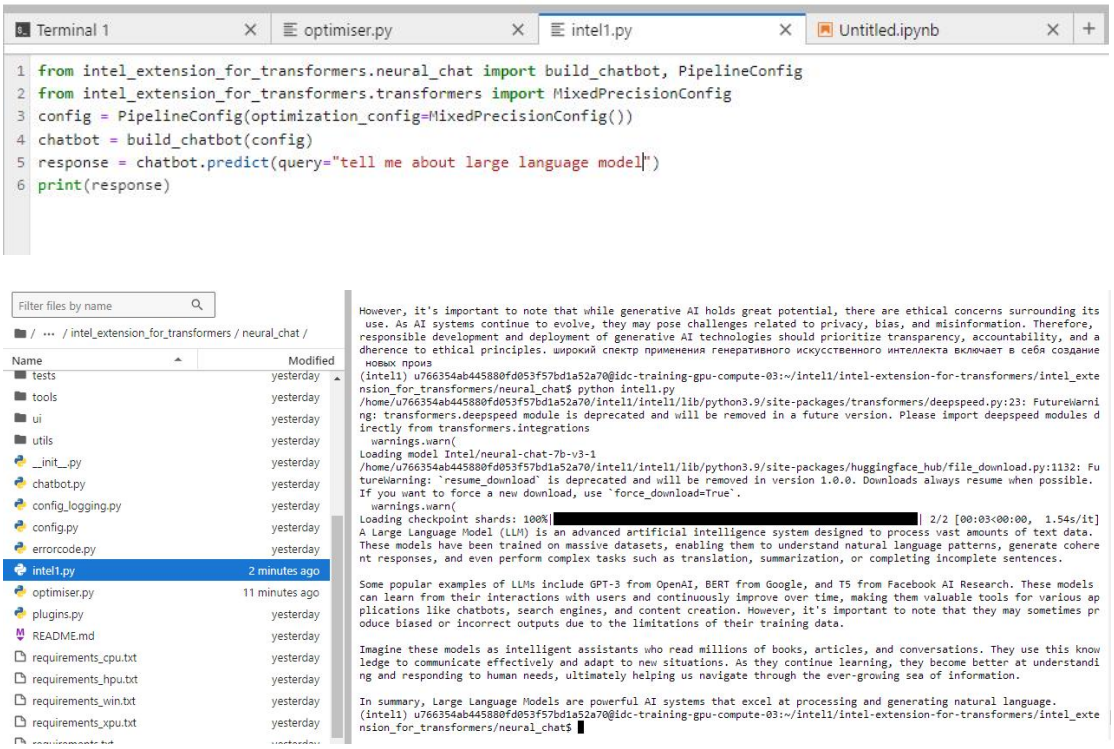
#1 Output



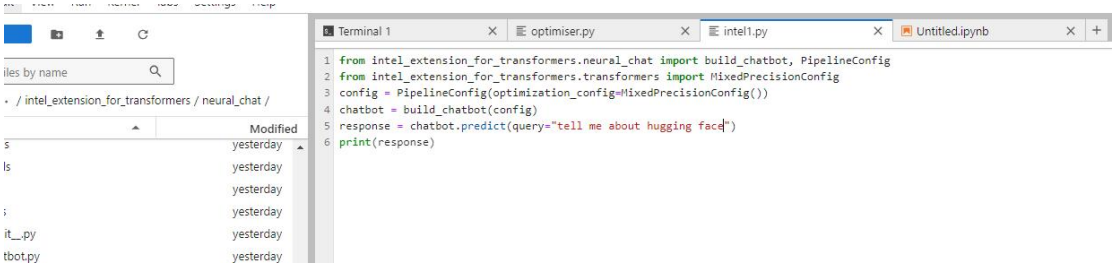
#2 Output



#3 Output

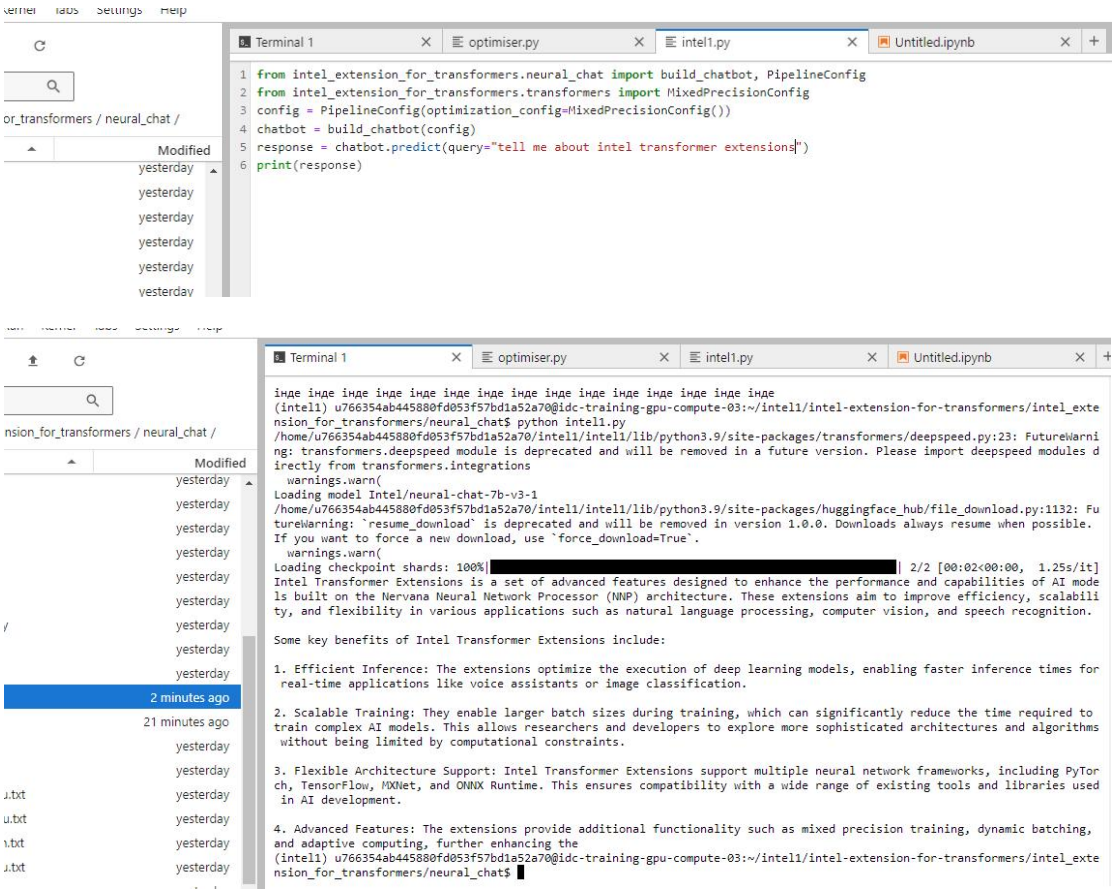


#4 Output




```
(intel1) u766354ab445880fd053f57bd1a52a70@idc-training-gpu-compute-03:~/intel1/intel-extension-for-transformers/intel_exte
nsion_for_transformers/neural_chat$ python intel1.py
/home/u766354ab445880fd053f57bd1a52a70/intel1/intel1/lib/python3.9/site-packages/transformers/deepspeed.py:23: FutureWarni
ng: transformers.deepspeed module is deprecated and will be removed in a future version. Please import deepspeed modules d
irectly from transformers.integrations
  warnings.warn(
Loading model Intel/neural-chat-7b-v3-1
/home/u766354ab445880fd053f57bd1a52a70/intel1/intel1/lib/python3.9/site-packages/huggingface_hub/file_download.py:1132: Fu
tureWarning: `resume_download` is deprecated and will be removed in version 1.0.0. Downloads always resume when possible.
If you want to force a new download, use `force_download=True`.
  warnings.warn(
Loading checkpoint shards: 100%|████████████████████████████████████████████| 2/2 [00:02<00:00,  1.27s/it]
Hugging Face is not just a place where people embrace each other; it's actually a popular platform for natural language pr
ocessing (NLP) researchers and developers. It provides various tools and resources to facilitate the creation of advanced
NLP models and applications. The name "Hugging Face" was inspired by the idea of bringing together different communities i
n the field of artificial intelligence to collaborate and share their knowledge. So, imagine a big virtual gathering of AI
 enthusiasts working together to create better ways for machines to understand human languages. инде инде инде инде инде и
нде инде инде инде инде инде инде инде инде инде инде инде инде инде инде инде инде инде инде инде инде инде инде инде инд
е инде инде инде инде инде инде инде инде инде инде инде инде инде инде инде инде инде инде инде инде инде инде инде инде
инде инде инде инде инде инде инде инде инде инде инде инде инде
(intel1) u766354ab445880fd053f57bd1a52a70@idc-training-gpu-compute-03:~/intel1/intel-extension-for-transformers/intel_exte
nsion_for_transformers/neural_chat$
```

