



Optimising LLM-Powered Chatbot Performance

A Comparative Analysis of Model Compression Strategies and Functional
Enhancements Using OpenVINO

Contents

1. Introduction	2
2. Objective	3
3. Model Selection and Optimization	4
4. Functional Enhancements	9
5. Benchmarks	11
6. Conclusion	12
7. Code Structure and Model	14
8. References	15

1. Introduction

The landscape of artificial intelligence (AI) and machine learning (ML) has been significantly enriched by the advent of open-source LLM models. Models such as LLaMA-2 and MISTRAL-7B, among others, have become pivotal in advancing the capabilities of chatbot applications. These open-source LLMs (Large Language Models) enable developers and researchers to create chatbots that offer highly engaging, context-aware interactions with users, democratising access to cutting-edge natural language processing technologies.

Open-source models derived from LLAMA-2 are at the heart of this transformative wave, providing the foundational technology for chatbots that can understand and generate human-like text. This ability to process and respond to natural language inputs in a contextually relevant manner has broadened the application scope of chatbots, extending from customer support systems to virtual assistants and interactive entertainment.

1.1 The Critical Role of Model Optimization

Optimising open-source LLMs for chatbot applications is vital for several key reasons:

1. **Enhanced Performance:** Optimization helps in fine-tuning these models to deliver faster and more accurate responses, ensuring that chatbot interactions are smooth and natural, which is essential for maintaining user engagement and satisfaction.
2. **Efficient Resource Use:** Through optimization, the computational footprint of LLMs can be significantly reduced. This allows for more cost-effective deployments and makes it possible to run sophisticated chatbots on a wider range of hardware, including devices with limited processing capabilities.
3. **Scalability and Accessibility:** Optimised models can serve a larger number of requests with lower latency, making it possible to scale chatbot services efficiently. This scalability is crucial for applications expecting high volumes of interactions and ensures that advanced chatbot capabilities are accessible to a broader audience.
4. **Sustainability:** In an era where the environmental impact of digital technologies is increasingly scrutinised, optimising LLMs for lower energy consumption is a step towards more sustainable AI practices. Reduced computational requirements mean less energy consumed per interaction, aligning with broader goals of reducing the carbon footprint of AI technologies.

1.2 User Experience at the Forefront

The optimization of open-source LLMs directly contributes to enhancing the user experience. By improving the speed, accuracy, and contextual awareness of chatbot responses, users enjoy more meaningful and efficient interactions. In the open-source ecosystem, where community contributions and collaborations are key, ongoing optimizations and improvements ensure that these models continue to set benchmarks for what is possible in AI-driven communication.

