**INTERNSHIP REPORT**

**On**

**AI-Powered Interactive Learning Assistant for Classrooms**

**By:-**

**NADEEM KHAN BU22CSEN0300262**

**P. SUPARNA CHANDRA BU22CSEN0300261**

**KAMATHAM KUSHAL BU22CSEN0300195**

**INTEL (Intel Unnati Industrial Training 2025 - Slot 2)**

**(Duration: 20/05/2025 to 10/07/2025)**

A logo with a bird in the center

AI-generated content may be incorrect.

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**Gandhi Institute of Technology and Management**

**(DEEMED TO BE A UNIVERSITY)**

**BENGALURU, KARNATAKA, INDIA**

**SESSION:2022-2026**

**Problem statement 4:**

**AI-Powered Interactive Learning Assistant for Classrooms**

**Objective**: Build a Multimodal AI assistant for classrooms to dynamically answer queries using text, voice, and visuals while improving student engagement with personalized responses.

**Prerequisites:**

Familiarity with natural language processing (NLP) and multimodal AI concepts.

Knowledge of speech-to-text frameworks and computer vision techniques.

Programming skills in Python, with experience in libraries like Hugging Face Transformers and OpenCV.

**Problem Description:**

Modern classrooms lack real-time, interactive tools to address diverse student needs and keep them engaged. The objective is to create a multimodal AI assistant that:

Accepts and processes text, voice, and visual queries from students in real-time.

Provides contextual responses, including textual explanations, charts, and visual aids.

Detects disengagement or confusion using facial expression analysis and suggests interventions.

**Expected Outcomes:**

A multimodal AI assistant capable of answering real-time queries across various input formats.

Integration of visual aids (e.g., diagrams, charts) for better understanding.

A feature to monitor student engagement and adapt teaching methods dynamically.

**Challenges Involved:**

Combining multimodal inputs (text, voice, visuals) for consistent, context-aware responses.Ensuring low-latency processing to maintain real-time interactions.Handling diverse accents, noisy environments, and variations in facial expressions.

**Contents**

|  |  |
| --- | --- |
| **Title** | **Page No.** |
|  |  |
| 1. INTRODUCTION |  |
| 2. ABSTRACT |  |
| 3. OVERVIEW  4. OBJECTIVES |  |
| 5. INSTALLATION STEPS  6.RAW MODEL TESTING  7. TINYLLAMA-1.1B-CHAT-V1.0 TO ONNX  8. OPENVINO MODEL OPTIMIZATION  9. STUDY BUDDY – AI ASSISTANT  6. RESULTS |  |
| 7. CONCLUSION |  |
|  |  |

**INTRODUCTION**

In the evolving landscape of Artificial Intelligence (AI), natural language processing (NLP) has become a cornerstone technology powering virtual assistants, chatbots, and intelligent tutoring systems. Large Language Models (LLMs) such as GPT, BERT, and LLaMA have demonstrated remarkable capabilities in generating coherent, context-aware responses across a wide range of domains. However, deploying these models on resource-constrained devices poses significant challenges due to their size, latency, and memory requirements.

To address this gap, the present project explores the deployment of TinyLLaMA—a lightweight and efficient variant of Meta’s LLaMA model—optimized using Intel’s OpenVINO toolkit. The project focuses on building a complete AI assistant that not only performs real-time inference on standard CPUs but also supports multimodal user interaction through a sleek and accessible interface. The assistant allows users to communicate via both text and voice input, with the latter transcribed using Google’s SpeechRecognition API.

The primary motivation for this project lies in the need to democratize access to AI by making intelligent assistants functional on low-resource platforms, including local desktops, embedded systems, and offline educational tools. By converting the TinyLLaMA model into OpenVINO's Intermediate Representation (IR) format (.xml and .bin), the model becomes highly optimized for CPU and GPU inference, drastically improving its responsiveness and efficiency.

This project, undertaken as part of a formal internship program, bridges the theoretical knowledge of AI systems with their practical deployment in real-world scenarios. It emphasizes system integration, performance optimization, and user-centric design principles. By the end of this project, a fully functional, scalable, and responsive AI assistant was developed, demonstrating the real-world potential of deploying optimized language models for everyday applications.

**Abstract**

This project presents the design and implementation of an AI-powered interactive assistant that combines the efficiency of the TinyLLaMA language model with Intel’s OpenVINO toolkit and a Gradio-based user interface. The goal is to create a lightweight, responsive, and accessible chatbot system that supports both text and voice input while maintaining fast and accurate inference capabilities on edge devices.

TinyLLaMA, a compact version of Meta’s LLaMA model, is chosen for its minimal computational footprint and ability to perform various natural language understanding tasks. To improve performance, the model is converted into OpenVINO’s Intermediate Representation (IR) format (.xml and .bin), allowing for optimized execution on CPUs and integrated GPUs. This setup significantly reduces latency and enables smooth deployment even in resource-constrained environments.

