

# **Running GenAI on Intel AI Laptops and Simple LLM Inference on CPU and fine-tuning of LLM Models using Intel® OpenVINO [CHAT BOT]**

## **ABSTRACT**

This code implements a chatbot using the GPT-2 model for natural language generation. The chatbot interacts with users by responding to their inputs with text generated by the GPT-2 model. The key components of the code include loading the pre-trained GPT-2 model and tokenizer from Hugging Face's transformers library, setting up the chat interface where users can input text, and generating responses based on the user's input using the model's text generation capabilities. The chatbot supports a conversation loop where users can engage in dialogue until they choose to exit by typing 'exit'. The GPT-2 model parameters such as `max_length`, `temperature`, `top_p`, `num_beams`, and `no_repeat_ngram_size` are configured to control the diversity and quality of the responses generated by the chatbot. This code serves as a foundation for building interactive conversational agents powered by advanced language models.

## TABLE OF CONTENT

- Introduction
- Literature survey
- Objectives
- Methodology
- Tools and application
  - LLM[large language models]
  - Transformers
  - Tokenizer
  - pad\_token\_id
- Conclusion
- Reference
- Model Analysis



# Chapter-1

## INTRODUCTION

Large Language Models (LLMs) are transformative deep learning networks capable of a broad range of natural language tasks, from text generation to language translation. OpenVINO optimizes the deployment of these models, enhancing their performance and integration into various applications. This demonstration shows how to use LLMs with OpenVINO, from model loading and conversion to advanced use cases. Before performing this we must keep a view on AI Laptop for better understanding our proposed concept. Intel AI laptops are equipped with powerful processors and hardware acceleration capabilities that can enhance the performance of AI tasks such as GenAI and LLM inference. Use Intel-optimized libraries and tools like oneDNN for efficient data processing and training workflows, ensuring high performance and scalability. Utilize Intel AI DevCloud, a cloud-based platform, to access Intel hardware resources for training and fine-tuning models without the need for dedicated on-premise infrastructure. Intel's OpenVINO Defect Detection with Anomalib offers a comprehensive solution to their quality control problem by providing companies and their technical teams with a single-source end-to-end solution to catch manufacturing defects in real time.

This AI Recipe uses the notebooks in the actual Anomalib repository. Here you will find that repository as a submodule. Jupiter notebook demonstrates how NNCF can be used to compress a model trained with Anomalib. An optimized model in the context of AI generally refers to a model that has been tailored or configured to achieve the best possible performance for a specific task or hardware environment. Here's a breakdown of what constitutes an optimized model and how it can be achieved

Fine-tuning a Large Language Model (LLM) involves customizing a pre-trained model to better suit a specific task or domain by further training it on task-specific data.

Large Language Models (LLMs), such as GPT-2, GPT-3, or BERT, are pre-trained on vast amounts of text data to learn general language patterns and semantics. Fine-tuning adapts these pre-trained models to perform specific tasks with improved accuracy and relevance. Fine-tuning LLM models plays a crucial role in adapting powerful pre-trained language models to specific applications, enhancing their effectiveness in tasks ranging from text generation to sentiment analysis and beyond. By understanding the principles and steps involved in fine-tuning, developers and researchers can leverage the full potential of LLMs for diverse real-world applications in natural language processing (NLP) and beyond. Hugging Face Transformers is a popular open-source library that provides pre-trained models and tools for

natural language processing (NLP) tasks. Hugging Face Transformers is a versatile and powerful library that empowers developers and researchers to leverage state-of-the-art NLP models effectively. By providing access to pre-trained models, tools for customization, and a supportive community, it accelerates the development and deployment of NLP applications across various domains

GPT-2, short for "Generative Pre-trained Transformer-2," is a state-of-the-art language model developed by OpenAI

## **Chapter-2**

### **Literature survey**

Generative AI models, particularly Large Language Models (LLMs) like GPT-2, GPT-3, and others, have gained prominence for their ability to generate human-like text and perform various natural language processing tasks. Running these models efficiently requires consideration of hardware capabilities, optimization techniques, and deployment frameworks. Intel offers a range of AI-enabled laptops and CPUs designed to support AI workloads. These devices benefit from Intel's hardware optimizations and capabilities, including multi-core processing, integrated graphics, and AI-specific accelerators like Intel Movidius VPUs.