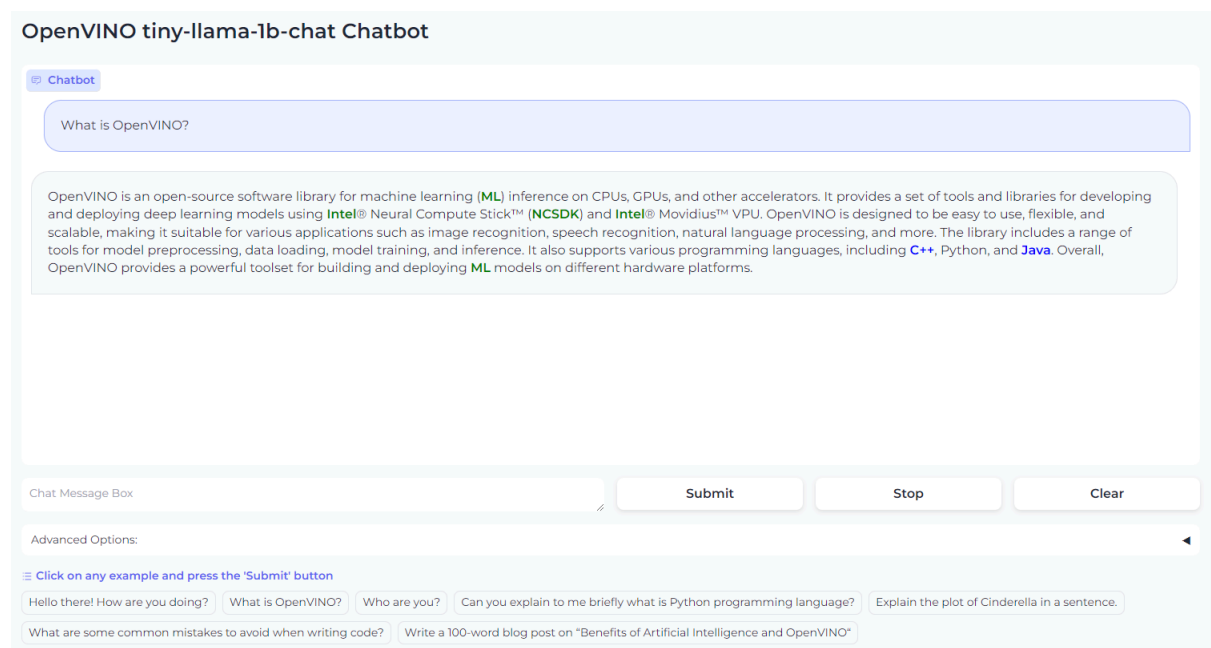
In summary, open-source LLMs have catalysed a significant shift in chatbot technologies, offering sophisticated tools for creating highly interactive and responsive chatbots. Optimising these models not only addresses practical considerations of performance and resource use but also elevates the overall user experience, ensuring that chatbots can fulfil their promise as engaging, intelligent conversational partners.

2. Objective

The primary objective of this project is to showcase advanced technical capabilities in enhancing the performance and functionality of an LLM-powered chatbot application through strategic optimization and benchmarking within the OpenVINO framework. By undertaking a comprehensive set of tasks, the project aims to demonstrate expertise in the following areas:

1. **Model Selection and Optimization:** To select an appropriate LLM model from among specified options (e.g., tiny-llama-1b-chat, red-pajama-3b-chat, llama-2-7b-chat) based on a set of criteria such as performance potential and application suitability. The task involves converting the chosen model to the OpenVINO Intermediate Representation (IR) format and exploring various compression strategies (4-bit and 8-bit) to understand and articulate their trade-offs in terms of model efficiency and accuracy.
2. **Chatbot Functional Enhancement:** To implement a novel feature within the chatbot application that significantly improves its utility and enhances user experience. This enhancement should not only demonstrate innovative thinking but also technical proficiency in integrating new functionalities into existing systems.
3. **Benchmarking:** To conduct rigorous tests comparing the original and optimised models across different hardware setups (CPU/GPU), focusing on key performance indicators such as inference speed and memory usage. The objective is to provide a detailed analysis of the optimization's impact, showcasing the ability to conduct meaningful benchmarking and draw insightful conclusions from the data.

The ultimate goal is to leverage this project as a platform to demonstrate a deep understanding of LLM-powered chatbot optimization, an ability to enhance chatbot functionalities creatively, and a proficiency in benchmarking and analytical reporting. Through this endeavour, the aim is to contribute valuable insights and methodologies to the field, showcasing a blend of technical expertise and innovative problem-solving skills.



OpenVINO tiny-llama-1b-chat Chatbot

Chatbot

What is OpenVINO?

OpenVINO is an open-source software library for machine learning (ML) inference on CPUs, GPUs, and other accelerators. It provides a set of tools and libraries for developing and deploying deep learning models using Intel® Neural Compute Stick™ (NCSDK) and Intel® Movidius™ VPU. OpenVINO is designed to be easy to use, flexible, and scalable, making it suitable for various applications such as image recognition, speech recognition, natural language processing, and more. The library includes a range of tools for model preprocessing, data loading, model training, and inference. It also supports various programming languages, including C++, Python, and Java. Overall, OpenVINO provides a powerful toolset for building and deploying ML models on different hardware platforms.

