INTEL-UNNATI AI&ML JOURNAL
Number 2 (2024), pp. 1–12
PROBLEM STATEMENT 16:

Code:IUCP-2024

MACHINE LEARNING

Running GenAI on Intel AI laptops and Simple LLM Interference on CPU and fine-tuning of LLM models using Intel OpenVINO

Abin Sabu Philip, Abinand M.

Saintgits Group of Institutions, Kottayam, Kerala

Abstract: This project is designed to help newcomers get started in the world of Generative Artificial Intelligence (GenAI) by engaging in a series of hands on activities.Participants will gain foundational knowledge in GenAI, perform inference on a text generation, Large Language Model (LLM) using the TinyLlama model, convert the model to ONNX format, optimize it with the Intel OpenVINO toolkit and evaluate the inference of the optimized model.The main goal of this project is to generate a chatbot by segmenting the input text into tokens, generating text from these tokens using sampling approaches. The goal is to get people better acquainted with GenAI and implementing improvements in a model leading to the creation of an interactive chatbot. This report discusses the method and execution process emphasizing the benefits and constraints of utilizing TinyLlama along with the complexities involved in converting and optimizing models.

Keywords: Generative AI (GenAI), Large Language Models (LLMs), Intel OpenVINO Toolkit, Model Optimization,Inference, Model Fine-Tuning, Hugging Face Transformers, Custom Chatbot Development, TinyLlama.

1 Introduction

Given the possibility and the speed at which Artificial Intelligence (AI) is being developed, new possibilities have been made available especially new advances in the Natural Language Processing (NLP). Software applications such as Generative AI (GenAI), and

Large Language Models such as GPT-3 the BERT, and other related models can all explain and even generate human-like texts. However, the Flow models are problematic in terms of theoretical computation where efficiency and costs are a major issue.

With the rapid advancement of Artificial Intelligence (AI), new opportunities have emerged, particularly in the realm of Natural Language Processing (NLP). Generative AI (GenAI) applications and Large Language Models (LLMs) like GPT-3 and BERT have demonstrated remarkable capabilities in generating and understanding human-like text. However, these powerful models often come with significant computational demands, making them expensive and resource-intensive to deploy.

This project explores the potential of using the TinyLlama model, a compact and efficient LLM that offers an excellent balance between performance and resource consumption for inference tasks on CPUs. While GPT-3 and similar models deliver powerful outputs, they require high-performance CPUs that are costly to acquire and maintain. In contrast, TinyLlama provides a more accessible approach to experimenting with GenAI technology, allowing users to gain valuable insights without the need for expensive infrastructure.

OpenVINO (Open Visual Inference and Neural Network Optimization) plays a crucial role in enhancing the efficiency of LLMs, enabling their easy deployment on Intel AI laptops. The primary goal of this project is to develop a straightforward solution for executing LLM inference on CPUs, particularly for applications like chatbots. By combining the advanced NLP capabilities of Hugging Face libraries with the optimization tools provided by OpenVINO, this project aims to deliver efficient and cost-effective AI applications.

The report outlines the step-by-step processes undertaken to achieve these goals, including environment setup, model optimization, and deployment strategies. Additionally, it showcases the practical application of these fine-tuned models through the creation of a custom chatbot, demonstrating the potential of integrating GenAI techniques with Intel's advanced optimization tools.

2 Libraries Used

In the project for various tasks, following packages are used.

```
torch
transformers
openvino
optimum[openvino]
nncf
gradio
psutil
numpy
```

3 Methodology

The methodology of this project involves several key steps to ensure a comprehensive understanding and practical experience in GenAI.

To start with, the model needed to be chosen. The reason for choosing the TinyLlama model was its suitability for environments with low hardware support. We chose TinyLlama as our model because of its excellent computable performance on less powerful machines. Then we had to test tinyLlama to see how well it works in producing text.

