Running GenAI on Intel AI Laptops

Abstract

This report documents the process of running Generative AI (GenAI) models on Intel AI laptops, focusing on simple language model (LLM) inference on CPU and fine-tuning LLM models using Intel OpenVINO. The experiments were conducted on the Intel Edge Developer Cloud due to compatibility and hardware constraints on local systems.

1 Technical Approach

1.1 Environment Setup

- **Platform**: Intel Edge Developer Cloud
- **Kernel**: Python 3.10 (OpenVINO Notebooks 2024.1.0)
- **CPU**: Intel(R) Xeon(R) Gold 5218R CPU @ 2.10GHz

1.2 Model Selection and Configuration

- **Inference Model**: neural-chat-7b-v3-1
- **Fine-Tuning Model**: Meta-LLama-2-7b-hf
- **Dataset**: Subset of "wikitext-2-raw-v1" from the Hugging Face datasets library

1.3 Model Optimization with OpenVINO

- **FP16 Conversion**:
  - The model was converted to FP16 format to leverage half-precision floating-point arithmetic, speeding up computations while maintaining accuracy.
- **INT8 and INT4 Compression**:
  - Explored INT8 (8-bit integer arithmetic) and INT4 (4-bit integer arithmetic) for further optimization, reducing memory usage and improving speed.

1.4 Inference Process

- **Device Selection**: Intel CPU (Xeon Gold 5218R)
- **Model Loading**: Used OVModelForCausalLM from the transformers library with OpenVINO.

- **Tokenization**: Input prompts were tokenized using AutoTokenizer from transformers.
- **Execution**: The model executed inference tasks based on input prompts, generating contextually coherent responses.

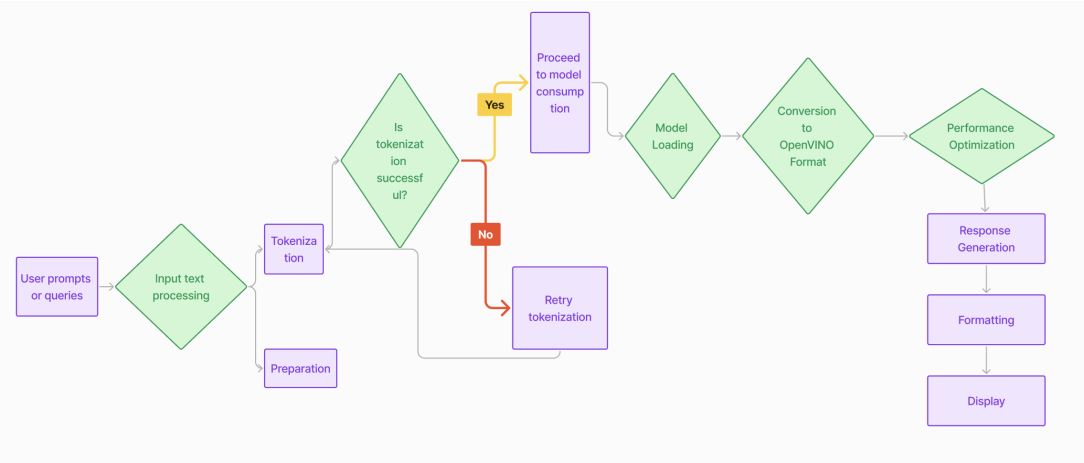

Figure 1: Inference Process

1.5 Fine-Tuning Process

- **Tokenization**: The dataset was tokenized using AutoTokenizer.
- **Training Setup**:
  - Configured training parameters: batch size, epochs, learning rate, logging.
  - Used DataCollatorForLanguageModeling for data batching during training.
- **Training**:
  - Used Hugging Face's Trainer API to train the model on the tokenized dataset.
  - Saved the trained model and tokenizer for future use.
- **Model Export to ONNX**:
  - Wrapped the model for ONNX compatibility and exported using torch.onnx.export with opset version 14.
- **OpenVINO Optimization**:
  - Converted the ONNX model to OpenVINO IR format using Model Optimizer.
  - Loaded the optimized model with OpenVINO Inference Engine, adjusted CPU configuration.
- **Inference and Benchmarking**:
  - Encoded input prompts, executed inference, and processed output logits.
  - Measured inference time and throughput.