Chat Message Box

Submit Stop Clear

Advanced Options:

Click on any example and press the 'Submit' button

Hello there! How are you doing? What is OpenVINO? Who are you? Can you explain to me briefly what is Python programming language? Explain the plot of Cinderella in a sentence.

What are some common mistakes to avoid when writing code? Write a 100-word blog post on "Benefits of Artificial Intelligence and OpenVINO"

Above is the final output of the LLM powered chatbot with functional enhancements.

3. Model Selection and Optimization

3.1 Model Selection

After a thorough evaluation of state-of-the-art models for developing an optimised LLM-powered chatbot application, the TinyLlama-1.1B-Chat-V1.0 model was chosen. This decision was influenced by several critical factors that align with the project's objectives and constraints, particularly regarding GPU memory limitations and the need for efficient processing.

1. **GPU Memory Limitations and Efficiency:** TinyLlama, with its compact design featuring only 1.1 billion parameters, is specifically optimised for scenarios with strict GPU memory constraints. This quality makes it exceedingly suitable for maintaining rapid response times and minimising computational overhead, especially in environments with limited hardware capabilities such as mobile applications, IoT devices, and edge computing scenarios.
2. **Performance and Resource Consumption Balance:** Despite its smaller size compared to models like Llama-2-7b-chat, TinyLlama offers a commendable balance between conversational performance and resource efficiency. This balance is crucial for delivering quality interactions without the substantial resource demands associated with larger models. Its architecture and tokenizer are exactly the same as those of Llama 2, ensuring high compatibility and ease of integration into existing Llama-based projects.
3. **Optimization, Efficiency, and Broad Applicability:** Emphasising optimization within the OpenVINO framework, TinyLlama's design is conducive to significant optimization efforts. This allows for substantial improvements in performance and efficiency without notable degradation in conversational quality. Its adaptability across various hardware configurations and the potential for streamlined deployment across different platforms highlight its broad applicability.
4. **Commonsense Reasoning and Token Training:** With commonsense reasoning capabilities marked at 52.99 and training on 3 trillion tokens, TinyLlama stands out for text analysis and data extraction in scenarios requiring quick and efficient processing. This includes analysing customer feedback, survey responses, or social media interactions. The model's extensive training surpasses that of similar models like RedPajama, which is trained with 800 billion tokens, indicating a potentially higher understanding and reasoning capacity.

3.1.1 Selection Justification

Given the hardware restrictions and the project's benchmarking criteria, TinyLlama emerges as the more favourable choice over alternatives like RedPajama and Llama-2. Its updated training (trained with data till 29/01/2021) and lightweight nature, coupled with its suitability for a broad range of tasks including RAG applications and commonsense reasoning, make it the optimal choice for developing an advanced, optimised chatbot application within the specified constraints.

3.1.2 Conclusion

The selection of TinyLlama-1.1B-Chat-V1.0 is a strategic decision aimed at navigating the challenges of deploying advanced chatbot applications within environments constrained by hardware limitations. Its efficiency, coupled with significant optimization potential and broad

hardware compatibility, perfectly aligns with the project's goals, setting a new standard for the development of high-quality, resource-efficient chatbot applications.

3.2 Conversion to OpenVINO IR Format

3.2.1 Steps for Conversion

The process of converting the selected LLM model, ***Tiny-llama-1b-chat***, to OpenVINO's Intermediate Representation (IR) involves several steps, facilitated by the integration of Hugging Face Optimum Intel. This conversion is crucial for optimising the model's performance on Intel hardware. The following outlines the conversion process:

1. Environment Setup: Ensure that the Python environment is prepared with all necessary dependencies installed. This includes the installation of the Hugging Face transformers library, the optimum library with Intel extensions, the OpenVINO toolkit, and langchain toolkit.