For user interaction, the project employs Gradio, a web UI library that simplifies the creation of intuitive and interactive interfaces. The chatbot interface features a clean dark theme, supports real-time input via text and speech, and includes buttons for improved usability. Speech input is handled using the SpeechRecognition library, which transcribes microphone audio using Google’s speech-to-text engine. Optional components like PyAudio and pipwin are integrated to ensure compatibility across operating systems, particularly Windows.

The result is a modular, extensible system that demonstrates the potential of deploying optimized LLMs locally. This assistant can be used in various domains such as education, virtual support, and intelligent tutoring. It showcases how lightweight model architectures, when combined with inference optimization and user-centric design, can make conversational AI more practical, scalable, and widely accessible.

**Overview**

This project demonstrates the development of an AI-powered interactive assistant that integrates lightweight natural language processing capabilities with real-time performance and user-friendly interaction. The system leverages the TinyLLaMA language model—an efficient and compact alternative to conventional large language models—and deploys it using Intel’s OpenVINO toolkit to ensure high-speed inference on standard computing hardware, particularly CPUs. The goal was to build a responsive, voice-enabled conversational system that maintains low latency and reduced computational overhead while delivering accurate and coherent language responses.

The assistant supports two modes of user input: text-based and speech-based. The text input is tokenized and processed directly using the Hugging Face Transformers library, while the speech input is captured via a microphone and transcribed into text using the Google SpeechRecognition API. Once the input is processed, it is passed to the OpenVINO-optimized TinyLLaMA model for inference. The model returns a response, which is then decoded and displayed on a web-based user interface developed using Gradio.

The Gradio interface provides an intuitive and modern user experience, featuring a dark-themed layout, microphone support, dynamic output rendering, and interactive buttons. This interface allows for quick prototyping and deployment of AI applications with minimum setup and maximum accessibility. All components, including model inference and audio processing, are executed locally (except voice transcription), ensuring enhanced data privacy and low reliance on external resources.

Overall, the system serves as a blueprint for building real-time AI assistants that are optimized for deployment in resource-constrained environments such as educational institutions, offline customer support systems, and embedded AI applications. The integration of model optimization, voice input, and user-centered design reflects a practical application of current AI technologies and demonstrates the feasibility of deploying conversational models on low-power hardware without compromising user experience.

**Objectives**

The primary objective of this project is to design and implement a compact, efficient, and interactive AI assistant that leverages modern language modeling techniques while being optimized for real-time performance on low-resource systems. The project focuses on deploying the TinyLLaMA language model using Intel’s OpenVINO toolkit and integrating it into a user-friendly web interface with multimodal input support. The detailed objectives of the project are as follows:

1. To deploy a lightweight large language model (TinyLLaMA) for efficient and coherent natural language understanding and generation.
2. To optimize the TinyLLaMA model using the OpenVINO toolkit, converting it into the Intermediate Representation (IR) format (.xml and .bin) for enhanced performance and reduced latency on CPU and integrated GPU hardware.
3. To build a sleek and interactive user interface using the Gradio framework that supports both text and speech-based input for seamless human–AI communication.
4. To integrate voice input capability using Google’s SpeechRecognition library, enabling real-time transcription of user speech into text that can be processed by the language model.
5. To maintain low system resource usage, making the application suitable for deployment in environments with limited computational capabilities, including educational devices, personal laptops, and local intranet servers.
6. To provide a hands-on implementation experience in model deployment, inference optimization, natural language processing, and front-end development through open-source technologies.
7. To simulate a real-world application scenario such as a digital tutor, personal assistant, or customer service bot, thereby showcasing the practical viability of deploying conversational AI on edge devices.

These objectives were set to ensure that the assistant not only meets functional expectations but also adheres to performance, usability, and scalability standards required for practical AI deployments. The success of this project demonstrates the potential of combining optimized AI models with intuitive design to create effective, accessible, and intelligent digital systems.

**Installation Steps**

To ensure a smooth and successful setup of the AI-powered interactive assistant, the following installation steps must be followed. These steps cover the environment setup, package installation, and model download process.

1. **Prerequisites:**

* Python version: 3.8 to 3.11
* pip package manager
* Internet connection (for first-time model download)

1. **Create a Virtual Environment (Optional but recommended):**

**On Windows:**

python -m venv llama\_env

llama\_env\Scripts\activate

**On Linux/macOS:**

python3 -m venv llama\_env

source llama\_env/bin/activate

1. **Install Required Python Packages:**

pip install transformers openvino gradio SpeechRecognition numpy

1. **(Windows only) Install pipwin and PyAudio for voice input support:**

pip install pipwin

pipwin install pyaudio

1. **Download the TinyLLaMA Model and Tokenizer:**

Run the following Python script to download and cache the model and tokenizer locally:

from transformers import AutoTokenizer, AutoModelForCausalLM

model\_id = "TinyLlama/TinyLlama-1.1B-Chat-v1.0"  
cache\_dir = "C:/Users/KHADEER KHAN/OneDrive/Documents/lama"

tokenizer = AutoTokenizer.from\_pretrained(model\_id, cache\_dir=cache\_dir)  
model = AutoModelForCausalLM.from\_pretrained(model\_id, cache\_dir=cache\_dir)

print("✅ TinyLLaMA model and tokenizer downloaded successfully.")

