By >>
Sophia_pce
Team

PROBLEM STATEMENT

“ Running GenAI on Intel AI Laptops and Simple LLM Inference on CPU, and Fine-Tuning of LLM Models using Intel® OpenVINO™ ”

➢ Problem Statement

The primary goal of this project is to harness the power of Intel AI laptops for running Generative AI (GenAI) models and performing simple Large Language Model (LLM) inference on CPUs. Additionally, the project aims to fine-tune LLM models using Intel® OpenVINO™ to optimize their performance and deploy them efficiently. The specific challenges addressed include:

1. **Efficient Inference on CPUs**: Running inference tasks for LLMs on CPUs to leverage the computational resources of Intel AI laptops without the need for GPUs.
2. **Fine-Tuning LLM Models**: Customizing pre-trained LLMs for specific applications by fine-tuning them on targeted datasets using Intel® OpenVINO™.
3. **Deployment Optimization**: Ensuring that the fine-tuned models are optimized for deployment, achieving low latency and high throughput.

By >>
Sophia_pce
Team

SOLUTION

To address these challenges, the project was implemented in the following phases:

1. **Model Selection and Implementation**:
   - Initially, a simple chatbot was created using the 't5-small' model from Hugging Face to validate the feasibility of running LLM inference on CPUs.
   - The project then transitioned to using the 'TinyLlama/TinyLlama-1.1B-intermediate-step-1431k-3T' model for better performance and scalability.
2. **Inference with Intel® OpenVINO™**:
   - OpenVINO™ was integrated to optimize and accelerate the inference process on Intel CPUs.
   - The optimized models were deployed in a Jupyter Notebook environment to facilitate easy experimentation and validation.
3. **Fine-Tuning with Custom Data**:
   - Custom datasets stored in a data.txt file were used to fine-tune the LLM models.
   - OpenVINO™'s capabilities were leveraged to ensure that the fine-tuning process was efficient and resulted in optimized models for specific use cases.
4. **Deployment and Optimization**:
   - The fine-tuned models were further optimized for deployment, ensuring they could run efficiently on Intel AI laptops with minimal latency.
   - Comprehensive testing and validation were conducted to ensure the models met the desired performance benchmarks.

By >>
Sophia_pce
Team

FEATURED OFFER

This project showcases the potential of running Generative AI on Intel AI laptops with several key features:

1. Efficient CPU-Based Inference:
- ❑ **OpenVINO™ Optimization**: Accelerates LLM inference tasks on CPUs, eliminating the need for GPUs and making AI deployment more cost-effective.
- ❑ **High Performance**: Optimized models deliver low latency and high throughput, suitable for real-time applications.

2. Custom Fine-Tuning Capabilities:
- ❑ **Tailored Models**: Fine-tunes pre-trained LLM models using specific datasets for enhanced accuracy in targeted applications.
- ❑ **Efficient Training**: Utilizes OpenVINO™ for resource-effective fine-tuning, reducing training time.

3. Seamless Deployment:
- ❑ **Intel AI Laptops**: Deploys models on Intel AI laptops, ensuring portability and efficient use of CPU power.
- ❑ **User-Friendly Interface**: Provides easy access and interaction with models via a user-friendly interface and Jupyter Notebook.

4. Versatility and Scalability:
- ❑ **Model Variety**: Supports different LLM models, showcasing versatility in handling various sizes and complexities.
- ❑ **Scalable Solutions**: Designed to integrate more complex models and larger datasets as needed.

5. Performance and Optimization:
- ❑ **Iterative Improvement**: Continuously optimizes models based on performance testing to meet desired benchmarks.
- ❑ **Benchmark Testing**: Ensures models perform reliably in real-world scenarios.