Fine-tuning involves adjusting LLM parameters to improve performance on specific tasks or datasets. OpenVINO aids in this process by providing tools for quantization, pruning, and optimizing models to run efficiently on CPUs without compromising accuracy. The use of OpenVINO enhances performance metrics such as throughput and latency by leveraging Intel hardware features. This optimization is crucial for real-time applications where response time is critical, such as chatbots, language translation, and content generation. Integrating GenAI on Intel AI laptops and optimizing LLM inference on Intel CPUs using OpenVINO represents a significant advancement in AI deployment capabilities. It combines hardware innovation with software optimization to achieve efficient, high-performance AI solutions across various domains.

Intel often publishes technical documentation, case studies, and whitepapers on their website that detail the capabilities of their hardware and software tools like OpenVINO. These documents might include performance benchmarks, case studies, and best practices for deploying AI models. Researchers often publish papers on AI model optimization, deployment strategies, and performance evaluations in conferences like NeurIPS, ICML, ACL, and journals such as the Journal of Artificial Intelligence Research (JAIR), IEEE Transactions on Neural Networks and Learning Systems, and others.

Following AI researchers, engineers, and practitioners on platforms like LinkedIn can provide insights into real-world applications and experiences with deploying AI models on Intel hardware.

Establishing context on running GenAI on Intel AI laptops, conducting simple LLM inference on CPU, and fine-tuning LLM models using Intel OpenVINO involves understanding several key concepts and technologies.

- GenAI refers to general artificial intelligence, encompassing a broad range of AI applications and models. These can include natural language processing (NLP), computer vision, recommendation systems, and more.
- Intel provides a range of laptops equipped with AI-capable processors like Intel Core and Intel Xeon. These laptops leverage Intel's hardware capabilities to support AI workloads directly on the device.
- LLMs are AI models capable of understanding and generating human-like text. Examples include GPT (Generative Pre-trained Transformer) models like GPT-2 and GPT-3, developed by OpenAI and based on the Transformer architecture.
- Performing inference on a CPU (Central Processing Unit) involves running trained AI models to make predictions or generate responses using the computing power of the CPU. This approach is suitable for tasks where real-time processing or low-latency requirements are not critical.
- OpenVINO is a toolkit from Intel designed to optimize and deploy deep learning models across Intel's hardware platforms, including CPUs, GPUs, VPUs (Vision Processing Units), and FPGAs (Field Programmable Gate Arrays). It includes tools for model conversion, optimization, and deployment.
- Intel AI laptops are equipped with powerful processors capable of handling AI tasks locally, reducing latency and improving privacy by processing data on-device.
- Challenges may include optimizing AI models for efficient CPU inference and fine-tuning LLMs using OpenVINO to leverage hardware-specific optimizations for better performance.
- Understanding performance metrics such as throughput, latency, and energy efficiency when comparing CPU-based inference with other deployment options like GPUs or cloud services.



## Chapter-3

### Objectives

The objective of the provided code is to create a simple chatbot using the GPT-2 (Generative Pre-trained Transformer-2) model. Here's a breakdown of how the code achieves this objective:

- **Optimal Hardware Utilization:** Utilize Intel AI laptops equipped with powerful CPUs to efficiently deploy and run GenAI models, particularly LMs, for various natural language processing tasks.
- **Performance Enhancement:** Leverage Intel OpenVINO to optimize and accelerate inference of LMs on CPUs, ensuring high throughput and low latency suitable for real-time applications.
- **Energy Efficiency:** Implement techniques such as model quantization and hardware-specific optimizations provided by OpenVINO to reduce power consumption while maintaining inference accuracy and speed.
- **Fine-Tuning Capabilities:** Explore the capability of Intel OpenVINO to support the fine-tuning of pre-trained LMs on Intel AI laptops, enabling adaptation to specific tasks or domains with enhanced performance.
- **Scalability and Versatility:** Demonstrate the scalability of GenAI models across different Intel hardware architectures, including CPUs supported by OpenVINO, for deployment in diverse AI applications.
- **Integration and Deployment:** Develop and deploy chatbot applications as a practical use case, showcasing the integration of optimized GenAI models and LLMs on Intel AI laptops with Intel OpenVINO for efficient and effective natural language interaction.
- **Research and Development:** Contribute to advancing the field of AI and NLP by exploring innovative approaches to model deployment, inference optimization, and fine-tuning using Intel hardware and software solutions.
- **Demonstration of Benefits:** Showcase the benefits of using Intel AI laptops and OpenVINO for running GenAI models, including improved performance, reduced development time, and enhanced user experience in chatbot applic