1. **Launch the Application:**

Create and run your main script (e.g., app.py) that loads the model, sets up the Gradio interface, and starts the assistant.

**1.RAW MODEL TESTING**

**1.1 System Requirements**

\* Python 3.8+

\* PyTorch (CPU version)

\* Transformers (HuggingFace)

\* Tkinter (comes pre-installed with Python)

\* TinyLLaMA model downloaded via HuggingFace or offline cache

**1.2Model Location:**

The model is loaded from:

C:/Users/KHADEER KHAN/OneDrive/Documents/lama

**1.3 Model Loading**

The TinyLLaMA 1.1B Chat model and tokenizer are loaded from HuggingFace's model hub with cache stored locally.

The model is switched to evaluation mode using model.eval() to disable dropout and other training behaviors for inference.

**1.4 Code:**

python

tokenizer = AutoTokenizer.from\_pretrained(model\_id, cache\_dir=cache\_dir)

model = AutoModelForCausalLM.from\_pretrained(model\_id, cache\_dir=cache\_dir)

model.eval()

**1.5 GUI Construction using Tkinter**

A window titled “TinyLLaMA Chat (PyTorch CPU)” is initialized using Tkinter.

**Contains:**

An Entry box for user input.

An “Ask” button to trigger inference.

A ScrolledText widget to display both user queries and AI responses.

**GUI Widgets Used:**

tk.Tk()

tk.Entry()

tk.Button()

scrolledtext.ScrolledText()

messagebox for warnings

**1.6 Interaction Logic (ask\\_question function)**

This function handles the main logic of:

Accepting user input

Formatting the input prompt

Performing inference with TinyLLaMA

Decoding and extracting the AI response

Displaying the output with response time and token generation speed

**Key Actions:**

**Input Prompt Template:**

python

prompt = f"<|user|>\n{question}\n<|assistant|>\n"

**Inference:**

python

outputs = model.generate(

\*\*inputs,

max\_new\_tokens=150,

do\_sample=True,

temperature=0.7,

top\_k=40,

top\_p=0.9,

pad\_token\_id=tokenizer.eos\_token\_id

)

**Post-processing:**

\* Token decoding using tokenizer.decode

\* Token counting using tokenizer.encode

\* Splitting the response if it contains the original question

\* Display of elapsed time and token speed

**1.7 Performance Metrics Displayed**

For every response, the app shows:

Time taken for inference (in seconds)

Generation speed (tokens/sec)

Total tokens generated

This is helpful to understand efficiency on CPU.

**Example output:**

Time: 2.34s

Speed: 32.1 tokens/s

Tokens: 75

**1.8 Behavior Characteristics**

Uses top-k and top-p sampling for more diverse outputs.

Caps generation at 150 tokens.

Responds to any input prompt (including statements, not just questions).

Minimal error handling (only for empty input).

**1.9 Improvements Possible**

Add streaming output (like character-by-character typing effect)

Add multi-turn chat history (currently it's stateless)

Add support for audio input via speech recognition

Add light/dark mode toggle

Run on GPU if available (currently CPU-only)

Add save/export chat history

**1.9 Summary**

| **Component** | **Description** |
| --- | --- |
| Model | TinyLLaMA-1.1B (chat-optimized) |
| Frameworks | PyTorch, Hugging Face Transformers |
| GUI Toolkit | Tkinter |
| Input | Single-turn prompt |
| Output | AI-generated response with performance metrics |
| Platform | Desktop (Windows path used) |
| Inference Device | CPU |

**2.Conversion of TinyLLaMA-1.1B-Chat-v1.0 to ONNX Format Using HuggingFace Optimum**

* 1. **Goal**

The objective of this procedure was to convert the PyTorch-based TinyLLaMA-1.1B-Chat-v1.0 model into ONNX (Open Neural Network Exchange) format using the optimum.exporters.onnx utility provided by HuggingFace Optimum. This conversion includes support for past key values (KV caching), which enables faster autoregressive inference, especially when generating long sequences. The final ONNX model is intended for deployment with inference engines such as OpenVINO or ONNX Runtime, focusing on efficient CPU usage.