First, let's create a virtual environment and install all dependencies.

```
virtualenv openvino
source openvino/bin/activate
pip install pip --upgrade
pip install --upgrade --upgrade-strategy eager optimum[openvino,nncf] evaluate
langchain pdfminer.six chromadb gradio spacy
spacy download en_core_web_sm
```

We must restart the runtime in order to use newly installed versions.

2. Model Loading: Load the ***Tiny-llama-1b-chat*** model using Hugging Face's transformers library. This involves retrieving the pre-trained model and its tokenizer, which are essential for processing input data and generating responses.

Next, moving to a Python environment, we import the appropriate modules and download the original model as well as its processor.

```
from config import SUPPORTED_LLM_MODELS
import ipywidgets as widgets
from optimum.intel.openvino import OVModelForCausalLM
from transformers import AutoTokenizer, pipeline
from transformers import AutoModelForCausalLM, AutoConfig
from optimum.intel import OVQuantizer
import openvino as ov
from pathlib import Path
import shutil
import torch
import logging
import nncf
```

```

import gc
import spacy
from transformers import (
    AutoTokenizer,
    StoppingCriteria,
    StoppingCriteriaList,
    TextIteratorStreamer,
)
from converter import converters, register_configs
register_configs()

model_non_optimized =
AutoModelForCausalLM.from_pretrained(model_configuration['model_id'
'])

from optimum.intel.openvino import OVModelForCausalLM
model =
OVModelForCausalLM.from_pretrained(model_configuration['model_id']
, export=True, use_cache=True)

```

The details of the *model_configuration* variable are defined in the notebook.

Selected model: *tiny-llama-1b-chat*

Selected model in Hub: *TinyLlama/TinyLlama-1.1B-Chat-v1.0*

3. Optimization with Optimum Intel: Utilise the Optimum Intel library to prepare the model for conversion. This library provides tools to optimise transformer models for Intel hardware, improving their execution efficiency.
4. Conversion to IR Format: Execute the conversion process using Optimum Intel's tools, which translate the model into OpenVINO's IR format. This step typically involves specifying the model's input and output parameters, along with any optimization flags that are relevant to the model's architecture.
5. Verification: After conversion, it's important to verify that the IR model maintains its integrity and performs as expected. This can involve running inference tests to compare the output of the IR model against the original Hugging Face model, ensuring that accuracy and response quality are preserved.

3.2.2 Compression Strategies

3.2.2.1 4-bit and 8-bit Compression

The conversion process also explores compression strategies, specifically 4-bit and 8-bit quantization, to further enhance the model's efficiency. These strategies reduce the precision of the model's weights, which can significantly decrease the model size and speed up inference times, with varying impacts on accuracy.

8-bit Quantization: This strategy compresses the model's weights to 8 bits, striking a balance between reducing model size and maintaining high accuracy. 8-bit quantization is generally well-supported across hardware, making it a versatile option for deployment. The

trade-off involves a slight reduction in model precision, which might affect performance in complex conversational scenarios but typically preserves the overall quality of interactions.

4-bit Quantization: More aggressive than 8-bit, 4-bit quantization further reduces the model size and can lead to even faster inference times. However, this comes at the cost of a more significant reduction in accuracy and model fidelity. The suitability of 4-bit quantization depends on the specific requirements of the chatbot application; it may be appropriate for scenarios where speed and model size are critical, and some loss in conversational quality is acceptable.

3.2.2.2 Trade-offs Discussion

Model Size vs. Inference Speed: Both 4-bit and 8-bit quantization effectively reduce the model size, with 4-bit achieving the most significant reduction. However, the choice between them often hinges on the target deployment environment's computational constraints and the need for rapid response times.

