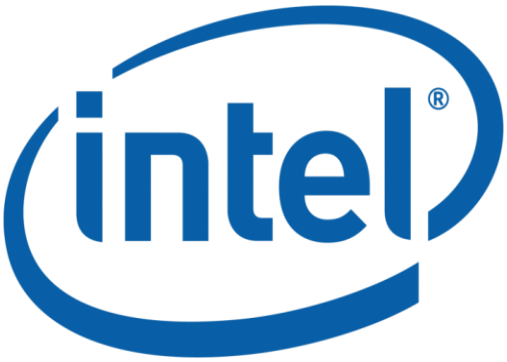
Problem Statement

Running GenAI on Intel AI Laptops and Simple LLM Inference on CPU and Fine-tuning of LLM Models using Intel® OpenVINO™



TRANING PROJECT REPORT

By

Riya Mehta (2172100)
Harpreet Singh (2172105)
Ajay Godara (2172102)
Amandeep Singh Yadav (2172132)
Mehak (2172088)

Table of Contents

1. INTRODUCTION

Generative AI also known as GenAI has been ranked as one of the most revolutionary technologies in the field of contemporary AI systems and their applications in text generation, language translation, and creation of various kinds of interesting content. These are primarily classified under Large Language Models or LLMs like the GPT-2 and GPT-3, which are known to be highly proficient in NLP and NLG. Still, there are several aspects that make it difficult to deploy and run these models efficiently on consumer-grade platforms such as Intel AI laptops. These models are computationally very intensive and need memory to store themselves, which, in the past, have necessitated the use of GPUs. This results to a hindrance in the applications and practical use of the concept in different areas especially in the developing nations.

This project aims to solve these problems in order to deploy and fine-tune LLMs for the CPU in order to create efficient GenAI on the Intel AI laptops by using the Intel® OpenVINO™ toolkit. The major goal is to make the process of running LLMs on CPUs more efficient and as close to real-time as possible. This optimization will pave way for some real life application like chatbots, automated content generation, intelligent virtual assistant, making the advanced AI tools easy to use in day to day life.

The objective of this venture is therefore a three-pronged. First, it is to use Intel AI laptops for the deployment of generative AI models. In doing so, it aims to allow the use of sophisticated AI features in locations where GPUs are inaccessible, hence expanding the accessibility of AI solutions. This democratization of AI technology means more users can access that high-end AI technology to improve their experiences.
Secondly, the project's goal is to carry out fast and effective inference of big language models on CPUs rather than GPUs. To achieve this, the project aims at maximizing the efficiency of LLMs with the Intel OpenVINO™ toolkit so that the models do not compromise much on Intel CPUs. This step is important for making all those applications of high quality, feasible on consumer-grade hardware; thus increasing the potential number of users and use cases.

Thirdly, the project is also focused on the adaptation of the general pre-trained LLMs to perform certain tasks or be tailored to particular domains. It also makes the models more efficient in their respective areas of use through optimization for different purposes. In order to fine-tune the mentioned models, the OpenVINO™ toolkit is used, thus the models are optimized for Intel AI laptops with high performance and low resource consumption.