After testing TinyLlama, we converted it into a format called ONNX. This involved taking the TinyLlama model that was already trained and using a tool called PyTorch to change it into ONNX format. This new format was checked to make sure everything was right, and it gave us a file called model.onnx that we could work with.

The next action is to optimize the ONNX model using the Intel OpenVINO toolkit. OpenVINO presents a range of tools that streamline the model inference on Intel hard-ware. The optimization process entails converting ONNX model into OpenVINOâĂŹs intermediate representation (IR) format that will produce two files model.xml and model.bin. These files have been optimized for efficient inference by considering Intel hardware acceleration capabilities.

Having our optimized model in place, we divided the input text into smaller entities referred to as tokens. Each token was given an ID that could be understood by the model.That is when these token IDs are used to make textual material through numerous sampling methods. Temperature and sampling controls the randomness of the predictions, while top-k and top-p sampling limit the model's predictions to the most probable tokens, enhancing the quality and coherence of the generated text.

Finally, we put our optimized model into a chatbot setup. We made a user interface using a tool called Streamlit, implementing backend logic to handle user inputs, tokenize the inputs, generate responses using the optimized model, and display the responses in the chatbot interface. The performance of the chatbot undergoes evaluation and improvement so that optimal results can be achieved.

The process flow of the project is shown below:

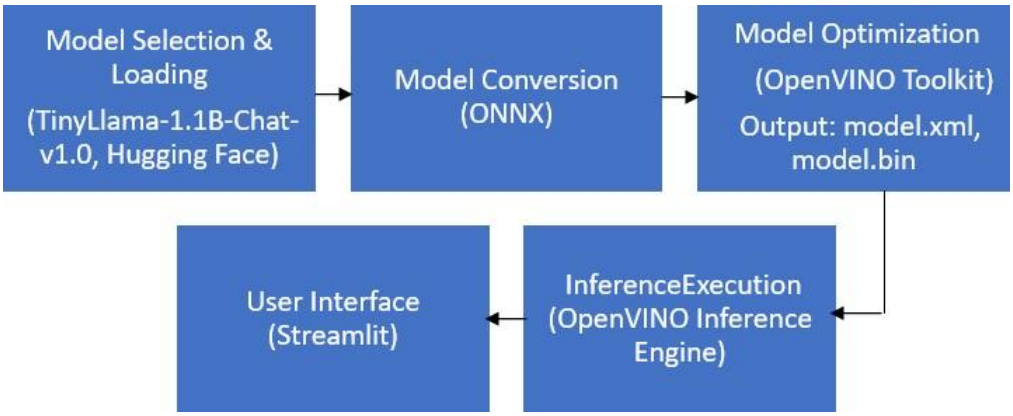

Figure 1: Process Flow

4 Implementation

We implemented our project in a step by step manner:

4.1 Model Selection and Initial Inference

We began by selecting a Large Language Model (LLM) using Hugging Face Transformers, specifically TinyLlama-1.1B-Chat-v1.0, due to its small size and lower computational requirements. This model has approximately 1.2 billion parameters, making it a balanced choice between performance and accessibility. Initial tests were conducted to assess the model's performance by feeding in sample input texts and analyzing the outputs. These tests helped us understand the model's strengths and weaknesses for future improvements.

4.2 Model Conversion to ONNX Format

At the VS code software, we ran this model and then converted this file into ONNX initiating a model conversion. The process flow had some space requirements, ie this model occupies around 2GB which was difficult. The pre-trained TinyLlama model was loaded and by using PyTorch's torch.onnx.export function, the model was exported into ONNX format.A check was made on integrity of resulting model.onnx file thereby confirming whether themodel is ready for Intel OpenVINO optimization.

4.3 Model Optimization with Intel OpenVINO

The ONNX model was improved using the Intel OpenVINO toolkit. Therefore, the Model Optimizer transformed this to the IR format of OpenVino which gave us model.xml and model.bin files. We did that so as to make them work as fast as possible on Intel hardware by leveraging OpenVino for higher speed and efficiency. This will enable faster and smoother processing.