1. **Accuracy Considerations:** While 8-bit quantization maintains a closer approximation to the original model's accuracy, 4-bit quantization might result in noticeable drops in performance, particularly in nuanced conversational contexts. The decision between these strategies should consider the acceptable balance between efficiency and conversational quality.
2. **Hardware Compatibility:** The effectiveness and efficiency gains from quantization heavily depend on the target hardware's support for these compressed formats. Intel hardware, especially when optimised with OpenVINO, tends to support both 8-bit and 4-bit quantizations well, but the optimal choice may vary based on specific hardware capabilities and the application's performance requirements.

In summary, the conversion of the **Tiny-llama-1b-chat** model to OpenVINO's IR format, coupled with strategic model compression, is aimed at enhancing the chatbot's operational efficiency. The choice of compression strategy—whether 8-bit for balanced performance or 4-bit for maximum efficiency—should align with the chatbot's intended use case, ensuring that the application delivers responsive, high-quality interactions.

3.2.3 Optimization Process

The optimization process using OpenVINO, as outlined in the provided code block, is a comprehensive approach aimed at enhancing the performance and efficiency of deploying Large Language Models (LLMs) across various hardware platforms. This process involves several key steps, each targeting a different aspect of model optimization and precision reduction, from FP16 (16-bit floating-point) to INT8 and INT4 (8-bit and 4-bit integer) quantizations.

```
if prepare_fp16_model.value:
    convert_to_fp16()
if prepare_int8_model.value:
    convert_to_int8()
if prepare_int4_model.value:
    convert_to_int4()
```


3.2.3.1 Overview

FP16 Conversion: The initial step involves converting the model to FP16 format, which reduces the model's memory footprint and speeds up computation on compatible hardware without significantly impacting the model's accuracy. This process checks for an existing FP16 model and, if absent, performs the conversion using either a direct method for non-remote models or applying specific transformations for remote models. This step is crucial for enabling subsequent quantization processes.

INT8 Quantization: Following FP16 conversion, the next phase involves quantizing the model to INT8 precision. This step further reduces the model size and accelerates inference times, making it particularly beneficial for deployment on edge devices. The process includes conditionally converting to FP16 first, then applying INT8 quantization techniques. For non-remote models, quantization is applied directly, whereas, for remote models, additional steps to ensure compatibility and configuration alignment are taken before quantization.

INT4 Quantization: The most aggressive optimization comes with INT4 quantization, significantly compressing the model while still striving to maintain acceptable levels of accuracy. This step is tailored based on the specific model being optimised, with different configurations for compression mode, group size, and compression ratio. This stage necessitates careful handling of model configurations to ensure that the aggressive quantization does not overly compromise the model's performance.

3.2.3.2 Impact on Utility and User Experience

1. **Performance Enhancement:** Each step of the optimization process is designed to enhance the model's performance, particularly in terms of inference speed and resource efficiency. This enables more responsive and scalable applications, especially in resource-constrained environments.
2. **Deployment Flexibility:** By reducing the model's precision and size, the optimization process allows for greater flexibility in deployment options, including the ability to deploy sophisticated LLMs on edge devices and other hardware with limited computational capabilities.
3. **Balancing Accuracy and Efficiency:** The described optimization process carefully balances the trade-offs between model accuracy and computational efficiency. Through strategic quantization steps, it ensures that models remain effective while benefiting from significant performance improvements.

4. Functional Enhancements

In the landscape of conversational AI, the quest for more sophisticated, engaging, and useful chatbots is relentless. To this end, the integration of advanced Natural Language Processing (NLP) features such as Contextual Memory and Named Entity Recognition (NER) presents a transformative approach to improving chatbot functionality. This section explores how these features serve as pivotal enhancements, significantly elevating both the utility of chatbots and the user experience they offer.

4.1 Contextual Memory

4.1.1 Functional Enhancement

Contextual Memory endows chatbots with the ability to remember and utilise past interactions within a session. This capability ensures a level of conversational coherence previously unattainable in standard models, allowing chatbots to maintain context over a series of exchanges.