#5 Output




```
инде инде инде инде инде инде инде инде инде инде инде инде инде инде
(intel1) u766354ab445880fd053f57bd1a52a70@idc-training-gpu-compute-03:~/intel1/intel-extension-for-transformers/intel_exte
nsion_for_transformers/neural_chat$ python intel1.py
/home/u766354ab445880fd053f57bd1a52a70/intel1/intel1/lib/python3.9/site-packages/transformers/deepspeed.py:23: FutureWarni
ng: transformers.deepspeed module is deprecated and will be removed in a future version. Please import deepspeed modules d
irectly from transformers.integrations
  warnings.warn(
Loading model Intel/neural-chat-7b-v3-1
/home/u766354ab445880fd053f57bd1a52a70/intel1/intel1/lib/python3.9/site-packages/huggingface_hub/file_download.py:1132: Fu
tureWarning: `resume_download` is deprecated and will be removed in version 1.0.0. Downloads always resume when possible.
If you want to force a new download, use `force_download=True`.
  warnings.warn(
Loading checkpoint shards: 100%|████████████████████████████████████████████| 2/2 [00:02<00:00,  1.25s/it]
Intel Transformer Extensions is a set of advanced features designed to enhance the performance and capabilities of AI mode
ls built on the Nervana Neural Network Processor (NNP) architecture. These extensions aim to improve efficiency, scalabili
ty, and flexibility in various applications such as natural language processing, computer vision, and speech recognition.

Some key benefits of Intel Transformer Extensions include:

1. Efficient Inference: The extensions optimize the execution of deep learning models, enabling faster inference times for
 real-time applications like voice assistants or image classification.

2. Scalable Training: They enable larger batch sizes during training, which can significantly reduce the time required to
train complex AI models. This allows researchers and developers to explore more sophisticated architectures and algorithms
 without being limited by computational constraints.

3. Flexible Architecture Support: Intel Transformer Extensions support multiple neural network frameworks, including PyTor
ch, TensorFlow, MXNet, and ONNX Runtime. This ensures compatibility with a wide range of existing tools and libraries used
 in AI development.

4. Advanced Features: The extensions provide additional functionality such as mixed precision training, dynamic batching,
and adaptive computing, further enhancing the
(intel1) u766354ab445880fd053f57bd1a52a70@idc-training-gpu-compute-03:~/intel1/intel-extension-for-transformers/intel_exte
nsion_for_transformers/neural_chat$
```

2. single_node_finetuning_on_spr.ipynb:

```
1  from transformers import AutoTokenizer, AutoModelForSequenceClassification, Trainer,TrainingArguments
2  from intel_extension_for_transformers.neural_chat.config import (
3      ModelArguments,
4      DataArguments,
5      FinetuningArguments,
6      TextGenerationFinetuningConfig,
7  )
8  from intel_extension_for_transformers.neural_chat.chatbot import finetune_model
9  model_args = ModelArguments(model_name_or_path="meta-llama/Llama-2-7b-chat-hf")
10 data_args = DataArguments(train_file="alpaca_data.json", validation_split_percentage=1)
11 training_args = TrainingArguments(
12     output_dir='./tmp2',
13     do_train=True,
14     do_eval=True,
15     num_train_epochs=2,
16     overwrite_output_dir=True,
17     per_device_train_batch_size=4,
18     per_device_eval_batch_size=4,
19     gradient_accumulation_steps=2,
20     save_strategy="no",
21     log_level="info",
22     save_total_limit=2,
23     bf16=True,
24 )
25 finetune_args = FinetuningArguments()
26 finetune_cfg = TextGenerationFinetuningConfig(
27             model_args=model_args,
28             data_args=data_args,
29             training_args=training_args,
30             finetune_args=finetune_args,
31         )
32 finetune_model(finetune_cfg)
```

The code above demonstrates how to fine-tune a large language model (LLM) for chatbot conversation using the Intel® Extension for Transformers (Intel® IET) `neural_chat` module.