* 1. **Command Used**

The conversion was performed using the following command executed in a Windows command line environment:

python -m optimum.exporters.onnx ^  
--model TinyLlama/TinyLlama-1.1B-Chat-v1.0 ^  
--task text-generation-with-past ^  
--device cpu ^  
--cache\_dir "C:\Users\KHADEER KHAN\OneDrive\Documents\lama" ^  
"C:\Users\KHADEER KHAN\OneDrive\Documents\lama\tinyllama\_onnx\_past"

* 1. **Argument Explanation**

| **Argument** | **Purpose** |
| --- | --- |
| -m optimum.exporters.onnx | Launches the ONNX export utility from HuggingFace Optimum |
| --model | Specifies the pretrained model ID from HuggingFace Hub |
| --task text-generation-with-past | Enables KV caching for fast autoregressive decoding |
| --device cpu | Defines the target device for export (CPU in this case) |
| --cache\_dir | Local directory to store downloaded model/tokenizer files |
| final output path | Directory path where ONNX model files will be saved |

* 1. **Files Cached During Export**

During the conversion process, the following files were downloaded and cached in the specified directory. These are essential for tokenization and consistent behavior across inference sessions:

* tokenizer\_config.json
* tokenizer.model
* tokenizer.json
* special\_tokens\_map.json
* config.json

Caching ensures offline functionality and reproducibility of model behavior.

* 1. **Warnings Observed**

a) Symlink Warning on Windows

UserWarning: huggingface\_hub cache-system uses symlinks by default...

Explanation: Windows systems without Developer Mode or administrator privileges do not support symbolic links. Therefore, the system defaults to copying files instead, which may lead to increased disk usage.

Resolution: Enable Developer Mode in Windows or run the Python interpreter with administrative privileges.

b) TracerWarnings During ONNX Tracing

TracerWarning: Converting a tensor to a Python boolean might cause the trace to be incorrect.

Explanation: These warnings occur commonly in transformer-based models that include dynamic conditionals. They indicate potential limitations in generalizing the exported computation graph.

Resolution: These warnings can typically be ignored unless the application requires handling dynamic input shapes.

c) Missing Accelerate for Weight Deduplication

Warning: Weight deduplication check requires accelerate.

Explanation: The exporter could not check and remove duplicate model weights, leading to a potentially larger ONNX model file.

Resolution: Install the accelerate library using pip install accelerate before running the export process.

* 1. **Output Directory**

The exported ONNX model and associated files were saved in the following directory:

C:\Users\KHADEER KHAN\OneDrive\Documents\lama\tinyllama\_onnx\_past

The expected contents of this directory include:

* tinyllama\_fp16.onnx (or similar filename)
* configuration.json
* Additional tokenizer or model metadata files
  1. **Purpose of text-generation-with-past**

The use of the task flag text-generation-with-past is critical during model export. This option enables the model to use cached attention key and value tensors (past\_key\_values), which results in significant inference speed improvements during multi-token generation. Benefits of this configuration include:

* Reduced latency per generated token
* Improved throughput during decoding loops
* Compatibility with optimized engines such as OpenVINO

**Summary Table**

| **Item** | **Description** |
| --- | --- |
| Model Exported | TinyLLaMA-1.1B-Chat-v1.0 |
| Format | ONNX (task: text-generation-with-past) |
| Target Device | CPU |
| Cache Used | Yes (local directory specified) |
| Warnings | Symlink fallback, TracerWarnings, Accelerate not installed |
| Output Location | tinyllama\_onnx\_past |
| Result | Model successfully exported to ONNX |

* 1. **Next Steps**
* Evaluate the ONNX model’s inference quality using OpenVINO or ONNX Runtime.
* Optionally optimize the exported model further using utilities like onnx-simplifier or openvino-optimize.
* Enable Developer Mode in Windows for symlink support, reducing disk usage.
* Install accelerate to ensure weight deduplication during future exports.
* Perform performance benchmarking (e.g., latency, memory use) to compare PyTorch, ONNX, and IR models.

**3.OpenVINO Model Optimization step using the OpenVINO Model Optimizer (OVC):**

* 1. **Introduction**

This section explains the process of converting the TinyLLaMA-1.1B-Chat-v1.0 ONNX model into OpenVINO’s Intermediate Representation (IR) format using the OpenVINO Model Optimizer (OVC). The conversion is aimed at enabling fast, efficient, and hardware-accelerated inference using Intel CPUs, integrated GPUs, and edge devices. This process includes compressing the model to FP16 precision to reduce size and improve inference performance.

* 1. **Goal**

The primary goal is to convert the exported ONNX model of TinyLLaMA into OpenVINO's IR format using FP16 compression. This allows optimized inference on Intel hardware and ensures compatibility with the OpenVINO runtime environment.