4.4 Tokenization and Text Generation

We applied a Tokenizer designed for TinyLlama model on input text in order to break it into smaller pieces called tokens. Each token was then given an ID number which was different from all other tokens that were used in making up the sentences. For this reason, we implemented several sampling methods in order to generate meaningful and high-quality answers such as adjusting the temperature to 0.7 and using top-k and top-p sampling. These techniques helped us generate coherent and natural-sounding text from the tokens.

4.5 Chatbot Development and Integration

The optimized model was integrated into a chatbot framework developed using Gradio. The backend logic handled user inputs, tokenized the inputs, generated responses using the optimized model and displayed the responses in the chatbot interface. The chatbot's performance was evaluated and refined to ensure it provided accurate and relevant responses, demonstrating the practical applications of GenAI.

5 Results & Discussion

The project has been successful in achieving the goal of developing a comprehensive solution that encompasses environment setup, CPU-based LLM inference optimization and creating specific applications like chatbots.The interaction with the customized chatbot we developed is given in the figure1 given below:

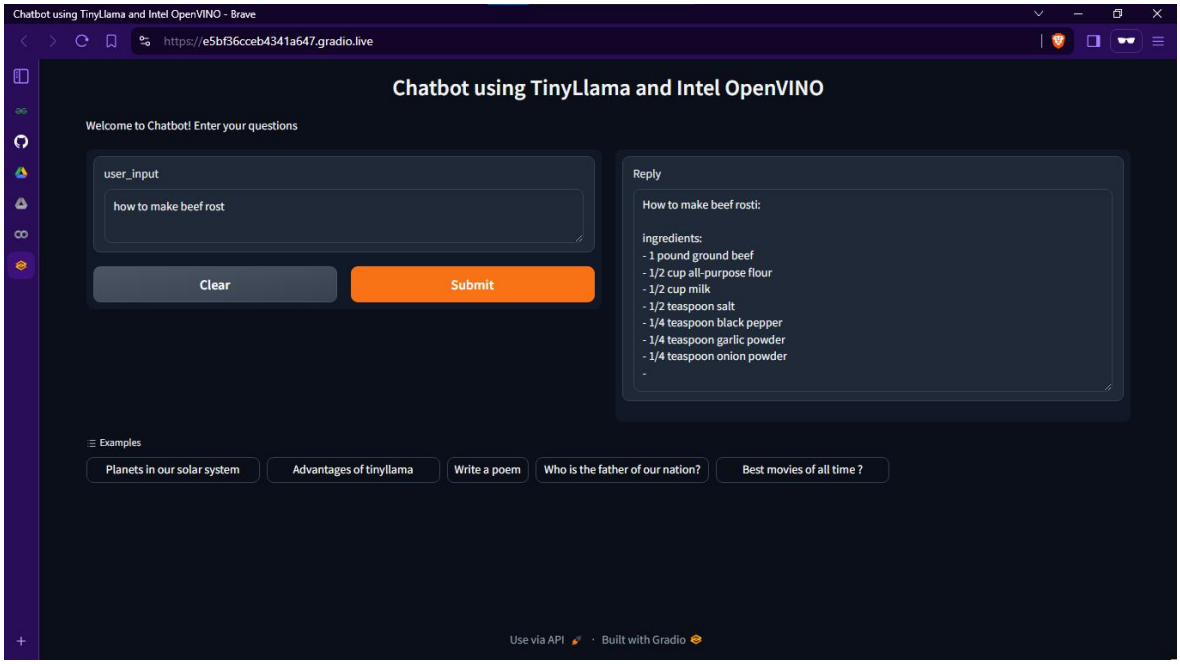

Figure 2: Customized chatbot

6 Conclusions

This project successfully introduced participants to the field of Generative Artificial Intelligence (GenAI), guiding them through the process of performing inference on an LLM, converting and optimizing the model, and developing a custom chatbot. By working with the TinyLlama model and Intel's OpenVINO toolkit, participants gained practical experience in model optimization and text generation techniques. The resulting chatbot showcases the potential of GenAI in creating interactive and intelligent applications, providing a solid foundation for further exploration and development in this exciting field.