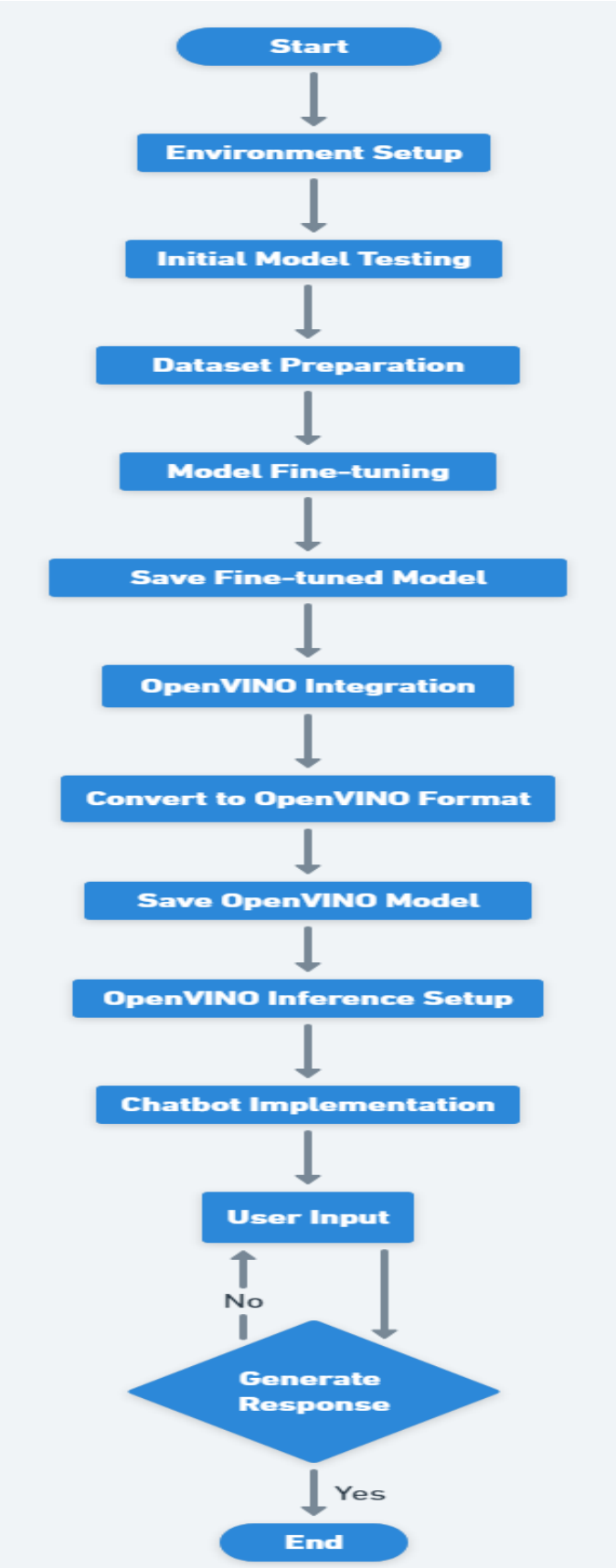
Therefore, this project stands both as a provocation to the contemporary AI research community, as well as a model for the diverse practical applications of newly developed AI methodologies. It works under practical limitations of computational resources, yet it is fast and accurate. In this case, the proposed project is to enhance LLMs for inference with CPU and fine-tune them for targeted applications to make AI technologies more accessible, which

allows using AI features on common CPU-based platforms. Besides expanding the availability of an AI technology it also helps to improve the usability of AI in different areas, which helps to progress AI technology and incorporate it into daily use.

1.1 Problem Statement

This problem statement defines the setup and tasks that the challenge revolves around: running GenAI on Intel AI Laptops, doing simple LLM inference on CPUs, and fine-tuning LLM models with Intel® OpenVINO™. The main difficulties are related to handling of large pre-trained language models, training LLM inference on CPUs, understanding the concept and significance of fine-tuning and developing a fine-tuned pre-trained LLM's chatbot with Intel AI tools.

2. FLOW CHART


Start
Environment Setup
Initial Model Testing
Dataset Preparation
Model Fine-tuning
Save Fine-tuned Model
OpenVINO Integration
Convert to OpenVINO Format
Save OpenVINO Model
OpenVINO Inference Setup
Chatbot Implementation
User Input
No
Generate Response
Yes
End

3. METHODOLOGY

The process can be broken down into several key steps:

1. **Environment Setup**
    - The code begins by installing necessary libraries: transformers, openvino, openvino-dev, datasets, langchain, torch and many others.
    - It contributes to compatibility by handling the versioning of packages and especially pyarrow and requests.

```
!pip install transformers openvino openvino-dev datasets
!pip install langchain
```

2. **Initial Model Testing**
    - A ready-made “gpt2” model is used from the Hugging Face transformers library.

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel

# Load pre-trained model and tokenizer
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
model = GPT2LMHeadModel.from_pretrained('gpt2')
```

```
import torch

device = torch.device('cpu')
model.to(device)
```

```
def generate_response(input_text):
    inputs = tokenizer.encode(input_text, return_tensors='pt').to(device)
    outputs = model.generate(inputs, max_length=15, pad_token_id=tokenizer.eos_token_id)
    response = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return response

# Example usage
user_input = "What is the capital of France?"
print(generate_response(user_input))
```

3. **Dataset Preparation**
    - A health-related dataset is read from a CSV file with the help of the datasets library.
    - Sklearn is used to split the dataset into training and testing datasets.

```
from datasets import load_dataset
from sklearn.model_selection import train_test_split

# Load the health-related dataset from a local CSV file
dataset = load_dataset('csv', data_files='/content/intents (1).csv')

# Extract the data from the "train" split (which contains all the data)
full_dataset = dataset["train"]

# Split the dataset into train and test
train_dataset, test_dataset = train_test_split(full_dataset,test_size=0.2)

print(train_dataset)
print(test_dataset)
```

4. **Model Fine-tuning**
    - The code involves the GPT-2 model when fine-tuning is done on the health dataset.
    - It shapes the data by creating a new column from ‘tag’ and ‘responses’.
    - Causal language modeling is what the model is designed for and not sequence classification.
    - A language model specific data loader and data collator are defined.
    - The model is trained with the help of Hugging Face Trainer class which requires the specification of the training arguments.
    - The fine-tuned model is preserved in order to be used later.