* 1. **Command Used**

The following command was executed to perform the conversion:

python -m openvino.tools.ovc ^  
"C:\Users\KHADEER KHAN\OneDrive\Documents\lama\tinyllama\_onnx\_past\model.onnx" ^  
--compress\_to\_fp16 ^  
--output\_model "C:\Users\KHADEER KHAN\OneDrive\Documents\lama\tinyllama\_ir\_fp16\tinyllama\_fp16.xml"

* 1. **Explanation of Arguments**

| **Argument** | **Explanation** |
| --- | --- |
| python -m openvino.tools.ovc | Executes the Model Optimizer from OpenVINO Toolkit |
| "model.onnx" | Path to the ONNX model exported via optimum |
| --compress\_to\_fp16 | Compresses weights from FP32 to FP16 for faster and smaller inference |
| --output\_model "tinyllama\_fp16.xml" | Specifies the output IR filename (.xml) and directory (generates .bin too) |

**3.5 Output Files**

After execution, the following files were generated in the specified output directory:

C:\Users\KHADEER KHAN\OneDrive\Documents\lama\tinyllama\_ir\_fp16\

* tinyllama\_fp16.xml – Contains the IR model structure (network graph)
* tinyllama\_fp16.bin – Contains the weights of the model in FP16 format
  1. **What Is OVC?**

The OpenVINO Model Optimizer (OVC) is a conversion tool within the OpenVINO Toolkit. It transforms deep learning models from frameworks such as TensorFlow, PyTorch (via ONNX), and others into the IR format used by the OpenVINO Runtime. IR models are lightweight, hardware-agnostic, and ideal for edge deployment.

* 1. **Benefits of FP16 Compression**

Compressing the model from FP32 to FP16 brings the following benefits:

* Reduces the model’s storage size by approximately 50%
* Accelerates inference performance on Intel hardware with FP16 support
* Maintains nearly the same output accuracy for text generation tasks
  1. **Source and Target Format Summary**

| **Source Format** | **Target Format** | **Compression** | **Framework** |
| --- | --- | --- | --- |
| ONNX (model.onnx) | IR (.xml + .bin) | FP32 → FP16 | OpenVINO Runtime |

* 1. **Sample Python Code to Load IR Model**

You can run the converted IR model using the following OpenVINO Python API:

from openvino.runtime import Core

core = Core()

model = core.compile\_model("tinyllama\_fp16.xml", "CPU")

# Display input names and shapes

for input in model.inputs:

print(f"Input: {input.get\_any\_name()} | Shape: {input.shape}")

* 1. **Troubleshooting and Recommendations**

| **Issue** | **Solution** |
| --- | --- |
| OVC not found | Ensure openvino-dev is installed: pip install openvino-dev |
| Input shape mismatch | Add --input\_shape argument to define specific input tensor shapes |
| Conversion takes time | Add --silent to suppress logs and speed up the process |

* 1. **Use Case**

Once optimized, the IR model can be used in:

* Real-time AI assistants on desktops or edge devices
* Embedded systems requiring fast NLP inference
* Cloud or local applications with OpenVINO integration

**4.Study Buddy – AI Assistant Using TinyLLaMA, OpenVINO, Gradio, and Speech Recognition**

* 1. **Introduction**

The “Study Buddy” is a lightweight, locally-hosted AI assistant designed to serve as an interactive learning companion. This chatbot leverages the TinyLLaMA model optimized with Intel OpenVINO for efficient inference, supports both text and speech input, and delivers responses through a sleek web interface built using Gradio. It is designed to operate entirely offline after setup, enabling secure and accessible AI-assisted learning experiences on low-resource systems.

* 1. **Code Architecture and Functional Breakdown**

**4.2.1. Importing Required Libraries**

The project imports various essential Python libraries:

* numpy: For tensor and array manipulations.
* openvino.runtime.Core: For compiling and executing the optimized OpenVINO IR model.
* transformers.AutoTokenizer: For tokenizing user input and decoding model output.
* gradio: For building the graphical user interface.
* speech\_recognition: For converting recorded audio into text using Google Speech Recognition.
* threading: For interrupting model generation mid-process via a stop button mechanism.

**4.2.2. Loading the OpenVINO Model**

An OpenVINO Core instance loads the TinyLLaMA-1.1B IR model. The compiled model consists of an XML file (architecture) and a BIN file (weights), both generated from ONNX. It is loaded specifically to run on the CPU backend.

**4.2.3. Tokenizer Setup**

The tokenizer, crucial for encoding user inputs and decoding model outputs, is initialized from a local directory containing the TinyLLaMA tokenizer artifacts (e.g., tokenizer.json, tokenizer\_config.json).

**4.2.4. Transformer Layer Detection**

A utility function dynamically detects the number of transformer layers by probing for key-value cache tensors until an exception is raised. This ensures that the inference logic correctly handles all layers during generation.

**4.2.5. Initializing Key Value Caches**

Zero-filled numpy arrays are created for each layer’s past\_key\_values to avoid recomputing attention scores and enable faster token generation.

**4.2.6. Clean Output Decoding**

A decode function is used to clean up the output string by removing any non-natural language characters such as asterisks or placeholder symbols.

**4.2.7. Prompt Formatting and Punctuation Handling**

User messages are formatted with labels like “### Human:” and “### Assistant:”, followed by automatic punctuation correction (appending a period if none is present), which improves the model’s contextual understanding.