7 Acknowledgments

We would like to express our heartfelt gratitude and appreciation to Intel© Corporation for providing an opportunity to this project. First and foremost, we would like to extend our sincere thanks to our team mentors *Dr.Anju Pratap* for their invaluable guidance and constant support throughout the project. We are deeply indebted to our college Saintgits College of Engineering and Technology for providing us with the necessary resources,and sessions. We would also like to thank the industrial mentor *Mr Abhishek Nandy*, for taking time out of his busy schedules to provide us with training and for answering our queries. We extend our gratitude to all the researchers, scholars, and experts in the field of machine learning and natural language processing and artificial intelligence, whose seminal work has paved the way for our project. We acknowledge the mentors, institutional heads, and industrial mentors for their invaluable guidance and support in completing this industrial training under Intel© -Unnati Programme whose expertise and encouragement have been instrumental in shaping our work.

References

[1] A. R. Borah, N. T N and S. Gupta, "Improved Learning Based on GenAI", *2nd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT), Bengaluru, India*, 2024

[2] J. Qadir, "Learning 101 reloaded: Revisiting the basics for the GenAI era", *IEEE*, 2024

[3] V. V. Zunin, "Intel OpenVINO Toolkit for Computer Vision: Object Detection and Semantic Segmentation", *International Russian Automation Conference (RusAutoCon), Sochi, Russian Federation*, 2021

[4] J. Zhang, Y. Zhang, M. Chu, S. Yang and T. Zu, "A LLM-Based Simulation Scenario Aided Generation Method",*IEEE 7th Information Technology and Mechatronics Engineering Conference (ITOEC), Chongqing, China*, 2023