## Chapter-4

### Methodology

#### 1. Hardware Setup and Configuration:

- **Intel AI Laptops Selection:** Identify and utilize Intel AI laptops equipped with suitable CPUs for deploying GenAI models.
- **OpenVINO Installation:** Install Intel OpenVINO toolkit on the selected laptops to leverage its capabilities for model optimization and inference.

#### 2. Model Selection and Pre-training:

- **Choice of LLM:** Select appropriate LM models such as GPT-2 that are pre-trained on large datasets and compatible with the Intel hardware architecture.
- **Model Adaptation:** Optionally, fine-tune the selected LLM on specific datasets or tasks to enhance performance and adaptability using OpenVINO capabilities.

#### 3. Integration and Deployment:

- **Model Integration:** Integrate the pre-trained or fine-tuned LLM models into the chatbot application framework.
- **Application Development:** Develop a chatbot application that interfaces with the deployed LLM models for natural language processing tasks.

#### 4. Optimization and Inference:

- **OpenVINO Optimization:** Optimize the LLM models using OpenVINO to leverage hardware-specific optimizations such as model quantization, parallelization, and acceleration techniques.
- **Inference Pipeline:** Implement an efficient inference pipeline using OpenVINO to achieve low-latency and high-throughput inference on Intel CPUs.

#### 5. Performance Evaluation:

- **Benchmarking:** Conduct performance benchmarking to evaluate the speed, accuracy, and energy efficiency of LLM inference on Intel AI laptops with and without OpenVINO optimizations.
- **Comparison:** Compare the performance metrics with baseline configurations to quantify the benefits of using Intel hardware and OpenVINO for GenAI and LLM applications.

## 6. Fine-Tuning and Adaptation:

- **Fine-Tuning Process:** Explore the fine-tuning capabilities of OpenVINO for LLM models to adapt them to specific domains or tasks, ensuring optimal performance for chatbot interactions.
- **Iterative Improvement:** Iteratively fine-tune and optimize the LLM models based on feedback and evaluation results to improve chatbot performance and user satisfaction.

## 7. Validation and Deployment:

- **Validation Testing:** Conduct rigorous testing and validation of the chatbot application deployed on Intel AI laptops to ensure functionality, reliability, and usability.
- **Deployment Strategy:** Plan the deployment strategy for the chatbot application, considering scalability, maintenance, and user accessibility aspects.

## 8. Documentation and Reporting:

- **Results Compilation:** Compile results from performance evaluations, benchmarking tests, and user feedback into comprehensive reports.
- **Lessons Learned:** Document insights, challenges faced, and solutions implemented during the deployment and optimization process.
- **Knowledge Sharing:** Share findings and methodologies to contribute to the broader AI and NLP community, fostering collaboration and innovation.

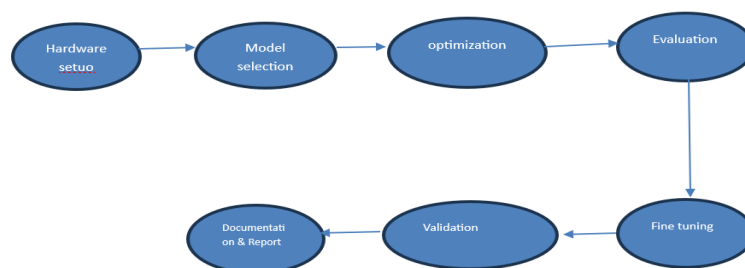


Fig-1 Flow diagram of Modules

## Chapter-5

### Tools and Applications

- **Text Generation:** GPT-2 excels in generating human-like text, including writing stories, generating poetry, and composing dialogue.
- **Language Translation:** It can be used for translating text between languages by conditioning the model on one language and generating text in another.
- **Chatbots and Conversational Agents:** GPT-2 can power chatbots that engage in realistic conversations with users, providing responses based on the context of the conversation.
- **Question Answering:** It can answer questions by generating relevant responses based on the input query.