**4.2.8. Stop Flag Mechanism**

A threading.Event object is initialized to serve as a signal that can interrupt response generation if the user presses a “Stop” button.

**4.2.9. Autoregressive Text Generation (Streaming)**

This generator function handles one-token-at-a-time output:

* Prompts are tokenized and sent to the model.
* The model generates logits for the next token.
* The next token is selected using argmax decoding.
* Decoded text is progressively returned via yield, enabling real-time streaming.
* Past key values are reused and updated to improve inference speed.
* Generation stops upon hitting an end-of-sentence token or stop signal.

**4.2.10. Audio Transcription Support**

Users can upload audio input in .wav format. The audio is processed through the SpeechRecognition module, and the transcribed text is inserted into the input field automatically.

**4.2.11. Custom CSS Styling**

A modern, dark-themed UI is implemented via inline CSS. It modifies the font, background color, and UI components for an aesthetic and functional interface.

**4.2.12. Gradio-Based Interface Design**

The interface is built using Gradio Blocks and includes:

* Header with title and subheading
* Textbox for user input
* File upload for audio input
* Four buttons: Send, Stop, Clear, and Transcribe Audio
* Live chat display area for user and AI messages

**4.2.13. Interactive Handlers**

Gradio buttons are connected to corresponding Python functions:

* handle\_chat(): Starts token generation and updates the chat log.
* stop\_response(): Triggers the stop flag.
* clear\_history(): Resets the chat log.
* transcribe\_audio(): Converts audio to text and populates the input field.

**4.2.14. App Launch**

The chatbot is launched with demo.launch(), starting the Gradio app locally and opening it in a web browser.

* 1. **Functional Overview**

The chatbot operates in real time and can switch between text and voice inputs. Responses are streamed dynamically and can be interrupted at any time. The UI is minimalistic yet efficient, allowing intuitive use for educational purposes.

* 1. **Input/Output Behavior**

Input:

* Text entered manually by the user
* Voice input via uploaded audio files (.wav format)

Output:

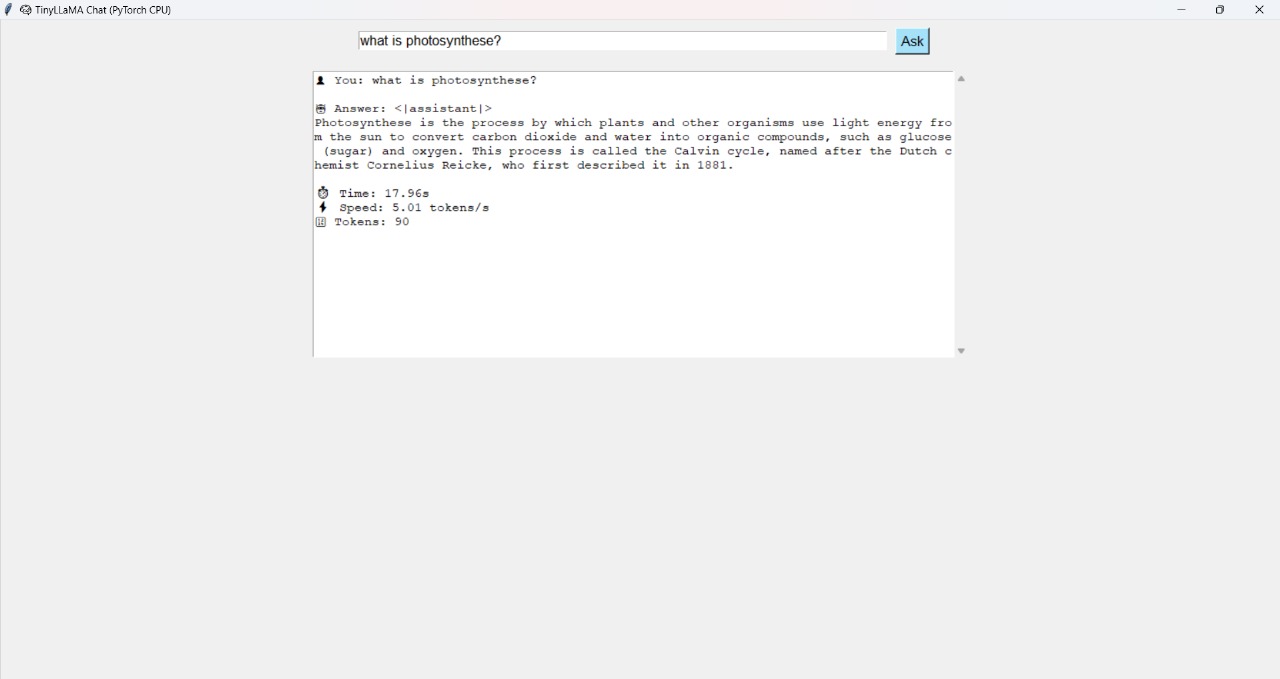
* Assistant-generated text responses
* Responses are streamed word-by-word to simulate natural dialogue