Importing Libraries:

- Lines 1-3: Import necessary classes from the `transformers` library for working with pre-trained models and training processes.
- Lines 5-9: Import specific classes from the `intel_extension_for_transformers.neural_chat.config` submodule. These classes define various configurations for model arguments, data arguments, fine-tuning arguments, and text generation fine-tuning specifically.
- Line 11: Imports the `finetune_model` function from `intel_extension_for_transformers.neural_chat.chatbot`. This function is responsible for the entire fine-tuning process.

Configuration:

- Lines 13-14: Defines `model_args` using the `ModelArguments` class. This specifies the pre-trained model used for fine-tuning. Here, it is set to "meta-llama/Llama-2-7b-chat-hf", which is a 7-billion parameter LLM from the Meta Llama family.
- Lines 16-17: Defines `data_args` using the `DataArguments` class. This specifies the training data file ("alpaca_data.json") and sets the validation split.
- Lines 19-38: Define `training_args` using the `TrainingArguments` class. This configures various aspects of the training process:

    - `output_dir`: Store training outputs to the specified path
    - `do_train` and `do_eval`: Set to True for training and evaluation.
    - `num_train_epochs`: Number of training epochs
    - `overwrite_output_dir`: Allows overwriting existing output directory.
    - `per_device_train_batch_size` and `per_device_eval_batch_size`: Batch size for training and evaluation.
    - `gradient_accumulation_steps`: Accumulate gradients across multiple batches before updating weights.
    - `save_strategy`: Set to "no" to not save intermediate checkpoints during training.
    - `log_level`: Set to "info" for informational logging.
    - `save_total_limit`: Maximum number of checkpoints to save
    - `bf16`: Enable mixed precision training using BF16 data type (potentially improves performance).

- Lines 39-40: Defines `finetune_args` using the `FinetuningArguments` class.
- Lines 42-43: Combines all configurations into a single `TextGenerationFinetuningConfig` object named `finetune_cfg`.

Fine-tuning the Model:

- Line 45: Calls the `finetune_model` function with the `finetune_cfg` object as an argument. This function performs the following steps:

    - Loads the pre-trained LLM specified in `model_args`.
    - Loads the training data from "alpaca_data.json" based on `data_args`.
    - Performs fine-tuning on the training data using the configurations specified in `training_args`.
    - Evaluates the fine-tuned model based on `training_args`.

The alpaca dataset is reduced to 1980 samples as the 52k sample dataset was taking lot of time to run and server was terminating in the process.