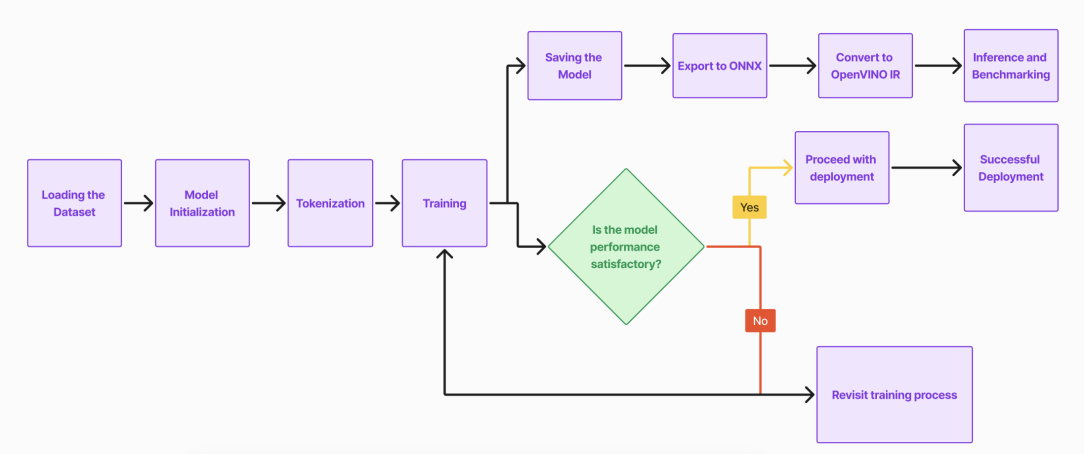

Figure 2: Fine-Tuning Process

2 Issues Faced

2.1 Local System Constraints

- **Library Incompatibility**: Some libraries required for model inference and fine-tuning were not compatible with the local system.
- **OS Support**: The local OS did not support certain dependencies, causing execution failures.

2.2 Intel Edge Developer Cloud Constraints

- **Interface Limitations**: Unable to use interfaces like Gradio and Voila.
- **Time Constraint**: The platform imposed a one-hour limit, preventing complete fine-tuning.
- **Reduced Epochs**: Fine-tuning was conducted with fewer epochs due to time limitations.

3 Results

3.1 Inference Performance (neural-chat-7b-v3-1)

- **Accuracy**: High accuracy with some prompts achieving 1.00 accuracy.
- **Inference Time**: Efficient inference with good response times.

3.2 Fine-Tuning Performance (Meta-LLama-2-7b-hf)

3.2.1 Training Configuration

```
training_args = TrainingArguments(
    output_dir="./new",
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
```

```
    num_train_epochs=1,
    evaluation_strategy="no",
    save_strategy="epoch",
    logging_dir="./old",
    logging_steps=10,
    learning_rate=5e-5,
    weight_decay=0.01,
    save_total_limit=1,
    load_best_model_at_end=False,
)
```

3.2.2 Benchmark Results

- **Inference Time**: 0.8281 seconds
- **Throughput**: 1.21 sequences per second

4 Benefits of OpenVINO Integration

4.1 Performance Enhancement

- **Optimized Execution**: The conversion to FP16, INT8, and INT4 formats significantly improved the speed and efficiency of model execution.
- **Low Latency**: OpenVINO's Inference Engine is designed for low-latency performance, making it suitable for real-time applications.

4.2 Resource Efficiency

- **CPU Utilization**: OpenVINO optimizes the model for execution on Intel CPUs, reducing the need for expensive GPU resources.
- **Scalability**: The optimized models can be scaled across different Intel hardware, including CPUs, integrated GPUs, and VPUs, providing flexibility in deployment.

4.3 Cost-Effectiveness

- **Existing Hardware**: Utilization of existing Intel CPU resources eliminates the need for additional hardware investment.
- **Broad Deployment**: Efficient inference on widely available hardware facilitates the deployment of AI applications in diverse environments.

5 Conclusion

Running GenAI models on Intel AI laptops using OpenVINO demonstrated significant improvements in inference performance and efficiency. Despite local system constraints, the Intel Edge Developer Cloud provided a viable platform for conducting these experiments. OpenVINO's optimizations, including FP16, INT8, and INT4 formats, proved effective in enhancing model performance on Intel CPUs.

The experiments highlighted the potential for deploying AI models on existing hardware infrastructure, offering a cost-effective and scalable solution for various AI-driven applications. Future work will focus on further optimizations, real-world application integration, and comprehensive benchmarking across different models and hardware configurations.

6 Future Directions

- **Further Optimization**: Explore additional OpenVINO optimizations and fine-tuning techniques to maximize performance.
- **Application Development**: Integrate the optimized model into real-world applications for conversational AI and other language-driven tasks.
- **Benchmarking**: Conduct comparative studies to evaluate performance metrics across different models and hardware configurations.