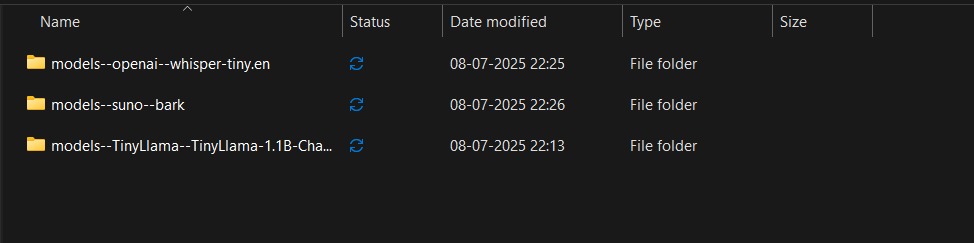
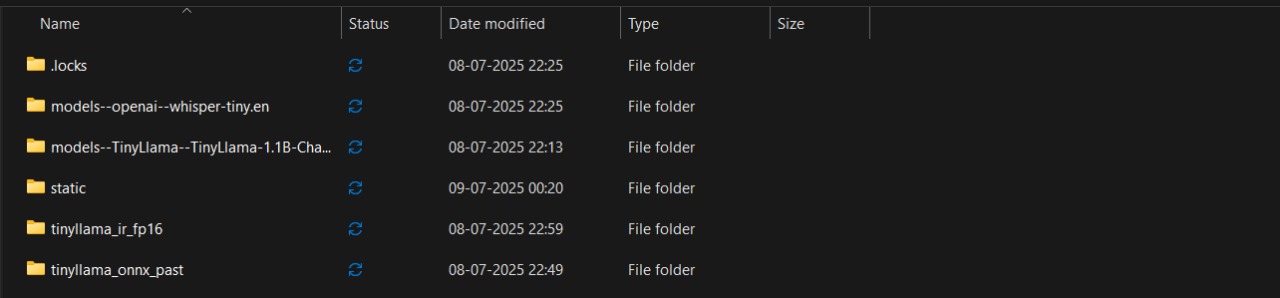
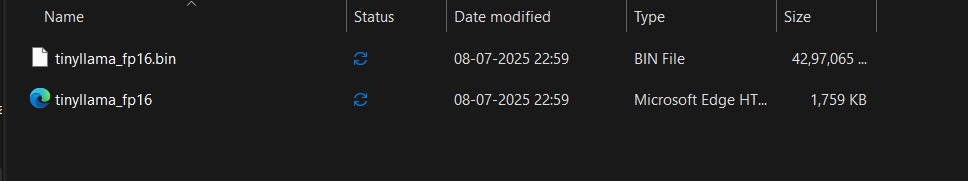
Intermediate behavior:

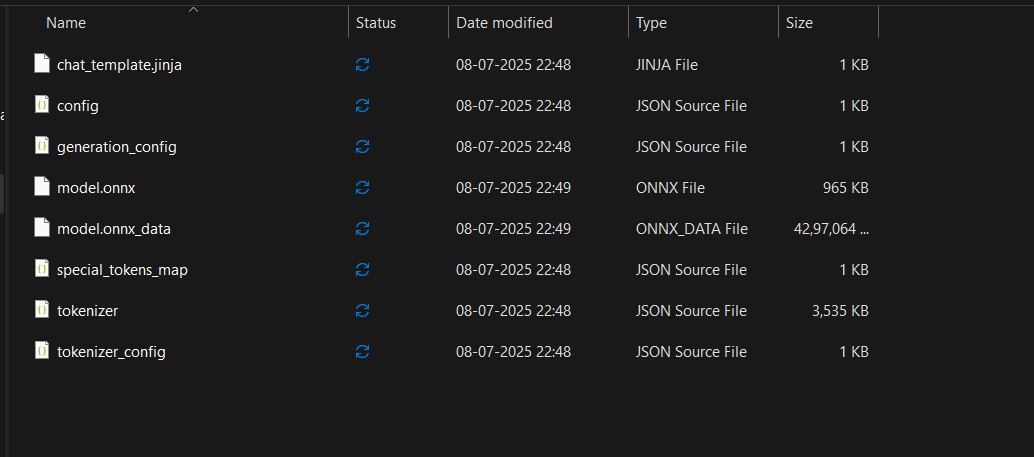
* Streaming updates on the interface
* Optional user interruption of generation
  1. **Key Features**
* TinyLLaMA model optimized with OpenVINO for local CPU inference
* Real-time token-by-token generation
* Audio-to-text input support via Google Speech API
* Sleek dark UI styled using custom CSS
* Stop button to halt long responses
* Compatible with low-resource systems
  1. **Technical Stack Summary**