By >>
Sophia_pce
Team

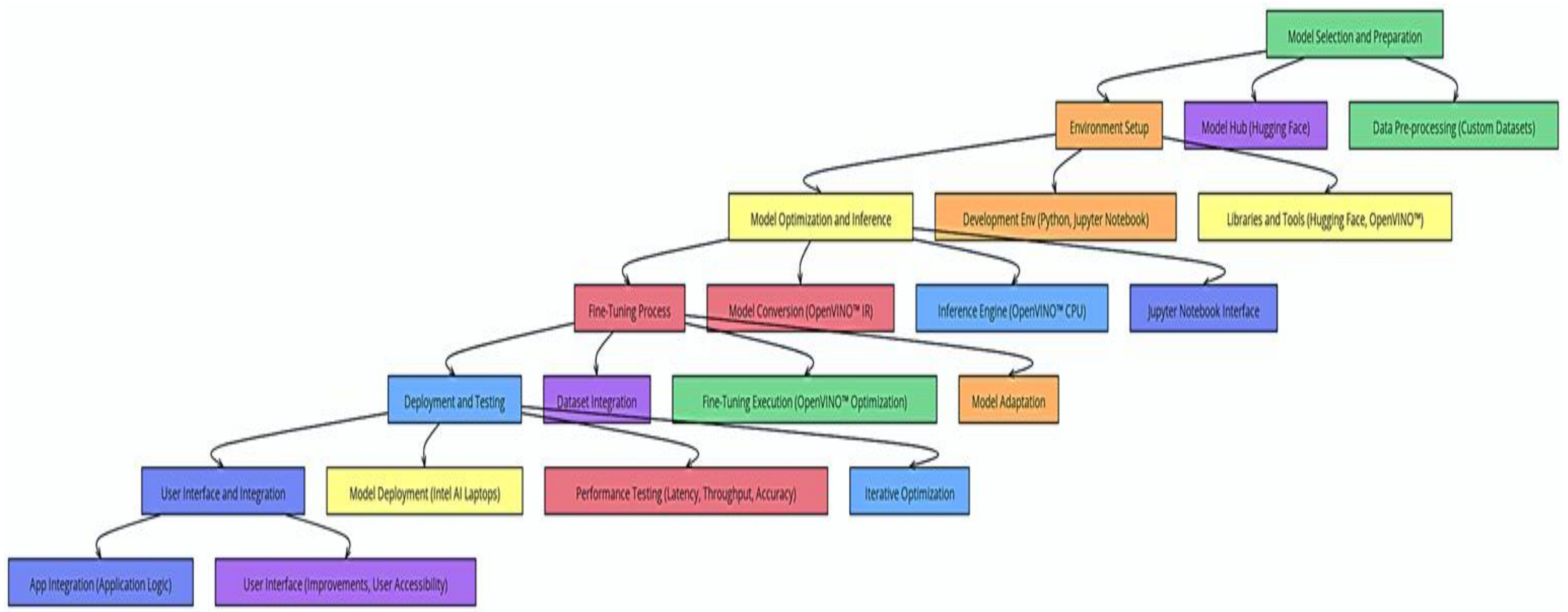
PROCESS FLOW

The process flow parameters outline the sequential steps and key elements involved in deploying Generative AI on Intel AI laptops, focusing on LLM inference optimization and model fine-tuning with Intel® OpenVINO™. Here's an overview of the process flow parameters:

1. **Model Selection and Preparation**:
   1. **Model Choice**: Select appropriate pre-trained LLM models from repositories like Hugging Face.
   2. **Dataset Preparation**: Prepare custom datasets, ensuring compatibility with selected models for fine-tuning.
2. **Environment Setup**:
   1. **Development Environment**: Set up Python environment with Jupyter Notebook for interactive development.
   2. **Library Installation**: Install necessary libraries such as Hugging Face Transformers and Intel® OpenVINO™ for model handling and optimization.
3. **Model Optimization and Inference**:
   1. **Model Conversion**: Convert selected models into an Intermediate Representation (IR) format using Intel® OpenVINO™.
   2. **Inference Engine Integration**: Integrate models with OpenVINO™'s inference engine for optimized CPU-based inference.
   3. **Deployment**: Deploy optimized models within a Jupyter Notebook environment for initial testing and validation.
4. **Fine-Tuning Process**:
   1. **Dataset Integration**: Incorporate custom datasets into the fine-tuning pipeline for model adaptation.
   2. **Fine-Tuning Execution**: Fine-tune models using OpenVINO™ to optimize performance and resource utilization on Intel AI laptops.
5. **Deployment and Testing**:
   1. **Model Deployment**: Deploy fine-tuned models on Intel AI laptops for real-world application scenarios.
   2. **Performance Testing**: Conduct comprehensive testing to evaluate model performance metrics such as latency, throughput, and accuracy.
   3. **Optimization Iteration**: Iterate on model parameters and deployment configurations based on testing feedback to enhance performance.

