**Intel Unnati Industrial Training Report**

Date: 1st July 2024

**Problem Statement**: Running GenAI on Intel AI Laptops and Simple LLM Inference on CPU and fine-tuning of LLM Models using Intel® OpenVINO™.

**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.

**Solution approach**:

Since intel openVINO is an open-source toolkit used for accelerating, optimizing and deploying LLMs (Large Language Models) on the edge and on the cloud irrespective of the hardware present on the computing machines.

It also allows very large language models which require a lot of memory and space to be run locally on any intel machine. Since the problem was to run an LLM on the edge then what better way to do that than to run a chatbot locally to answer all your questions. So we will select an efficient model and fine-tune it and then run inference locally.

**Step 1**: We need to select a Generative Pre-Training model which is already available on the huggingface hub. While selecting a precise model we also need to search for an efficient model which occupies less memory and space and can run on our machine.

Since we will be using NNCF, Neural Network Compression Function, which is responsible for quantization of the model

We have selected the gpt2 with INT4 precision with the task of text-generation

**Step 2**: We need to download the model first in the pytorch form factor and then convert it into an Intermediate representation (IR) that openVINO can understand which is basically creating an .xml file and a .bin file along with the config.json which contains the backbone of the model describing what are the different parameters that keeps the model running

**Step 3**: Now we fine-tune the model by exporting the model and applying weight compression using the command:

!optimum-cli export openvino --model gpt2  --weight-format int4 ov\_model\_dir

There is a certain need for reduction of weights because the chatbots that are deployed currently are using cloud high end resources for computation and while we are trying to run the model on the edge we will need to reduce the weights so that we conserve resources for other system activities as well

While weight compression is a great way to optimize model footprint but the higher compression we do the accuracy is sacrificed

**Step 4**: Now using a hugging face pipeline we get out model from the directory in which the .xml and the .bin files were stored specify the task of text generation and mention the model arguments such as:

"KV\_CACHE\_PRECISION": "u8",

"DYNAMIC\_QUANTIZATION\_GROUP\_SIZE": "32",

"PERFORMANCE\_HINT": "LATENCY",

"NUM\_STREAMS": "1",

"CACHE\_DIR": ""

1. **KV\_CACHE\_PRECISION**: This parameter specifies the precision of the key-value cache used during inference. In this case, "u8" indicates that the cache should use 8-bit unsigned integer precision.
   * **Purpose**: Reducing precision (e.g., from 32-bit floating-point to 8-bit integer) helps in lowering memory usage and can speed up the inference process, although there might be a slight trade-off in accuracy.
2. **DYNAMIC\_QUANTIZATION\_GROUP\_SIZE**: This parameter defines the group size for dynamic quantization.
   * **Purpose**: Dynamic quantization converts weights and activations to a lower precision on-the-fly during inference. Setting a group size helps control the granularity of this quantization. Smaller groups can lead to finer quantization but might require more computational resources.
3. **PERFORMANCE\_HINT**: This parameter provides a high-level performance hint to the inference engine. Common values include "LATENCY", "THROUGHPUT", and "BALANCED".
   * **LATENCY**: Optimizes for the lowest possible latency, i.e., the time it takes to process a single inference request.
   * **Purpose**: This is useful for applications where quick response times are critical, such as real-time systems or interactive applications.
4. **NUM\_STREAMS**: This parameter sets the number of inference streams (or parallel inference requests) that can be processed simultaneously.
   * **Purpose**: Adjusting the number of streams can help balance the workload and improve overall throughput. For example, setting "NUM\_STREAMS": "1" means only one inference request will be processed at a time, which can be optimal for latency-sensitive applications but might not fully utilize the available computational resources.
5. **CACHE\_DIR**: This parameter specifies the directory where the model cache is stored.
   * **Purpose**: Caching compiled models can significantly speed up subsequent inferences by avoiding the need to recompile the model each time it is loaded. Setting "CACHE\_DIR": "" indicates that no caching directory is specified, and caching might be disabled.

**Step 5**: Now using gradio interface we get the input generate the output and send it to the webpage starting a conversation here we have made a function which conserves the history of a conversation and this is helpful to retain context while answering a query.

* Why use **gpt2** model?

The reason why we are going with gpt2 because it is the state-of-the-art language model developed by OpenAI which has parameters anywhere between 124M and 1.5B. GPT-2 comes in various sizes with different numbers of layers (transformer blocks), hidden units, and attention heads.

The input to GPT-2 is a sequence of tokens, which are numerical representations of words or subwords.

The output is also a sequence of tokens, which can be converted back to text using the tokenizer.

**Limitations**

1. **Bias and Ethical Concerns**:
   * Like many large language models, GPT-2 can produce biased or harmful content, reflecting the biases present in the training data.
   * It is important to use such models responsibly and be aware of their limitations.
2. **Resource Intensive**:
   * Running GPT-2, especially the larger versions, requires significant computational resources, including powerful GPUs and a substantial amount of memory

**Diagrammatic approach**

A diagram of a company

Description automatically generated

A diagram of a flowchart

Description automatically generated

**Code contribution:**

Complete code has been written by Aakash Gangurde as I am the only person in my team and there have been instances where I have taken help from the [github repositories](https://github.com/openvinotoolkit/openvino) of intel and the jupyter notebooks provided by them along with the help of <docs.openvino.ai>