with Epoch=1

```
trainable params: 4,194,304 || all params: 6,742,609,920 || trainable%: 0.06220594176090199
[INFO|trainer.py:641] 2024-07-14 01:53:23,396 >> Using cpu_amp half precision backend
[INFO|trainer.py:2078] 2024-07-14 01:53:23,687 >> ***** Running training *****
[INFO|trainer.py:2079] 2024-07-14 01:53:23,687 >>   Num examples = 1,980
[INFO|trainer.py:2080] 2024-07-14 01:53:23,687 >>   Num Epochs = 1
[INFO|trainer.py:2081] 2024-07-14 01:53:23,687 >>   Instantaneous batch size per device = 4
[INFO|trainer.py:2084] 2024-07-14 01:53:23,687 >>   Total train batch size (w. parallel, distributed & accumulation) = 8
[INFO|trainer.py:2085] 2024-07-14 01:53:23,687 >>   Gradient Accumulation steps = 2
[INFO|trainer.py:2086] 2024-07-14 01:53:23,687 >>   Total optimization steps = 247
[INFO|trainer.py:2087] 2024-07-14 01:53:23,689 >>   Number of trainable parameters = 4,194,304
  0%|                                                                                                | 1/247 [00:25<1:45:03, 25.62s/it]

[INFO|trainer.py:2081] 2024-07-14 01:53:23,687 >>   Instantaneous batch size per device = 4
[INFO|trainer.py:2084] 2024-07-14 01:53:23,687 >>   Total train batch size (w. parallel, distributed & accumulation) = 8
[INFO|trainer.py:2085] 2024-07-14 01:53:23,687 >>   Gradient Accumulation steps = 2
[INFO|trainer.py:2086] 2024-07-14 01:53:23,687 >>   Total optimization steps = 247
[INFO|trainer.py:2087] 2024-07-14 01:53:23,689 >>   Number of trainable parameters = 4,194,304
100%|████████████████████████████████████████████████████████████████████████████████████████| 247/247 [16:24<00:00,  3.72s/it]
[INFO|trainer.py:2329] 2024-07-14 02:09:47,864 >>

Training completed. Do not forget to share your model on huggingface.co/models =)


{'train_runtime': 984.1753, 'train_samples_per_second': 2.012, 'train_steps_per_second': 0.251, 'train_loss': 1.1783620239752024, 'epoch': 1.0}
100%|████████████████████████████████████████████████████████████████████████████████████████| 247/247 [16:24<00:00,  3.98s/it]
[INFO|trainer.py:3410] 2024-07-14 02:09:47,868 >> Saving model checkpoint to ./
Repo card metadata block was not found. Setting CardData to empty.
2024-07-14 02:09:47,869 - repocard.py - huggingface_hub.repocard - WARNING - Repo card metadata block was not found. Setting CardData to empty.
[INFO|tokenization_utils_base.py:2513] 2024-07-14 02:09:47,891 >> tokenizer config file saved in ./tokenizer_config.json
[INFO|tokenization_utils_base.py:2522] 2024-07-14 02:09:47,893 >> Special tokens file saved in ./special_tokens_map.json
2024-07-14 02:09:47,910 - finetuning.py - intel_extension_for_transformers.transformers.llm.finetuning.finetuning - INFO - *** Evaluate After T
raining***
[INFO|trainer.py:3719] 2024-07-14 02:09:47,920 >> ***** Running Evaluation *****
[INFO|trainer.py:3721] 2024-07-14 02:09:47,920 >>   Num examples = 20
[INFO|trainer.py:3724] 2024-07-14 02:09:47,920 >>   Batch size = 4
100%|████████████████████████████████████████████████████████████████████████████████████████| 5/5 [00:02<00:00,  1.86it/s]
***** eval metrics *****
  epoch                   =      0.998
  eval_loss               =     1.1051
  eval_ppl                =     3.0195
  eval_runtime            = 0:00:03.29
  eval_samples            =         20
  eval_samples_per_second =      6.073
  eval_steps_per_second   =      1.518
(intel1) u766354ab445880fd053f57bd1a52a70@idc-training-gpu-compute-06:~/intel1/intel-extension-for-transformers/intel_extension_for_transformer
s/neural_chat$
```