A Main code sections for the solution

A.1 Inference

```python
# Imports and Setup
import torch
from transformers import LlamaTokenizer, LlamaForCausalLM
import os
import time
import psutil
import openvino.runtime as ov
from openvino.runtime import Core
import numpy as np
import matplotlib.pyplot as plt

# Initialize model and tokenizer
model_name = 'TinyLlama/TinyLlama-1.1B-Chat-v1.0'
tokenizer = LlamaTokenizer.from_pretrained(model_name)
model = LlamaForCausalLM.from_pretrained(model_name)

# Load quantized model
output_dir = "/content/drive/MyDrive/openvino/chatbot/FINAL"
quantized_ir_path = os.path.join(output_dir, "tinyllama_quantized.xml")
core = Core()
quantized_model = core.read_model(quantized_ir_path)
quantized_compiled_model = core.compile_model(quantized_model, "CPU")

# Define utility functions
def get_file_size(file_path):
    return os.path.getsize(file_path) / (1024 ** 2)  # Size in MB

def benchmark_model(model, input_text, tokenizer, num_runs=10):
    inputs = tokenizer(input_text, return_tensors="pt")
    start_time = time.time()
    for _ in range(num_runs):
        outputs = model(**inputs)
    end_time = time.time()
    avg_time = (end_time - start_time) / num_runs
    return avg_time

def get_memory_usage():
    process = psutil.Process(os.getpid())
    return process.memory_info().rss / (1024 ** 2)  # Convert to MB

def benchmark_quantized_model(compiled_model, input_text, tokenizer, num_runs=10):
    inputs = tokenizer(input_text, return_tensors="pt")
    input_ids = inputs["input_ids"].detach().cpu().numpy()

    # Get the expected input shape from the model
    input_shape = compiled_model.input(0).shape

    # Pad or truncate the input to match the expected shape
    if input_ids.shape[1] < input_shape[1]:
        pad_length = input_shape[1] - input_ids.shape[1]
        input_ids = np.pad(input_ids, ((0, 0), (0, pad_length)), mode='constant', constant_values=tokenizer.pad_token_id)
    elif input_ids.shape[1] > input_shape[1]:
        input_ids = input_ids[:, :input_shape[1]]

    start_time = time.time()
    for _ in range(num_runs):
        infer_request = compiled_model.create_infer_request()
        infer_request.infer(inputs={"input_ids": input_ids})
    end_time = time.time()
    avg_time = (end_time - start_time) / num_runs
    return avg_time
```

```python
# Calculate model sizes
original_model_size = sum(p.numel() * p.element_size() for p in model.parameters()) / (1024 ** 2)
quantized_model_size = get_file_size(quantized_ir_path) + get_file_size(quantized_ir_path.replace('.xml', '.bin'))

print(f"Original model size: {original_model_size:.2f} MB")
print(f"Quantized model size: {quantized_model_size:.2f} MB")

# Cell 6: Benchmark models
input_text = "India is known for"
original_model_time = benchmark_model(model, input_text, tokenizer)
original_model_memory = get_memory_usage()

# Get the expected input shape from the quantized model
input_shape = quantized_compiled_model.input(0).shape
print(f"Expected input shape for quantized model: {input_shape}")

quantized_model_time = benchmark_quantized_model(quantized_compiled_model, input_text, tokenizer)
quantized_model_memory = get_memory_usage()

# Print comparison results
print(f"Original model - Average Inference Time: {original_model_time:.4f} seconds")
print(f"Quantized model - Average Inference Time: {quantized_model_time:.4f} seconds")
print(f"Original model - Peak Memory Usage: {original_model_memory:.2f} MB")
print(f"Quantized model - Peak Memory Usage: {quantized_model_memory:.2f} MB")
print(f"Original model size: {original_model_size:.2f} MB")
print(f"Quantized model size: {quantized_model_size:.2f} MB")


# Sample data
models = ['Original', 'Quantized']
inference_times = [original_model_time, quantized_model_time]
model_sizes = [original_model_size, quantized_model_size]

# Plot Inference Time
fig, ax1 = plt.subplots(figsize=(8, 6))
ax1.bar(models, inference_times, color=['blue', 'orange'])
ax1.set_ylabel('Inference Time (seconds)')
ax1.set_title('Inference Time Comparison')
for i, v in enumerate(inference_times):
    ax1.text(i, v, f'{v:.4f}', ha='center', va='bottom')

# Adjust layout and save the plot
plt.tight_layout()
plt.savefig('inference_FINAL.png')
plt.close(fig)  # Close the figure to avoid overlap

# Plot Model Size
fig, ax2 = plt.subplots(figsize=(8, 6))
ax2.bar(models, model_sizes, color=['blue', 'orange'])
ax2.set_ylabel('Model Size (MB)')
ax2.set_title('Model Size Comparison')
for i, v in enumerate(model_sizes):
    ax2.text(i, v, f'{v:.2f}', ha='center', va='bottom')

# Adjust layout and save the plot
plt.tight_layout()
plt.savefig('model_size_FINAL.png')
plt.close(fig)  # Close the figure to avoid overlap

print("Graphs saved successfully.")
```