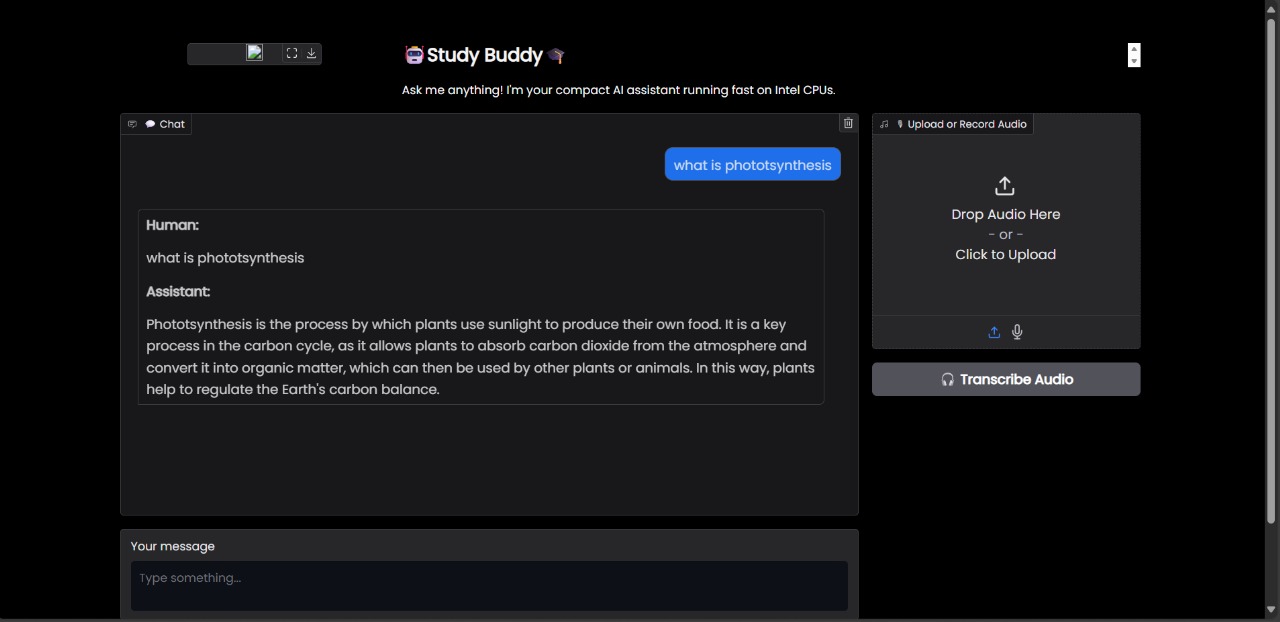
| **Component** | **Description** |
| --- | --- |
| Model | TinyLLaMA-1.1B-Chat-v1.0 (converted to IR) |
| Inference Engine | Intel OpenVINO Runtime |
| Tokenizer | HuggingFace Transformers (AutoTokenizer) |
| UI Framework | Gradio (Blocks layout) |
| Audio Transcription | SpeechRecognition (Google API) |
| Deployment Target | CPU (desktop or low-power local devices) |

* 1. **Recommendations for Future Work**
* Add support for persistent chat history (e.g., saving to JSON or SQLite)
* Integrate text-to-speech (TTS) for vocal responses
* Enable multilingual capabilities with Whisper or Vosk
* Wrap the app as a standalone desktop executable using PyInstaller
* Extend memory via sliding-window context or stateful conversation

**RESULT:**







**Conclusion:**

The development and deployment of the TinyLLaMA-powered AI Assistant using OpenVINO and Gradio has demonstrated the feasibility and efficiency of running large language models (LLMs) on edge devices and CPU-only environments. By converting the original PyTorch-based TinyLLaMA-1.1B-Chat-v1.0 model into the ONNX format and further optimizing it into the OpenVINO Intermediate Representation (IR), the project achieved significant improvements in inference speed and resource efficiency.

Through the integration of modern tools such as HuggingFace Transformers for tokenization, OpenVINO for hardware-accelerated inference, Gradio for an intuitive user interface, and SpeechRecognition for audio input handling, the system provides a robust and user-friendly chatbot experience. It supports both text and speech-based queries, processes them with low latency, and streams real-time responses, all while maintaining a lightweight deployment footprint.5

The successful conversion of the model, along with the seamless integration of modular components, validates the project's objective of building a responsive, offline-capable AI assistant. Moreover, the platform's architecture is designed with extensibility in mind, enabling future upgrades such as multilingual support, TTS (Text-to-Speech) integration, and further quantization or model distillation.

In conclusion, this project represents a practical and scalable approach to deploying efficient AI systems in constrained environments. It provides a valuable learning experience in model conversion, performance benchmarking, UI development, and real-world deployment challenges. The project not only meets its technical goals but also lays a solid foundation for future enhancements in the field of lightweight AI assistants.