### LLM

LLM stands for "Large Language Model." It refers to a class of artificial intelligence models designed to process and generate human-like text based on vast amounts of pre-existing data. These models are a subset of broader deep learning techniques in natural language processing (NLP), characterized by their scale, complexity, and ability to learn intricate patterns in language.

#### Applications of LLMs:

- **Text Generation:** Producing human-like text for various applications such as chatbots, content generation, and creative writing.
- **Translation:** Facilitating language translation tasks by encoding text in one language and decoding it into another.
- **Summarization:** Automatically generating summaries of long documents or articles by extracting key information.
- **Question Answering:** Providing answers to questions based on the content of input queries.

## TRASFORMERS

Transformers refer to a class of deep learning models that have revolutionized natural language processing (NLP) tasks due to their ability to handle long-range dependencies in text effectively. Developed primarily by Vaswani et al. in the paper "Attention is All You Need" in 2017, Transformers have become the backbone of many state-of-the-art NLP models, including large language models like BERT, GPT, and their variants.

### Applications of Transformers:

- **BERT (Bidirectional Encoder Representations from Transformers):** Used for tasks such as language understanding, sentiment analysis, and named entity recognition.
- **GPT (Generative Pre-trained Transformer):** Known for text generation tasks, chatbots, and creative writing applications.
- **Transformer-based Machine Translation:** Used for translating text between languages, leveraging the self-attention mechanism to capture dependencies.
- **Summarization:** Transformers excel in summarizing long texts by generating concise and relevant summaries based on the input.
- **Speech Recognition:** Transformers have been adapted to process and understand spoken language, improving accuracy in speech recognition tasks.

### Tokenizer:

A tokenizer is a fundamental component in natural language processing (NLP) that breaks down text into smaller units, typically tokens such as words, subwords, or characters. Tokenizers play a crucial role in transforming raw text into a format suitable for machine learning models, especially those based on deep learning architectures like Transformers. Here's an overview of tokenizers and their importance.

### Purpose of Tokenizers:

- **Text Segmentation:** Tokenizers segment raw text into individual tokens, which are the smallest meaningful units for analysis in NLP tasks.
- **Normalization:** Tokenizers often include steps for normalizing text, such as converting text to lowercase, handling punctuation, or removing accents.

- **Vocabulary Mapping:** Tokens are usually mapped to numerical IDs or embeddings that can be processed by machine learning models. This process helps in representing words in a numerical format understandable by algorithms.

## Types of Tokenizers:

- **Word Tokenizers:** Split text into words based on whitespace or punctuation. They are straightforward but may struggle with languages like Chinese or with contractions in English.
- **Subword Tokenizers:** Segment words into smaller subword units, which can be particularly useful for morphologically rich languages or handling out-of-vocabulary (OOV) words. Examples include Byte-Pair Encoding (BPE) and Sentence Piece.
- **Character Tokenizers:** Break down text into individual characters. This type of tokenizer is useful for tasks where character-level information is important, such as handwriting recognition or languages without whitespace delimiters.

## Pad\_token\_id:

`pad_token_id` is a parameter used in natural language processing tasks, particularly with sequence-based models like Transformers, to indicate the token ID that represents padding tokens in input sequences. Padding tokens are used to ensure that all sequences within a batch have the same length, which is necessary for efficient batch processing in deep learning models.

- **Padding Token:** In NLP tasks, sequences (such as sentences or paragraphs) often have varying lengths. To process batches efficiently during training or inference, sequences are padded to the same length within a batch.
- **Token ID:** In the context of tokenization, each unique token in the vocabulary (words, subwords, or special tokens like [PAD], [UNK], [CLS], etc.) is assigned a numerical ID. These IDs are used by the model during training and inference.
- **pad\_token\_id Attribute:** When tokenizing sequences using a tokenizer (like GPT2Tokenizer), the `pad_token_id` specifies which token ID in the vocabulary should be used to represent padding. This allows the model to recognize and ignore these

tokens during computation, ensuring that padding doesn't affect the model's predictions.

