Disha: ChatBot for IIIT Nagpur

Project report submitted to
Indian Institute of Information Technology, Nagpur,
In partial fulfillment of the requirements for the award of
the degree of

Bachelor of Technology
In
Computer Science and Engineering

by

Gyanbardhan (BT21CSE194)

Lakshit Upreti (BT21CSE161)

Abhishek Kumar (BT21CSE188)

Under the guidance of

Dr. Mangesh Kose

Indian Institute of Information Technology, Nagpur **441108**
(India)

2025

Disha: ChatBot for IIIT Nagpur

Project report submitted to
Indian Institute of Information Technology, Nagpur,
In partial fulfillment of the requirements for the award of
the degree of

Bachelor of Technology
In
Computer Science and Engineering

by

Gyanbardhan (BT21CSE194) **Lakshit Upreti (BT21CSE161)** **Abhishek Kumar (BT21CSE188)**

Under the guidance of

Dr. Mangesh Kose

Indian Institute of Information Technology, Nagpur **441108**
(India)

2025

Department of Computer Science and Engineering
Indian Institute of Information Technology, Nagpur

Declaration

We, Lakshit Upreti (BT21CSE161), Gyahbardhan (BT21CSE194), Abhishek Kumar (BT21CSE188) hereby declare that this project work titled "Disha: ChatBot for IIIT Nagpur" is carried out by me/us in the Department of Computer Science and Engineering of Indian Institute of Information Technology, Nagpur. The work is original and has not been submitted earlier whole or in part for the award of any degree/diploma at this or any other Institution /University.

Sr. No.	Enrollment No	Name	Signature
1	BT21CSE194	Gyanbardhan	
2	BT21CSE161	Lakshit Upreti	
3	BT21CSE188	Abhishek Kumar	

Date: 23/11/2024

Declaration

We, Lakshit Upreti (BT21CSE161), Gyanbardhan (BT21CSE194), Abhishek Kumar (BT21CSE188), understand that plagiarism is defined as any one or the combination of the following:

1. Uncredited verbatim copying of individual sentences, paragraphs or illustrations (such as graphs, diagrams, etc.) from any source, published or unpublished, including the internet.
2. Uncredited improper paraphrasing of pages or paragraphs (changing a few words or phrases, or rearranging the original sentence order).
3. Credited verbatim copying of a major portion of a paper (or thesis chapter) without clear delineation of who did or wrote what. (Source: IEEE, the institute, Dec.2004) I have made sure that all the ideas, expressions, graphs, diagrams, etc. that are not a result of my own work, are properly credited. Long phrases or sentences that had to be used verbatim from published literature have been clearly identified using quotation marks.

I affirm that no portion of my work can be considered as plagiarism and I take full responsibility if such a complaint occurs. I understand fully well that the guide of the thesis may not be in a position to check for the possibility of such incidences of plagiarism in this body of work.

Date: 23/11/2024

Sr. No.	Enrollment No	Name	Signature
1	BT21CSE194	Gyanbardhan	
2	BT21CSE161	Lakshit Upreti	
3	BT21CSE188	Abhishek Kumar	

Certificate

This is to certify that the project titled "Disha: ChatBot for IIIT Nagpur", submitted by Lakshit Upreti (BT21CSE161), Gyahbardhan (BT21CSE194), Abhishek Kumar (BT21CSE188) in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering**, IIIT Nagpur. The work is comprehensive, complete and fit for final evaluation.

Date: 23/11/2024

Dr. Mangesh Kose
Supervisor
Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Information Technology, Nagpur

Dr. Nishat A. Ansari
Head of the Department
Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Information Technology, Nagpur

Acknowledgment

We would like to express our sincere gratitude to our supervisor, **Dr. Mangesh Kose**, Assistant Professor, Department of Computer Science and Engineering, Indian Institute of Information Technology, Nagpur, for his invaluable guidance and encouragement throughout this project.

We are also deeply grateful to **Dr. Nishat Ansari**, Head of the Department, CSE for providing the necessary facilities and support and to **Dr. Tausif Diwan**, Associate Dean, for his insights and motivation.

Lastly, we express our profound gratitude to **Dr. Prem Lal Patel**, Director, Indian Institute of Information Technology Nagpur, for fostering an environment of academic excellence that greatly benefited our work.