4.1.2 Impact on Utility and User Experience

Coherent Dialogue Maintenance: By referencing previous dialogue, chatbots can produce responses that are relevant and consistent with the ongoing conversation, greatly enhancing the natural flow of interaction.

Personalised User Engagement: The contextual awareness facilitated by this feature allows for more personalised responses, tailoring interactions to the user's specific needs and preferences based on the conversational history.

4.2 Named Entity Recognition (NER)

Functional Enhancement: NER allows chatbots to identify and classify key information (e.g., names, locations, organisations) within the text. This semantic understanding enriches the chatbot's responses, making them more informative and relevant. By highlighting entities, chatbots can offer users quick access to additional information or actions related to these entities, fostering a more interactive and enriched conversational experience.

4.3 Synergizing Contextual Memory and NER

The combination of Contextual Memory and NER amplifies the chatbot's capabilities, creating a synergistic effect that enhances both the depth and breadth of conversational interactions. This integration not only makes chatbots more responsive and context-aware but also transforms them into powerful tools for information retrieval and user engagement.

OpenVINO is an open-source software library for deep learning inference on CPUs, GPUs, and FPGAs. It provides a set of tools and libraries for developing and deploying machine learning models using **Intel® Neural Compute Stick (NCS)** and other devices. The library includes support for various neural network architectures, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and deep learning frameworks such as **TensorFlow** and **PyTorch**. OpenVINO also offers pre-trained models for popular tasks like object detection, image classification, and speech recognition. By leveraging OpenVINO, developers can accelerate their machine learning projects by up to 10x compared to traditional approaches.

Above is a sample response from LLM with NER and Contextual functionality enabled.

```
def user(message, history):  
    """  
        callback function for updating user messages in interface on submit  
        button click  
    Params:  
        message: current message  
        history: conversation history  
    Returns:  
        None  
    """  
    # Append the user's message to the conversation history  
    return "", history + [[message, ""]]
```

This code block updates the user query with in-memory conversational history.

5. Benchmarks

In the development of our optimised chatbot application, I conducted comprehensive benchmarks to evaluate the performance and efficiency of the TinyLlama-1b-chat model, focusing on its suitability for environments with stringent hardware constraints. Here's a summary of the benchmark findings that can be included in the whitepaper:

Quantization	Initialization Throughput	Inference Throughput	Size
Original*	~5 tokens/sec	~6 tokens/sec	4.2 GB
FP16	~2.5 tokens/sec	~6 tokens/sec	2 GB
INT8	~4 tokens/sec	~21 tokens/sec	1.1 GB
INT4	~9 tokens/sec	~26 tokens/sec	700 MB

** The original model is loaded using the transformer `AutoModelForCausalLM` API whereas the other models are loaded using `OVMModelForCausalLM` API from Optimum library.*

The original implementation of the TinyLlama model, utilising the transformer's **AutoModelForCausalLM** API, demonstrated an initialization throughput of approximately 5 tokens per second and an inference throughput of around 6 tokens per second, with a size of 4.2 GB. This baseline serves as a comparison point for the optimised models.

Applying FP16 quantization, I observed a modest adjustment in initialization throughput to roughly 2.5 tokens per second, while inference throughput remained stable at approximately 6 tokens per second. The model size was effectively halved to 2 GB, marking a substantial reduction with minimal impact on throughput.

More dramatic results were achieved with INT8 quantization, where initialization throughput increased to about 4 tokens per second, and inference throughput saw a significant jump to around 21 tokens per second. The model size was further reduced to 1.1 GB, demonstrating the effectiveness of INT8 quantization in balancing efficiency and performance.

The most remarkable enhancement came with INT4 quantization. The model's initialization throughput surged to approximately 9 tokens per second, and inference throughput increased to about 26 tokens per second. Moreover, the model size was reduced to a mere 700 MB. These improvements highlight the potential of advanced quantization techniques in optimising LLM-powered applications.