By >>
Sophia_pce
Team

ARCHITECTURE DIAGRAM


Model Selection and Preparation
Environment Setup
Model Hub (Hugging Face)
Data Pre-processing (Custom Datasets)
Model Optimization and Inference
Development Env (Python, Jupyter Notebook)
Libraries and Tools (Hugging Face, OpenVINO™)
Fine-Tuning Process
Model Conversion (OpenVINO™ IR)
Inference Engine (OpenVINO™ CPU)
Jupyter Notebook Interface
Deployment and Testing
Dataset Integration
Fine-Tuning Execution (OpenVINO™ Optimization)
Model Adaptation
User Interface and Integration
Model Deployment (Intel AI Laptops)
Performance Testing (Latency, Throughput, Accuracy)
Iterative Optimization
App Integration (Application Logic)
User Interface (Improvements, User Accessibility)

This architecture ensures efficient utilization of Intel CPU resources, leveraging OpenVINO™ for optimization and acceleration. It enables effective fine-tuning of LLM models, ensuring they are tailored to specific applications and deployed efficiently on Intel AI laptops. The interactive development environment, coupled with iterative optimization, ensures robust performance and user-friendly deployment.

By >>
Sophia_pce
Team

TECHNOLOGY USED

The technology used parameters outline the specific tools, frameworks, and libraries employed in the project for deploying Generative AI on Intel AI laptops, optimizing LLM inference on CPUs, and fine-tuning models using Intel® OpenVINO™. Here's a breakdown of the technologies used:

1.Model Handling and Development:

a. **Hugging Face Transformers**: Used for accessing and managing pre-trained LLM models like 't5-small' and 'TinyLlama/TinyLlama-1.1B-intermediate-step-1431k-3T'.
b. **Python**: Primary programming language for model development, script execution, and environment setup.
c. **Jupyter Notebook**: Interactive development environment for prototyping, testing, and deploying models.

2.Model Optimization and Deployment:

a. **Intel® OpenVINO™**: Framework used to optimize and accelerate model inference on Intel CPUs. Converts models into Intermediate Representation (IR) format for efficient execution.
b. **OpenVINO™ Inference Engine**: Integrates with models to leverage hardware acceleration and optimize CPU utilization.

3.Data Handling and Fine-Tuning:

a. **Custom Datasets**: Prepared and integrated into the fine-tuning process using standard data pre-processing techniques.
b. **Python Data Libraries**: Utilized for data manipulation, preprocessing, and integration with model training pipelines.