We are very thankful to the project review committee for their everlasting support and guidance on the ground in which we have acquired a new field of knowledge. We also do not like to miss the opportunity to acknowledge the contribution of all faculty members of the department for their kind assistance and cooperation during the development of our project.

We also acknowledge with a deep sense of reverence, our gratitude towards our parents and members of our family, who have always supported us merely as well as economically. And last but not least, gratitude goes to all of our friends who directly and indirectly helped us to complete this project and this project report.

Project Students :

Lakshit Upreti (BT21CSE161)

Gyanbardhan (BT21CSE194)

Abhishek Kumar (BT21CSE188)

Abstract

This project presents ***Disha***, an intelligent chatbot designed to enhance user interaction and simplify navigation on the **IIIT Nagpur** website. Leveraging advanced NLP techniques, cutting-edge machine learning algorithms, and large language models (LLMs), ***Disha*** offers a seamless and voice-enabled interface tailored to meet diverse user needs. The system integrates **FineTtuned LLMs** and **Retrieval-Augmented Generation (RAG)** methodologies to provide precise, context-aware responses, ensuring a reliable and engaging conversational experience.

Disha supports the English language, catering to the global users while enabling intuitive interactions through voice commands. With specialised machine learning models, including **fine-tuned LLaMA-3.2-1b** and **Phi-5**, and enhanced by **PEFT** techniques like **LoRA** and **QLoRA**, ***Disha*** is optimised for IIIT Nagpur-specific queries. It utilises robust tools such as Pinecone and LangChain for data storage and pipeline creation, alongside **Google Gemini** for response generation. A Summarization **LLM** ensures cohesive and user-friendly outputs, blending information from **RAG** and fine-tuned models.

Preliminary evaluations demonstrate ***Disha's*** ability to deliver accurate, contextual, and verified responses, significantly improving user accessibility and satisfaction. Feedback has highlighted its usability and efficiency in navigating website information.

Future advancements aim to expand language support to additional regional and international languages, scale the system to accommodate larger datasets and complex queries, and refine its conversational intelligence. With continuous improvements in model training, real-time capabilities, and user-centric design, ***Disha*** aspires to set a benchmark in educational chatbot solutions, fostering an inclusive and efficient digital ecosystem for IIIT Nagpur.

Table Of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviations	Description
LLM	Comparison of different LLMs
RAG	Retrieval Augmented Generation
NLP	Natural Language Processing
PEFT	Parameter Efficient Fine Tuning
LORA	Low-Rank Adaptation
QLORA	Quantized Low-Rank Adaptation
GPT	Generative Pretrained Transformer
LLAMA	Large Language Model Meta AI
RL	Reinforcement Learning
BLEU	Bilingual Evaluation Understudy
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
SFT	Supervised Fine Tuning
ANN	Approximate Nearest Neighbour
BERT	Bidirectional Encoder Representations from Transformers
API	Application Programing Interface
JSON	JavaScript Object Notation
OCR	Optical Character Recognition
trl	Transformer Reinforcement Learning

Chapter 1
Introduction

1.1. Problem Statement

Information Overload! In today's digital age, finding the right information can be overwhelming. Our college website is no exception. We noticed that navigating the site can be particularly challenging, especially for newcomers like prospective students and their parents searching for admission criteria and other official instructions. Both students and visitors often struggle to locate the information they need, making the website feel like a puzzle with missing pieces.

1.2. Disha

We decided to tackle this problem head-on with ***Disha***, our innovative solution designed to simplify the user experience. Picture this: asking questions naturally, just like a conversation, and receiving instant, accurate answers! Powered by cutting-edge Machine Learning, Natural Language Processing, and Large Language Models, ***Disha*** is a conversational chatbot built to revolutionise how users navigate our college website.

From prospective students and parents exploring admission criteria to current students seeking specific information, ***Disha*** is here to provide quick, convenient, and reliable responses. Its intuitive interface and smart capabilities ensure a smooth and stress-free browsing experience.

Disha is more than just a chatbot—it’s a transformative tool. By saving time and eliminating frustration, it makes accessing information effortless and enjoyable. "With ***Disha***, finding what you need is a breeze!"

Chapter 2

Literature Review

In the development of Disha, the chatbot for IIIT Nagpur, we extensively reviewed relevant research papers to gather insights and ideas. These studies provided a strong foundation for understanding conversational AI, Fine Tuning and information retrieval techniques, which were instrumental in shaping the design and functionality of our chatbot.