A.2 Conversion and optimisation

```python
import torch
from transformers import LlamaTokenizer, LlamaForCausalLM
import os
import openvino.runtime as ov
from nncf import quantize
from nncf.quantization import QuantizationPreset
from torch.utils.data import Dataset, DataLoader

# Initialize model and tokenizer
model_name = 'TinyLlama/TinyLlama-1.1B-Chat-v1.0'
tokenizer = LlamaTokenizer.from_pretrained(model_name)
model = LlamaForCausalLM.from_pretrained(model_name)

# Create dummy input
dummy_input = tokenizer("Hello, how are you?", return_tensors="pt")
print("Input shape:", dummy_input['input_ids'].shape)

# Specify output directory
output_dir = "FINAL"
os.makedirs(output_dir, exist_ok=True)

# Specify the output file path for ONNX
onnx_path = os.path.join(output_dir, "tinyllama.onnx")

# Export the model to ONNX format
torch.onnx.export(
    model,
    (dummy_input['input_ids'],),
    onnx_path,
    opset_version=14,
    input_names=["input_ids"],
    output_names=["output"]
)

# Load the ONNX model and convert to OpenVINO IR
core = ov.Core()
onnx_model = core.read_model(onnx_path)

# Specify the output paths for the OpenVINO IR format files
ir_xml_path = os.path.join(output_dir, "tinyllama_ir.xml")
ir_bin_path = os.path.join(output_dir, "tinyllama_ir.bin")

# Convert the ONNX model to OpenVINO IR format
ov.serialize(onnx_model, ir_xml_path, ir_bin_path)

print("Model has been successfully converted to OpenVINO IR format and saved to", output_dir)

# Create a dummy calibration dataset
class CalibrationDataset(Dataset):
    def __init__(self, num_samples, batch_size=1):
        self.num_samples = num_samples
        self.batch_size = batch_size

    def __len__(self):
        return self.num_samples

    def __getitem__(self, item):
        # Generate a tensor with the correct shape [1, 7]
        input_ids = torch.randint(0, 100, (1, 7), dtype=torch.int32)  # Ensure the shape is [1, 7]
        return {"input_ids": input_ids}
```

A.3 Chatbot integretion

```python
import os
import re
import string
import torch
import gradio as gr
from transformers import pipeline, LlamaTokenizer
import logging

# Set up logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Preprocessing function
def preprocess_text(text):
    text = text.lower()
    text = text.translate(str.maketrans('', '', string.punctuation))
    text = ' '.join(text.split())
    return text

# Postprocessing function
def postprocess_response(response):
    response = response.strip()
    sentences = re.split(r'(?<!\w\.\w.)(?<![A-Z][a-z]\.)(?<=\.|\?)\s', response)
    sentences = [sentence.capitalize() for sentence in sentences]
    response = ' '.join(sentences)
    return response

try:
    # Load the tokenizer and quantized model from the saved directory
    model_path = '/content/drive/MyDrive/openvino/chatbot/FINAL'
    logger.info(f"Loading tokenizer from {model_path}")
    tokenizer = LlamaTokenizer.from_pretrained(model_path)
    logger.info("Tokenizer loaded successfully")


    logger.info(f"Loading model from {model_path}")
    pipe = pipeline("text-generation", model=model_path, tokenizer=tokenizer)
    logger.info("Model loaded successfully")
except Exception as e:
    logger.error(f"Error loading model or tokenizer: {e}")
    raise e

# Function to generate response
def generate_response(user_input):
    try:
        user_input = preprocess_text(user_input)
        logger.info(f"Generating response for input: {user_input}")
        response = pipe(user_input, max_length=100, temperature=0.7, top_k=50, top_p=0.9, num_return_sequences=1)[0]['generated_text']
        response = postprocess_response(response)
        logger.info(f"Response generated: {response}")
        return response
    except Exception as e:
        logger.error(f"Error generating response: {e}")
        return "Error generating response"
```

```python
# Sample text generation prompts
sample_prompts = [
    "Write an e-mail",
    "Advantages of tinyllama",
    "Write a poem",
    "Write a code for :",
    "Best movies of all time ?"
]

# Define Gradio interface
iface = gr.Interface(
    fn=generate_response,
    inputs=gr.Textbox(lines=2, placeholder="Enter your queries "),
    outputs=gr.Textbox(label="Reply"),
    title="Chatbot using TinyLlama and Intel OpenVINO ",
    description="Welcome to Chatbot! Enter your questions ",
    examples=sample_prompts,
    theme="default",
    allow_flagging="never"
)

# Launch the Gradio interface
try:
    logger.info("Launching Gradio interface...")
    iface.launch(share=True)
    logger.info("Gradio interface launched successfully")
except Exception as e:
    logger.error(f"Error launching Gradio interface: {e}")
```

B Github Repository

https://github.com/ABINSABUPHILIP/TECHNOWIZZ