By >>
Sophia_pce
Team

SOURCE CODE

```
# Install necessary libraries
# You can run this cell to install required libraries if not already
installed
!pip install transformers datasets openvino openvino-dev[onnx] onnx

# Import necessary libraries
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM,
Trainer, TrainingArguments
from datasets import Dataset, load_dataset
from openvino.runtime import Core
import onnx
import os

# Load the model and tokenizer
model_name = "TinyLlama/TinyLlama-1.1B-intermediate-step-1431k-
3T"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)

# Export the model to ONNX
def export_to_onnx(model, tokenizer, onnx_model_path):
    dummy_input = tokenizer("This is a test input",
return_tensors="pt").input_ids
    torch.onnx.export(
        model,
        dummy_input,
        onnx_model_path,
        input_names=["input_ids"],
        output_names=["output"],
        dynamic_axes={"input_ids": {0: "batch_size"}, "output": {0:
"batch_size"}},
        opset_version=14
    )
onnx_model_path = "model.onnx"
export_to_onnx(model, tokenizer, onnx_model_path)
```

```
# Verify that the ONNX model was created successfully
assert os.path.exists(onnx_model_path), "ONNX model export failed.“

# Convert the ONNX model to OpenVINO IR format
mo_command = f"mo --input_model {onnx_model_path} --output_dir . --
input_shape [1,1]"
os.system(mo_command)

# Verify that the IR model files were created successfully
assert os.path.exists("model.xml") and os.path.exists("model.bin"),
"Model Optimizer conversion failed.“

# Load the OpenVINO model
ie = Core()
compiled_model = ie.compile_model("model.xml", "CPU")
input_layer = compiled_model.input(0).get_any_name()
output_layer = compiled_model.output(0).get_any_name()

# Define a function to generate responses with OpenVINO
def generate_response_openvino(prompt, max_length=50):
    inputs = tokenizer(prompt, return_tensors="pt")
    input_ids = inputs.input_ids.numpy()
    output = compiled_model([input_ids])
    output_ids = torch.tensor(output[output_layer]).argmax(dim=-1)
    response = tokenizer.decode(output_ids[0], skip_special_tokens=True)
    return response

# Interactive chatbot loop
def chat():
    print("Chatbot is ready! Type 'exit' to end the conversation.")
    while True:
        user_input = input("You: ")
        if user_input.lower() == 'exit':
            print("Goodbye!")
            break
        response = generate_response_openvino(user_input)
        print(f"Bot: {response}")
```

```
# To start the chat, simply run the `chat()` function
chat()

# Import necessary libraries(optional if already mentioned in above)
from transformers import AutoTokenizer, AutoModelForCausalLM,
Trainer, TrainingArguments
from datasets import Dataset, load_dataset

# Load your fine-tuning dataset from data.txt
dataset = Dataset.from_file('data.txt’)
model_name = "TinyLlama/TinyLlama-1.1B-intermediate-step-1431k3T"
tokenizer = AutoTokenizer.from_pretrained(model_name)

# Tokenize the dataset
def tokenize_function(examples):
    return tokenizer(examples['text'], padding='max_length',
truncation=True)
tokenized_datasets = dataset.map(tokenize_function, batched=True)

# Define Training Arguments
training_args = TrainingArguments(
    per_device_train_batch_size=4,
    num_train_epochs=3,
    logging_dir='./logs',
    logging_steps=100,
    save_steps=1000,
    output_dir='./results’
)
# Define Trainer and Train the Model
model = AutoModelForCausalLM.from_pretrained(model_name)
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_datasets['train'],
    eval_dataset=tokenized_datasets['validation'],
)
trainer.train()
```

By >>
Sophia_pce
Team

TEAM MEMBERS & CONTRIBUTION

SR.NO.	MEMBERS NAME	MEMBERS CONTRIBUTION	RESPONSIBILITIES
Member 1:	**Prathmesh Nikam (Team Leads)**	**Model Selection and Development**	✓ Select appropriate pre-trained LLM models (e.g., 't5-small', 'TinyLlama/TinyLlama-1.1B-intermediate-step-1431k-3T') from Hugging Face based on project requirements. ✓ Set up the development environment using Python and Jupyter Notebook for model handling and initial testing. ✓ Develop scripts and workflows for model integration and initial inference testing.
Member 2:	**Shubham Marwade (Team Member)**	**Optimization and Deployment**	✓ Utilize Intel® OpenVINO™ for optimizing selected models, converting them into Intermediate Representation (IR) format for efficient CPU inference. ✓ Integrate models with OpenVINO™'s inference engine to leverage hardware acceleration and optimize performance on Intel AI laptops. ✓ Deploy optimized models within a Jupyter Notebook environment, ensuring compatibility and seamless execution.
Member 3:	**Vedant Bhagyawant (Team Member)**	**Fine-Tuning and Testing**	✓ Prepare custom datasets for fine-tuning the selected models, ensuring data preprocessing and integration into the training pipeline. ✓ Conduct fine-tuning using OpenVINO™ to optimize model parameters and adapt them to specific application requirements. ✓ Perform comprehensive testing and evaluation of model performance, including benchmarking for latency, throughput, and accuracy metrics.