Tan, Ting Fang et al. in 2024 [1] proposes a design and implementation of a chatbot for an accounting firm, leveraging fine-tuned LLMs to address complex fiscal queries. Two novel datasets were developed: the Dataset, from real-world fiscal Q&A on the Italian tax authority portal, and the personal Dataset, synthesized using the Generator for cost-effective domain-specific training. Using the QLoRA fine-tuning approach, the chatbot improved tax query handling by 40% over generic LLMs, achieving 92% accuracy in professional scenarios.

Sepulveda at el. in 2024 [2] addresses the accessibility challenges of Colombia's Aeronautical Regulations (RAC), a dense legal framework critical to aviation. It introduces the first RAC-specific dataset with 24,478 labeled Q&A pairs, created using semi-automated extraction for high-quality data. Leveraging the GEMMA 1.1 2b model, techniques like LoRA, flash attention, and PEFT improved memory efficiency and domain adaptability, while GPU-efficient methods like Unsloth reduced costs and environmental impact. The V8 model achieved a training loss of 0.194 and an expert evaluation score of 5.6/7, excelling in simplifying RAC sections.

Kulkarni at el. in 2024 [3] introduces a novel RL-based approach to optimize RAG pipelines for domain-specific chatbots, focusing on credit card FAQs. An in-house embedding model trained with infoNCE loss effectively handles English and Hinglish queries, achieving top-1 accuracy of 97% and 94%, respectively, outperforming public models. The RL framework employs policy-based decisions to fetch FAQ

contexts selectively, reducing LLM token usage by 31% and improving accuracy by minimizing unnecessary retrievals. This balance of cost reduction and performance demonstrates the potential of combining RL optimization with fine-tuning techniques.

The study [4] presents BARKPLUG V.2, a RAG-based chatbot designed to enhance access to university resources at Mississippi State University. Neupane at el. in 2024 incorporated data from 42 campus departments, combining efficient context retrieval via embedding models and vector databases with GPT-3.5-turbo for response generation. The chatbot achieved a score of 0.96, reflecting high accuracy and contextual relevance, while a System Usability Scale (SUS) score of 67.75 indicated satisfactory user experience with room for improvement. Highlighting RAG's potential in academic contexts, integrating this approach with fine-tuned LLMs in our project aims to further improve accuracy and efficiency in personalized responses.

In 2024, Zhang at el. [13] examined the effects of scaling factors, including LLM size, pre-training data volume, and fine-tuning methods (Full-Model Tuning, Prompt Tuning, and LoRA), on downstream performance. Experiments with bilingual LLMs (1B–16B parameters) on translation and summarization tasks reveal a power-law scaling relationship, with model size having a greater impact than pre-training data, while scaling fine-tuning parameters offers limited gains. The findings underscore the task- and data-dependent nature of fine-tuning, highlighting challenges in selecting optimal methods based on scaling behaviors.

The overview study [14] highlights comparison of traditional course material search functionality with a RAG-based AI chatbot for information-seeking in a web development course. Both tools were useful, but the chatbot scored slightly higher (4.57/5 vs. 4.36/5). The chatbot excelled at summarizing large content, while the search was faster for pinpointing specifics. However, the chatbot risked hallucinated responses, requiring user validation. Participants preferred the tool they used second, reflecting task- and context-dependency. Combining both systems could balance speed, accuracy, and strengths, offering superior outcomes for diverse tasks.

In 2024, H. K. Chaubey at el. [15] introduces an efficient healthcare chatbot leveraging LangChain, RAG, and performance-optimized LLMs fine-tuned with

PEFT techniques like LoRA and QLoRA. By integrating domain knowledge with web-scraped data, the chatbot delivers accurate and contextually relevant responses. Evaluation metrics showed significant performance gains, with RAG achieving the highest accuracy (BLEU SCORE : 0.83, ROUGE SCORE : 0.91). Future work focuses on advanced RAG techniques, improved ambiguity handling, and enhanced user interfaces to revolutionize healthcare information access and patient education.