With Epoch=2

```
llator of type DataCollatorForSeq2Seq
trainable params: 4,194,304 || all params: 6,742,609,920 || trainable%: 0.06220594176090199
[INFO|trainer.py:641] 2024-07-14 02:30:13,564 >> Using cpu_amp half precision backend
[INFO|trainer.py:2078] 2024-07-14 02:30:13,810 >> ***** Running training *****
[INFO|trainer.py:2079] 2024-07-14 02:30:13,810 >>   Num examples = 1,980
[INFO|trainer.py:2080] 2024-07-14 02:30:13,810 >>   Num Epochs = 2
[INFO|trainer.py:2081] 2024-07-14 02:30:13,810 >>   Instantaneous batch size per device = 4
[INFO|trainer.py:2084] 2024-07-14 02:30:13,810 >>   Total train batch size (w. parallel, distributed & accumulation) = 8
[INFO|trainer.py:2085] 2024-07-14 02:30:13,810 >>   Gradient Accumulation steps = 2
[INFO|trainer.py:2086] 2024-07-14 02:30:13,810 >>   Total optimization steps = 494
[INFO|trainer.py:2087] 2024-07-14 02:30:13,812 >>   Number of trainable parameters = 4,194,304
  0%|                                                                                                | 0/494 [00:00<?, ?it/s]

[INFO|trainer.py:2080] 2024-07-14 02:30:13,810 >>   Num Epochs = 2
[INFO|trainer.py:2081] 2024-07-14 02:30:13,810 >>   Instantaneous batch size per device = 4
[INFO|trainer.py:2084] 2024-07-14 02:30:13,810 >>   Total train batch size (w. parallel, distributed & accumulation) = 8
[INFO|trainer.py:2085] 2024-07-14 02:30:13,810 >>   Gradient Accumulation steps = 2
[INFO|trainer.py:2086] 2024-07-14 02:30:13,810 >>   Total optimization steps = 494
[INFO|trainer.py:2087] 2024-07-14 02:30:13,812 >>   Number of trainable parameters = 4,194,304
100%|████████████████████████████████████████████████████████████████████████████████████████| 494/494 [31:23<00:00,  3.19s/it]
[INFO|trainer.py:2329] 2024-07-14 03:01:37,081 >>

Training completed. Do not forget to share your model on huggingface.co/models =)


{'train_runtime': 1883.2694, 'train_samples_per_second': 2.103, 'train_steps_per_second': 0.262, 'train_loss': 1.1319538070122723, 'epoc
h': 2.0}
100%|████████████████████████████████████████████████████████████████████████████████████████| 494/494 [31:23<00:00,  3.81s/it]
[INFO|trainer.py:3410] 2024-07-14 03:01:37,085 >> Saving model checkpoint to ./temp
[INFO|tokenization_utils_base.py:2513] 2024-07-14 03:01:37,120 >> tokenizer config file saved in ./temp/tokenizer_config.json
[INFO|tokenization_utils_base.py:2522] 2024-07-14 03:01:37,122 >> Special tokens file saved in ./temp/special_tokens_map.json
2024-07-14 03:01:37,139 - finetuning.py - intel_extension_for_transformers.transformers.llm.finetuning.finetuning - INFO - *** Evaluate
After Training***
[INFO|trainer.py:3719] 2024-07-14 03:01:37,146 >> ***** Running Evaluation *****
[INFO|trainer.py:3721] 2024-07-14 03:01:37,146 >>   Num examples = 20
[INFO|trainer.py:3724] 2024-07-14 03:01:37,146 >>   Batch size = 4
100%|████████████████████████████████████████████████████████████████████████████████████████| 5/5 [00:02<00:00,  2.01it/s]
***** eval metrics *****
  epoch                   =      1.996
  eval_loss               =     1.0847
  eval_ppl                =     2.9586
  eval_runtime            = 0:00:03.11
  eval_samples            =         20
  eval_samples_per_second =       6.43
  eval_steps_per_second   =      1.607
(intel1) u766354ab445880fd053f57bd1a52a70@idc-training-gpu-compute-06:~/intel1/intel-extension-for-transformers/intel_extension_for_tran
sformers/neural_chat$
```

Conclusion

In conclusion, our exploration into the world of Generative AI (GenAI) and Large Language Models (LLMs) has been both enlightening and empowering, particularly from a student's perspective. The journey began with a deep dive into the foundational concepts of GenAI, providing us with a clear understanding of its capabilities and the revolutionary impact it can have across various fields.

By implementing simple LLM inference on a CPU, we discovered that cutting-edge AI technologies are more accessible than ever. This hands-on experience demonstrated that high-performance AI applications are not restricted to those with advanced hardware, opening doors for students and enthusiasts to experiment and innovate using their existing resources.

The process of fine-tuning an LLM to create a custom chatbot was particularly exciting. It allowed us to see firsthand how these models can be adapted to meet specific needs and contexts, making them more relevant and effective. This practical application of fine-tuning techniques underscored the importance of personalization in AI, showing how we can tailor responses to enhance user interaction and satisfaction.

Throughout this project, we learned valuable lessons about the practical steps and considerations involved in developing intelligent and responsive chatbots. The experience not only broadened our technical skills but also sparked a deeper appreciation for the potential of AI in creating impactful solutions.

Overall, this journey has been a testament to the transformative power of AI and the opportunities it offers for innovation. As students, we are now better equipped with the knowledge and skills to leverage AI's capabilities, paving the way for future projects and discoveries. This exploration has laid a solid foundation for our continued learning and development in the ever-evolving field of artificial intelligence.