INTEL-UNNATI AI&ML JOURNAL
Number 2 (2024), pp. 1–10

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

Aiswarya Rahul, Jyothsna Sara Abey, Cinta Susan Thomas, Jacksilin P Titus, and Tebin Philip George

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

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.

We worked on this project by using the TinyLlama model, which is a compact and effective LLM that works efficiently on small hardware. This model was chosen basically because of its good performance to resource consumption ratio for inference task on CPUs. GPT-3 has a powerful output but needs high performing CPUs that are expensive to buy and maintain. It is evident that TinyLlama provides a more open-ended approach to experimenting with GenAI technology. People can gain knowledge about it without having to set up an expensive infrastructure or environment. OpenVINO (Open Visual Inference and Neural Network Optimization) assists in increasing the efficiency of LLMs so they can be deployed easily on Intel AI laptops.

The main focus of this project is to come up with a simple solution for creating the environment for executing LLM inference on CPUs and for the particular application, for instance, chatbots. Thus, this solution intends to provide efficient and inexpensive artificial intelligence applications by combining the functionalities of Hugging Face advanced NLP libraries with OpenVINO optimization tools. In addition, it also demonstrates the practicality of these fine-tuned models through the creation of a bespoke chatbot having demonstrated the ability of combining GenAI techniques and Intel's advanced optimization tools. The report however, contains step by step processes undertaken to achieve such goals including, environment setup, model optimization as well as deployment strategies.

2 Libraries Used

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

```
NumPy
Transformers
OpenVINO
Optimum Intel
ONNX
Torch
Streamlit
```

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 hardware. 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:

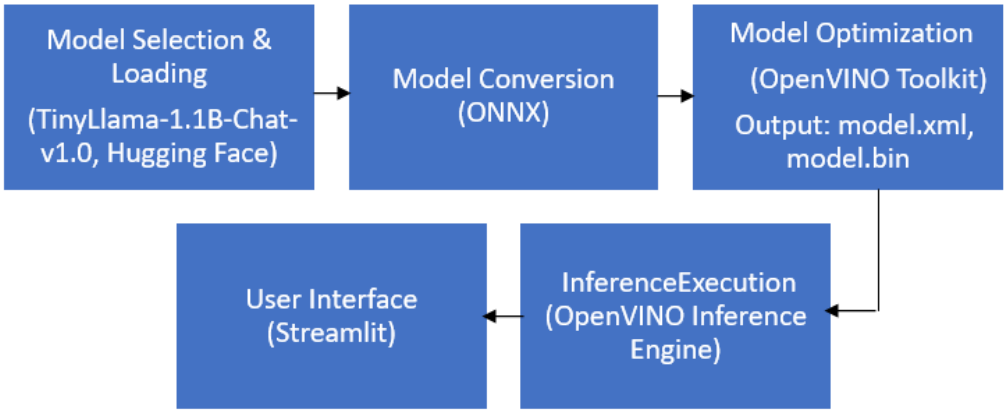

Figure 1: Process Flow

4 Implementation

We implemented our project in a step by step manner:

4.1 Model Selection and Initial Inference

Firstly we chose a LLM model using Hugging face transformers. ie,**TinyLlama-1.1B-Chat-v1.0** is selected due to it's small size and lower computational requirements. It contains around only 1.2 billion parameters for the text generation.TinyLlama as a model was chosen on account of its balance between performance and accessibility. Initial tests were conducted on the model to assess how it performs.We then fed in some sample input texts and analyzed the outputs.These tests helped us understand the strengths and weaknesses of the model so that we could do better in future.

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 the model 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 Streamlit. 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:

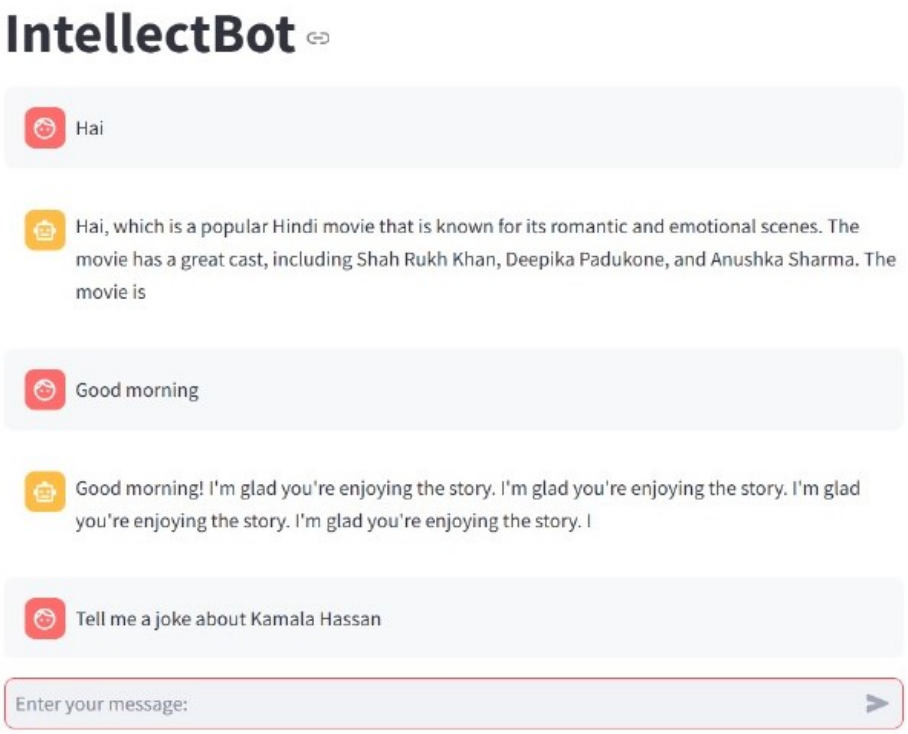

Figure 2: Customized chatbot **IntellectBot**

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.Pradeep C & Siju Swamy 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 TinyLlama Inference

The code for the TinyLlama Inference is given by:

```
from torch.profiler import profile, ProfilerActivity
from transformers import AutoTokenizer, TextStreamer, AutoModelForCausalLM
from threading import Thread
import torch
import time
import sys

model_id = "TinyLlama/TinyLlama-1.1B-Chat-v1.0"
model = AutoModelForCausalLM.from_pretrained(model_id, use_cache=True).eval()
tokenizer = AutoTokenizer.from_pretrained(model_id, use_default_system_prompt=True
                                          )
tokenizer.pad_token_id = tokenizer.eos_token_id
```

```
streamer = TextStreamer(tokenizer, skip_special_tokens=True)

# Context Management
context_tokens = []
max_context_length = 4096  # Adjust based on model and memory

def get_user_input():
    return input("You: ")

while True:
    query = get_user_input()
    if query.lower() in ["exit", "quit"]:
        break

    new_tokens = tokenizer(query, return_tensors="pt")["input_ids"]
    context_tokens.extend(new_tokens[0].tolist())

    # Trim context to avoid exceeding model's max length
    if len(context_tokens) > max_context_length:
        context_tokens = context_tokens[-max_context_length:]

    generation_kwargs = dict(
        input_ids=torch.tensor([context_tokens], device=model.device),
        streamer=streamer,
        do_sample=True,
        top_k=50,
        top_p=0.9,
    )

    print("Assistant:")
    _ = model.generate(**generation_kwargs)

    # Update context with the model's response
    response_tokens = generation_kwargs["input_ids"][0].tolist()
    context_tokens = response_tokens
```

A.2 Conversion to ONNX

The python code section for this stage is shown below:

```
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
import os

model_id = "TinyLlama/TinyLlama-1.1B-Chat-v1.0"
output_onnx_path = "model/model.onnx"

# Load the model and tokenizer
model = AutoModelForCausalLM.from_pretrained(model_id, use_cache=True).eval()
tokenizer = AutoTokenizer.from_pretrained(model_id, use_default_system_prompt=True
                                          )
tokenizer.pad_token_id = tokenizer.eos_token_id

# Export the model to ONNX format
def export_to_onnx(model, tokenizer, output_path):
    max_context_length = 4096  # Adjust based on model and memory
    dummy_input = torch.randint(0, tokenizer.vocab_size, (1, max_context_length))
```

```
    # Use symbolic trace to avoid issues with Python values in the trace
    with torch.no_grad():
        torch.onnx.export(
            model,
            (dummy_input,),
            output_path,
            input_names=["input_ids"],
            output_names=["output"],
            dynamic_axes={"input_ids": {0: "batch_size", 1: "sequence"}, "output":
                                                    {0: "batch_size", 1: "
                                                   sequence"}},
            opset_version=14,
            export_params=True,
            do_constant_folding=True
        )
    print(f"Model exported to ONNX format at {output_path}")

# Export the model
export_to_onnx(model, tokenizer, output_onnx_path)

# Verify the file exists
if os.path.exists(output_onnx_path):
    print(f"ONNX model file '{output_onnx_path}' successfully created.")
else:
    print(f"Failed to create ONNX model file '{output_onnx_path}'.")
```