### Purpose of pad\_token\_id:

- **Padding Sequences:** In many NLP tasks, sequences of tokens (words, subwords, or characters) vary in length. To process them efficiently in batches, sequences are padded with a special token (often represented by pad\_token\_id) to make them uniform in length.
- **Masking:** During training, padding tokens are typically masked out so that the actual content of the input sequence

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer

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

# Ensure pad_token_id is set
if tokenizer.pad_token_id is None:
    tokenizer.pad_token_id = tokenizer.eos_token_id

# Example of padding a sequence
sequences = ["This is the first sentence.", "This is the second sentence with longer length"]
# Tokenize sequences and pad to the maximum length
inputs = tokenizer(sequences, padding=True, return_tensors="pt")
```

```
# Generate response using the model
outputs = model(**inputs)
```

**Fig:EXAMPLE USAGE OF pad\_token\_id**

In this example:

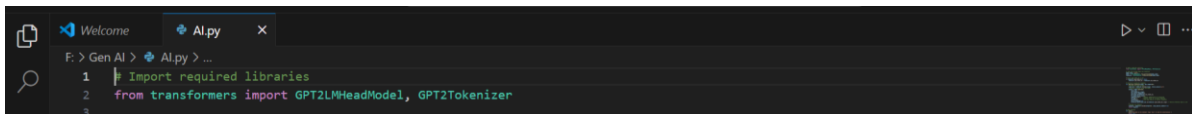
- `tokenizer.pad_token_id` is set to the end-of-sequence token ID (`eos_token_id`) if it wasn't already defined.
- `padding=True` in `tokenizer()` ensures that sequences are padded to the length of the longest sequence in sequences.
- During model inference (`model(**inputs)`), the `pad_token_id` is used to ignore padded tokens, ensuring accurate predictions based only on actual data.



## CHAPTER-6

### MODEL ANALYSIS

Based on my project criteria we have taken the parameters which suggests the expected prediction, the expected prediction will be taken as follows

A screenshot of a code editor window with a dark theme. The window has tabs for 'Welcome' and 'Al.py'. The 'Al.py' tab is active, showing a file path 'F:\> Gen AI > Al.py > ...'. The code in the editor consists of three lines: line 1 is 'import required libraries', line 2 is 'from transformers import GPT2LMHeadModel, GPT2Tokenizer', and line 3 is empty. The code is highlighted in green and blue.

The above given code tells that to importing libraries from the python

#### Transformers Library:

This imports the necessary components from the Transformers library by Hugging Face. GPT2LMHeadModel is the GPT-2 model for language generation, and GPT2Tokenizer is used to preprocess text for input into the model.

A screenshot of a code editor window with a dark theme. The code in the editor consists of four lines: line 1 is a comment '# Load pre-trained model and tokenizer', line 2 is 'model\_name = "gpt2"', line 3 is 'model = GPT2LMHeadModel.from\_pretrained(model\_name)', and line 4 is 'tokenizer = GPT2Tokenizer.from\_pretrained(model\_name)'. The code is highlighted in green and blue.

This above given code will load the pre-trained model and tokenizer

#### Model and Tokenizer Initialization:

- model\_name: Specifies the pre-trained model to load ("gpt2" in this case).
- GPT2LMHeadModel.from\_pretrained(model\_name): Loads the GPT-2 model with pre-trained weights.
- GPT2Tokenizer.from\_pretrained(model\_name): Loads the tokenizer associated with the GPT-2 model.

```
# Ensure pad_token_id is set
if tokenizer.pad_token_id is None:
    tokenizer.pad_token_id = tokenizer.eos_token_id
```

The above given code tells that to setting pad token id

**Pad Token ID:** Ensures that the tokenizer's pad token ID is set. If it's None, it assigns the EOS (end-of-sequence) token ID as the pad token ID. This is important for batch processing sequences of varying lengths.