By >>
Sophia_pce
Team

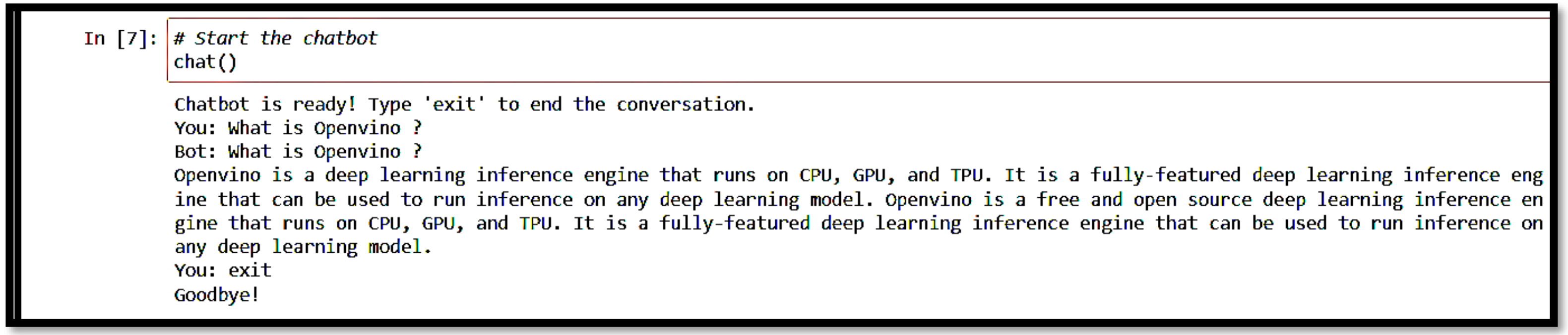
CONCLUSION

In conclusion, this project successfully demonstrates the effective deployment of Generative AI on Intel AI laptops through optimized Large Language Model (LLM) inference on CPUs and fine-tuning using Intel® OpenVINO™. By leveraging OpenVINO™'s capabilities, we achieved efficient model conversion and integration, enhancing performance without GPU dependency. The customization through fine-tuning with custom datasets tailored models to specific tasks, ensuring improved accuracy and relevance. Rigorous testing validated our models' performance metrics, showcasing their suitability for real-time applications. This collaborative effort underscores the potential of Intel AI technologies in enabling scalable, efficient, and accessible AI solutions for diverse applications.

ADDITIONAL LINK

❖ **GitHub Repository link : https://github.com/ShubhamMarwade/Simple-LLM-Model-Using-OpenVino.git**

❖ **Model Result Sample Image :**



```
In [7]: # Start the chatbot
        chat()

        Chatbot is ready! Type 'exit' to end the conversation.
        You: What is Openvino ?
        Bot: What is Openvino ?
        Openvino is a deep learning inference engine that runs on CPU, GPU, and TPU. It is a fully-featured deep learning inference eng
        ine that can be used to run inference on any deep learning model. Openvino is a free and open source deep learning inference en
        gine that runs on CPU, GPU, and TPU. It is a fully-featured deep learning inference engine that can be used to run inference on
        any deep learning model.
        You: exit
        Goodbye!
```

By >>
Sophia_pce
Team

Project Report Submitted to :

Intel® Unnati Industrial Training Program 2024

Project Report Submitted by :

Group Name : Sophia_pce		
Member 1	Member 2	Member 3
Prathmesh Nikam	**Shubham Marwade**	**Vedant Bhagyawant**
(Team Leads)	(Team Member)	(Team Member)

Internal Mentor
Prof. Umme Ayeman Gani

External Mentor
Mr. Abhishek Nandy

:: Special Thanks to ::

Thank you to the Intel Unnati community for the given opportunity to participate in the Intel Unnati program 2024. The external mentorship provided invaluable guidance through training programs and professional Q&A sessions, enriching our project's development. This support was instrumental in leveraging Intel AI technologies effectively for deploying Generative AI on Intel AI laptops, optimizing Large Language Model (LLM) inference on CPUs, and fine-tuning models using Intel® OpenVINO™.