A.3 OpenVINO Optimization

The code for this task is given below:

```
import subprocess

onnx_model_path = "model/model.onnx"
output_model_path = "optimized_model/model.xml"

# Optimize the ONNX model using OpenVINO
def optimize_model_with_openvino(onnx_model_path, output_model_path):
    subprocess.run([
        "ovc",
        onnx_model_path,
        "--output_model", output_model_path
    ])
    print(f"Model optimized and saved in {output_model_path}")

# Optimize the model
optimize_model_with_openvino(onnx_model_path, output_model_path)
```

A.4 Text Inference for IR Model

Code for Text Inference of the IR Model is:

```
import streamlit as st
import numpy as np
from openvino.runtime import Core
from transformers import AutoTokenizer
```

```
# Load the OpenVINO model
def load_model(model_path):
    core = Core()
    model = core.read_model(model=model_path)
    compiled_model = core.compile_model(model=model, device_name="CPU")
    return compiled_model

# Preprocess input text
def preprocess_input(input_text, tokenizer):
    inputs = tokenizer(input_text, return_tensors="np", padding=True, truncation=
                                                    True)
    return inputs["input_ids"]

# Top-k sampling
def top_k_sampling(logits, k):
    logits = logits - np.max(logits)  # Normalize logits for numerical stability
    exp_logits = np.exp(logits)
    top_k_indices = np.argsort(exp_logits)[-k:]
    top_k_logits = exp_logits[top_k_indices]
    top_k_probs = top_k_logits / np.sum(top_k_logits)
    selected_index = np.random.choice(top_k_indices, p=top_k_probs)
    return selected_index

# Top-p (nucleus) sampling
def top_p_sampling(logits, p):
    logits = logits - np.max(logits)  # Normalize logits for numerical stability
    exp_logits = np.exp(logits)
    sorted_indices = np.argsort(exp_logits)[::-1]
    sorted_logits = exp_logits[sorted_indices]
    cumulative_probs = np.cumsum(sorted_logits / np.sum(sorted_logits))
    cutoff_index = np.searchsorted(cumulative_probs, p)
    top_p_indices = sorted_indices[:cutoff_index + 1]
    top_p_logits = exp_logits[top_p_indices]
    top_p_probs = top_p_logits / np.sum(top_p_logits)
    selected_index = np.random.choice(top_p_indices, p=top_p_probs)
    return selected_index

# Generate response using OpenVINO
def generate_response(compiled_model, tokenizer, input_text, max_length=50,
                                          temperature=0.7, top_k=50, top_p=0.9):
    # Preprocess input text
    input_tensor = preprocess_input(input_text, tokenizer)

    # Initialize the generated response with the input
    generated_tokens = input_tensor[0].tolist()

    # Generate tokens until the maximum length is reached
    for _ in range(max_length):
        # Perform inference
        outputs = compiled_model([np.array([generated_tokens], dtype=np.int32)])

        # Extract logits from the model output
        logits = outputs[compiled_model.output(0)][0, -1]

        # Apply temperature to logits
        logits /= temperature

        # Apply top-k and top-p sampling
        if top_k > 0:
```

```
            sampled_token = top_k_sampling(logits, top_k)
        else:
            sampled_token = top_p_sampling(logits, top_p)

        # Append the sampled token to the generated sequence
        generated_tokens.append(sampled_token)

        # Check if the generated token is the end-of-sequence token
        if sampled_token == tokenizer.eos_token_id:
            break
    # Decode the token ID to text
    response = tokenizer.decode(generated_tokens, skip_special_tokens=True)
    return response

# Streamlit UI
def main():
    st.title("IntellectBot")

    # Load the optimized model
    model_path = "D:\\intel\\IntelProj\\optimized_model\\model.xml"
    compiled_model = load_model(model_path)

    # Load the tokenizer
    tokenizer = AutoTokenizer.from_pretrained("TinyLlama/TinyLlama-1.1B-Chat-v1.0"
                                              )

    # Initialize session state for messages
    if "messages" not in st.session_state:
        st.session_state.messages = []

    # Display past messages
    for message in st.session_state.messages:
        with st.chat_message(message["role"]):
            st.markdown(message["content"])

    # Input field for user
    if user_input := st.chat_input("Enter your message:"):
        st.session_state.messages.append({"role": "user", "content": user_input})
        with st.chat_message("user"):
            st.markdown(user_input)

        # Generate response
        bot_response = generate_response(compiled_model, tokenizer, user_input)
        st.session_state.messages.append({"role": "assistant", "content":
                                                  bot_response})

        # Display response
        with st.chat_message("assistant"):
            st.markdown(bot_response)

if _name_ == "_main_":
    main()
```

B Github Repository

https://github.com/23Jyo/SiliconSquad