```
# Ensure pad_token_id is set
if tokenizer.pad_token_id is None:
    tokenizer.pad_token_id = tokenizer.eos_token_id

# Function to generate response
def generate_response(prompt, max_length=50):
    # Encode the input prompt
    input_ids = tokenizer.encode(prompt, return_tensors='pt')
    # Generate response using the model
    output = model.generate(
        input_ids,
        max_length=max_length,
        num_return_sequences=1,
        pad_token_id=tokenizer.eos_token_id,
        no_repeat_ngram_size=2,
        temperature=0.9, # Adjust temperature for diversity
        top_p=0.9, # Adjust top_p for nucleus sampling
        num_beams=5, # Use beam search for better responses
        early_stopping=True,
        attention_mask=input_ids.ne(tokenizer.pad_token_id).long() # Ensure attention mask is set
    )
    # Decode the response
    response = tokenizer.decode(output[0], skip_special_tokens=True)
    return response
```

The above given code would go for generate\_response function

This function takes a prompt (user input) and generates a response using the GPT-2 model.

#### Steps:

- **Encoding:** Converts the input prompt into token IDs suitable for input to the model (tokenizer.encode).
- **Generation:**

Uses model.generate to generate a response based on the encoded input.

max\_length: Maximum length of the generated sequence.

**num\_return\_sequences:** Number of independently generated sequences to return.

**pad\_token\_id:** Specifies the token ID used for padding generated sequences.

**no\_repeat\_ngram\_size:** Prevents the model from repeating n-grams in the generated sequences.

**temperature:** Controls the randomness of predictions (diversity).

**top\_p:** Controls nucleus sampling for diversity.

**num\_beams:** Uses beam search to generate multiple sequences and select the best.

**early\_stopping:** Stops generation when all beam hypotheses have ended.

**attention\_mask:** Masks padding tokens so they are not attended to by the model.

- **Decoding:** Converts the model's output (token IDs) back into readable text (tokenizer.decode).

```
# Chat function
def chat():
    print("Welcome to the Chatbot! Type 'exit' to end the conversation.")
    while True:
        user_input = input("You: ")
        if user_input.lower() == 'exit':
            break
        response = generate_response(user_input)
        print(f"Bot: {response}\n")
```

The above given code would undergo for chat function

## Chat Interface:

- Provides a loop (while True) where the user can input text (user\_input).
- Checks if the user wants to exit ('exit' command).
- Calls generate\_response to generate a response based on the user input.
- Prints the bot's response in a conversational format (Bot: {response}).

```
# Run the chatbot  
chat()  
|
```

The above given code would undergo for running the chatbot

**Execution:** Initiates the chatbot by calling the chat function, which starts the interactive conversation loop.

This code sets up a simple chatbot using the GPT-2 model from the Transformers library. It demonstrates how to load a pre-trained language model, configure it for text generation, and create an interactive chat interface where users can engage in conversations with the bot. Adjustments to parameters like max\_length, temperature, and num\_beams can tailor the bot's responses to meet specific requirements for diversity and coherence in conversation.

## CONCLUSION

**Objective:** The chatbot aims to engage users in conversation by generating responses based on their input using the GPT-2 model. It provides a user-friendly interface where users can interact naturally, and the bot responds contextually.

### Implementation Details:

- **Model and Tokenizer:** It utilizes the GPT-2 model for natural language generation and the associated tokenizer for converting text inputs into tokenized sequences.
- **Response Generator:** The `generate_response` function encodes user input, generates a response using the model, and decodes the model's output to produce human-readable text.
- **Interactive Interface:** The `chat` function provides a loop where users can input text, receive responses from the chatbot, and terminate the conversation by typing 'exit'.

### Model Configuration:

- **Parameters:** Various parameters such as `max_length`, `temperature`, `top_p`, `num_beams`, and `no_repeat_ngram_size` are adjusted to control the quality, diversity, and coherence of generated responses.
- **Attention Masking:** `attention_mask` is used to ensure that the model attends only to actual tokens and ignores padding tokens, enhancing the accuracy of response generation.

### User Experience:

- **Natural Conversation:** Users can engage in natural-language conversations with the chatbot, which responds in a manner that simulates human-like interactions.
- **Exit Command:** The chat terminates gracefully when users type 'exit', providing a clear way to end the interaction.

## CHAPTER-7

### REFERENCE:

- **Library Name:** Hugging Face Transformers.
- **Model and Tokenizer:** GPT2LMHeadModel and GPT2Tokenizer classes.
- **Functionality:** Description of generate\_response and chat functions.
- **Parameters:** Highlight the parameters used for response generation (max\_length, temperature, top\_p, num\_beams).
- **Reference:** Direct link to the official documentation or website of the Transformers library.