It's important to note that except for the original model loaded using the transformer's **AutoModelForCausalLM** API, all other models were optimised and loaded using the **OVMModelForCausalLM** API from the Optimum library. This distinction underscores the impact of specialised optimization APIs in achieving **superior model performance and efficiency**.

This benchmark analysis confirms that through strategic model optimization, including quantization, it's possible to significantly enhance the TinyLlama model's throughput and reduce its size, making it more suitable for a wide range of applications, particularly those with stringent hardware constraints.

6. Conclusion

6.1 Summary of Findings

The comprehensive efforts to optimise and enhance chatbot models, incorporating techniques such as model quantization (FP16, INT8, and INT4) via OpenVINO and the integration of advanced NLP features like Contextual Memory and Named Entity Recognition (NER), have yielded significant advancements in both model performance and chatbot functionality.

Model Optimization: The optimization process facilitated by OpenVINO, transitioning models through various levels of quantization, has demonstrated substantial improvements in inference speed and computational efficiency. Benchmarking results highlighted the success of these optimizations, showing that reduced precision models can achieve near-original accuracy with significantly less computational overhead, making them well-suited for deployment on edge devices and platforms with limited resources.

Functional Enhancements: The introduction of Contextual Memory and NER into chatbot models has enriched the user experience by providing more coherent, context-aware, and informative interactions. These enhancements have not only improved the chatbot's ability to maintain engaging and relevant conversations but also its capacity to offer personalised responses based on user history and to identify and highlight key information within the chat.

6.2 Implications and Future Work

The findings from this project carry substantial implications for future chatbot development and conversational AI research:

Scalability and Accessibility: The optimization techniques applied here underscore the potential for deploying advanced AI models in a broader range of environments, including those with stringent computational limitations. This opens new avenues for scalable and accessible AI applications, extending the reach of chatbots to more users and use cases.

Enhanced User Engagement: The functional enhancements introduced signify a move towards more intelligent and user-centric chatbots. Future projects can build on these improvements to create chatbots that are not only more responsive and helpful but also capable of driving higher user engagement through personalised and contextually rich interactions.

Cross-Platform Deployment: The project's outcomes highlight the importance of model optimization for cross-platform deployment. Future initiatives might focus on creating universal optimization pipelines that accommodate a wider array of hardware specifications, ensuring that chatbots can deliver consistent performance across different platforms.

In conclusion, the project's achievements in model optimization and functional enhancement represent a significant step forward in the development of advanced, efficient, and

user-friendly chatbot applications. By continuing to explore new optimization techniques and NLP features, the field can anticipate the emergence of chatbots that are not only technically proficient but also deeply attuned to the needs and preferences of their users, paving the way for a new era of conversational AI.

7. Code Structure and Model

Model used: *TinyLlama/TinyLlama-1.1B-Chat-v1.0*

Code files:

1. **Benchmark.ipynb**: A Jupyter notebook dedicated to benchmarking the performance of various models.
2. **LLM_chatbot.ipynb**: Gradio-based chat application leveraging quantized Large Language Models (LLMs)
3. **Create_quantized_models.ipynb**: A Jupyter Notebook focused on the process of quantization neural network models.
4. **Config.py**: A Python script that contains configuration settings for the project.
5. **Converter.py**: This Python script is responsible for converting models between different formats or preparing them for optimization.
6. **Ov_llm_model.py**: A Python script that defines the integration of Large Language Models (LLMs) with the OpenVINO toolkit.

8. References

1. [Openvino notebooks](#)
2. https://docs.openvino.ai/2023.3/openvino_docs_install_guides_installing_openvino_pip.html
3. https://docs.openvino.ai/2023.3/openvino_sample_benchmark_tool.html
4. https://docs.openvino.ai/2023.3/openvino_sample_bert_benchmark.html