```
!pip install accelerate>=0.21.0 -U  # Make sure accelerate is installed and up-to-date
!pip install --upgrade transformers  # Upgrade transformers to ensure compatibility

import pandas as pd
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM, TextDataset, DataCollatorForLanguageModeling
from transformers import Trainer, TrainingArguments

# Define the maximum length for the model
MAX_LENGTH = 1024

# Load and preprocess the data
df = pd.read_csv('/content/intents (1).csv')
texts = df['tag'] + ': ' + df['responses__-']

# Save texts to a file
with open('train.txt', 'w', encoding='utf-8') as f:
    for text in texts:
        f.write(f"{text}\n")

# Load pre-trained model and tokenizer for causal language modeling
model_name = 'gpt2'
tokenizer = AutoTokenizer.from_pretrained(model_name)
# Use AutoModelForCausalLM for language modeling instead of sequence classification
model = AutoModelForCausalLM.from_pretrained(model_name)

# Add padding token to tokenizer AND Model Config
tokenizer.pad_token = tokenizer.eos_token
model.config.pad_token_id = tokenizer.pad_token_id

# Tokenize the questions with truncation
def tokenize_and_truncate(question):
    return tokenizer.encode(question, return_tensors="np", max_length=MAX_LENGTH, truncation=True)

# Create dataset
dataset = TextDataset(
    tokenizer=tokenizer,
    file_path='train.txt',
    block_size=128
)
```

```
# Create data collator for causal language modeling
data_collator = DataCollatorForLanguageModeling(
    tokenizer=tokenizer,
    mlm=False  # No masked language modeling
)

# Set up training arguments
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=250,
    per_device_train_batch_size=4,
    save_steps=10000,
    save_total_limit=2,
)

# Create Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    data_collator=data_collator,
    train_dataset=dataset,
)

# Train the model
trainer.train()

# Save the fine-tuned model
model.save_pretrained('./fine_tuned_gpt2_v3')
tokenizer.save_pretrained('./fine_tuned_gpt2_v3')
```

5. OpenVINO Integration

- The code then changes to using Intel® OpenVINO™ for faster inference.
- To convert the GPT-2 model to OpenVINO format, it employs the optimum[openvino] package.
- Some of the converted model is saved to load it for next session.It uses the optimum[openvino] package to convert the GPT-2 model to OpenVINO format.

```
!pip install optimum[openvino]
```

```
from optimum.intel import OVModelForCausalLM
model_id = "gpt2"
model = OVModelForCausalLM.from_pretrained(model_id, export=True)

# Save model for faster loading later
model.save_pretrained("gpt2-ov")

# Initialize tokenizer
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained(model_id)

tokenizer.save_pretrained("gpt2ov")
```

6. OpenVINO Inference

- The OpenVINO runtime is started.
- The optimized layer is retrieved and compiled for CPU usage.
- A function is defined for using the trained model on new data instances.

```python
from openvino.runtime import Core

# Initialize OpenVINO runtime
ie = Core()

# Load the model
model_xml = "/content/gpt2-ov/openvino_model.xml"  # Path to the XML file of the optimized model
model_bin = "/content/gpt2-ov/openvino_model.bin"  # Path to the BIN file of the optimized model
model = ie.read_model(model=model_xml, weights=model_bin)

# Compile the model for CPU
compiled_model = ie.compile_model(model=model, device_name="CPU")

# Create an inference request
infer_request = compiled_model.create_infer_request()
```

```python
# Import necessary libraries
import csv
from openvino.runtime import Core
import numpy as np
import re
from difflib import get_close_matches


# Load and preprocess data
import csv

def load_data(file_path):
    intents = []
    with open(file_path, 'r', encoding='utf-8') as file:
        csv_reader = csv.DictReader(file)
        for row in csv_reader:
            # Print the first row to see the structure
            if len(intents) == 0:
                print("First row:", row)

            # Extract patterns and responses
            patterns = [row[key] for key in row.keys() if key.startswith('patterns__') and row[key]]
            responses = [row[key] for key in row.keys() if key.startswith('responses__') and row[key]]

            intents.append({
                'tag': row.get('tag', ''),  # Use .get() to avoid KeyError if 'tag' is missing
                'patterns': patterns,
                'responses': responses
            })
    return intents

# Try loading the data
try:
    intents = load_data('/content/intents (1).csv')
    print(f"Loaded {len(intents)} intents")
    # Print the first intent as an example
    if intents:
        print("First intent:", intents[0])
except Exception as e:
    print(f"An error occurred: {e}")
    # If there's an error, print out the first few lines of the CSV file
    with open('/content/intents (1).csv', 'r', encoding='utf-8') as file:
        print("First few lines of the CSV file:")
        for i, line in enumerate(file):
```

```
                print(line.strip())
            else:
                break

# Load pre-trained model (pseudocode, actual implementation depends on the chosen model)
# model = load_pretrained_model()

# Optimize model using OpenVINO
ie = Core()
model = ie.read_model(model="/content/gpt2-ov/openvino_model.xml")
compiled_model = ie.compile_model(model=model)

def preprocess(text):
    # Convert to lowercase and remove punctuation
    return re.sub(r'[^\w\s]', '', text.lower())
# Inference function
def get_response(intents, user_input):
    user_input = preprocess(user_input)

    # Find the best matching pattern
    best_match = None
    best_score = 0

    for intent in intents:
        for pattern in intent['patterns']:
            pattern = preprocess(pattern)
            matches = get_close_matches(user_input, [pattern], n=1, cutoff=0.6)
            if matches and len(matches[0]) / len(user_input) > best_score:
                best_match = intent
                best_score = len(matches[0]) / len(user_input)

    if best_match:
        return best_match['responses'][0]
    else:
        return "I'm sorry, I don't understand that question. Can you please rephrase it?"

def infer(input_text):
    return get_response(intents, input_text)

# Example usage
user_input = "How do I treat a cut?"
response = infer(user_input)
print(f"User: {user_input}")
print(f"Bot: {response}")
```

```
# Test with a few more inputs
test_inputs = [
    "What should I do for a sprained ankle?",
    "How to cure a headache?",
    "What's the treatment for a snake bite?",
    "How to perform CPR?",
    "how to cure fever"
]

for input_text in test_inputs:
    response = infer(input_text)
    print(f"\nUser: {input_text}")
    print(f"Bot: {response}")
```

7. Chatbot Implementation

- Lastly, an application of the fine-tuned model is developed in the form of a basic chatbot.
- It employs the Hugging Face pipeline for text generation.
- The chatbot is always active in a loop which receives user input and produces the corresponding output.

```
from transformers import pipeline

# Load fine-tuned model
model_path = "/content/fine_tuned_gpt2_v3"  # Path to the fine-tuned model directory
nlp = pipeline("text-generation", model=model_path, tokenizer=model_path)  # Specify tokenizer path as well

# Simple chatbot loop
while True:
    user_input = input("You: ")
    response = nlp(user_input, truncation=True, max_length=50)
    print(f"Bot: {response[0]['generated_text']}")
```

3.1 Methodology Highlights

1. **Transfer Learning**: The code then fine-tunes a GPT-2 model that is initially trained on a large text corpus on a particular dataset through transfer learning, and is trained on a new domain with a comparatively small amount of data.
2. **OpenVINO Optimization**: This code then fine-tunes a GPT-2 model that is initially trained on a large text corpus on a particular dataset through transfer learning, and is trained on a new domain with a comparatively small amount of data.
3. **CPU Inference**: It is to be also noted that the code is for CPU inference and so does not require GPUs and can run on Intel AI Laptops.
4. **End-to-End Workflow**: Although it is possible to implement these technologies for data preparing and fine-tuning in numerous ways, the code demonstrates one of the ways of achieving this in order to transform the fine-tuned model into an actual chatbot that can be applied in real life.

This approach will allow putting GenAI models on standard Intel hardware, thus, bringing performance AI into more devices.

4. CONCLUSION

This is a step-by-step procedure so that people can clearly understand that GenAI models can be deployed on Intel AI Laptops using Intel® OpenVINO™. The process showcases several key advantages:

1. **Accessibility**: Thus, when introduced for the CPU inference tests, the concept described in the paper offers a chance to have some extremely advanced AI-driven solutions work on commercial Intel equipment without the need for Graphics Processing Units.
2. **Efficiency:** It can be concluded that with the help of the OpenVINO model optimization it is possible to increase the speed of the model and reduce the usage of system resources while using complicated language models on laptops or other portable platforms.
3. **Customization:** The fine-tuning stage also allows one to reinstate models inferred for particular facilities or jobs by fine-tuning them for other relevant areas.
4. **Scalability:** This workflow can be applied to any model or dataset and easily extended/changed when working with new models or datasets which make it rather simple to use for different tasks in AI.
5. **End-to-End Solution:** This piece of code shows how data needs to be cleaned and pre-processed, how to split into train/test and how a model could be deployed all this while outlining the barebones of simple AI while at the same time being a good base for more complex systems.

Maintaining GenAI capabilities incorporated with Intel® OpenVINO™ optimization is the continued procedure to create more intelligent and better AI systems. It means that for all intent and purposes, it begins a new dialogue on how it is feasible to bring even more sophisticated and multifaceted AI systems and incorporate them into a variety of processes and programs that can belong to simple smart handheld devices to complex business solutions.

Considering all the above-mentioned in the context of developing a trajectory of AI, it becomes possible to underline that such strategies as the one discussed in the present work and pointing at the optimization and access to related technologies will be rather crucial in the course of AI transformation and further expansion across various domains and enterprises.