Inspired by studies demonstrating the benefits of fine-tuning large language models (LLMs) for domain specificity and the effectiveness of Retrieval-Augmented Generation (RAG) in enhancing contextual accuracy, we aim to integrate both approaches in the development of **Disha**, the chatbot for **IIIT Nagpur**. By combining the advantages of fine-tuning LLMs for tailored responses with RAG's capability to retrieve relevant external context, we anticipate superior results in terms of accuracy, scalability, and user experience, ensuring robust performance across various domains within the university.

Chapter 3

Proposed Models and Techniques

This section discusses the models and methods proposed for Disha's development. It includes an overview of Large Language Models like LLaMA and Phi, a comparison of various LLMs, fine-tuning techniques, and the implementation of Retrieval-Augmented Generation (RAG), highlighting their roles in achieving efficient and accurate results.

3.1. Large Language Models - LLaMA & Phi

3.1.1. LLMs

Large Language Models are highly advanced AI systems that can both understand as well as generate basically human-like text. These models are based on deep learning techniques and are trained on large datasets of books, articles, and web content. In contrast, LLMs have huge parameters in the order of billions that enable generating coherent and contextually accurate responses across a wide spread of domains. Its transformer-based architecture and vast training data enable it to perform very well in text generation and summarization.

Key Features of LLMs:

- **Autoregressive Generation**: LLMs such as GPT are autoregressive and generate text token by token using the earlier generated tokens as context.
- **Conditional Generation**: They output text according to prompts or specified inputs, thus allowing for custom output.
- **Fine tuning** : Models can be fine-tuned to particular tasks by training on smaller task-specific datasets.

Advantages of LLM :-

LLMs show remarkable versatility and can therefore be adapted to a broad set of tasks, from answering complex questions to the generation of creative content such as poems, stories, and even code. Efficiency comes from the ability of these LLMs to automatically perform tedious text-based processes, significantly reducing time and effort needed for document summarization, translation, and other forms of content generation. Additionally, LLMs enhance accessibility by supporting multiple languages, enabling users from diverse linguistic backgrounds to interact seamlessly. This combination of adaptability, efficiency, and inclusivity makes LLMs powerful tools for individuals and organizations across various domains.

Architecture:-

The architecture of LLMs consists of embedding layers, feedforward layers, and self-attention mechanisms. A word is an encoding in a vector that corresponds to its semantic and syntactic properties within the embedding layer. Self-attention mechanisms enable the model to zoom in on relevant text parts, ensuring nuanced understanding. When put together, these elements enable LLMs to process complex language structures effectively. Their recurrent and feedforward layers enhance abstraction and coherence enough to produce high-quality, human-like text.

3.1.2. LLaMA - Meta

By Meta AI in February 2023, the Llama series, or Large Language Model Meta AI, refers to a line of autoregressive language models based on NLP capabilities that relate to comprehensive reasoning, summarization, and language comprehension. The parameter sizes range from lightweight, 1B, to enormous, 405B. Among these models, **Llama 2** brought greater accessibility and enhanced performance to culminate into **Llama 3.2**

Llama 3.2.

Released in September 2024, Llama 3.2 features text-only lightweight models (1B & 3B) for mobile and edge devices, along with multimodal models (11B & 90B) for image-text reasoning. The lightweight versions have a 56% size reduction and up to 3x speed improvements over predecessors, making them ideal for on-device deployment.

Llama 3.2 contains major advances: the integration of lightweight text-only models, which would efficiently utilize mobile and edge devices; multimodal image-text reasoning capabilities. It benefits from structured pruning from 8B model for size reduction without loss of performance and utilizes logits from even bigger models to provide knowledge distillation capabilities from 8B & 70B. It has the optimality of long-range context support up to 128K tokens; an optimal set of alignment techniques like SFT, RS, and DPO; and high-quality synthetic datasets in summarization, reasoning, and efficient edge deployment on Qualcomm, MediaTek, and Arm chips.

1B & 3B Lightweight Models

The 1B and 3B models are designed for efficient deployment on mobile. They were developed using pruning and knowledge distillation techniques, achieving a compact size without losing critical performance.

- **Pruning**: Structured pruning applied to the 8B model reduces size while retaining essential knowledge.
- **Knowledge Distillation**: Outputs from 8B and 70B models guide pre training for better performance in smaller models.
- **Post-Training Alignment**: SFT, RS, and DPO ensure fine-tuning for summarization, instruction-following, and conversational tasks.
- **Use Cases**: Suitable for rewriting, summarization, and conversational AI in resource-constrained environments.

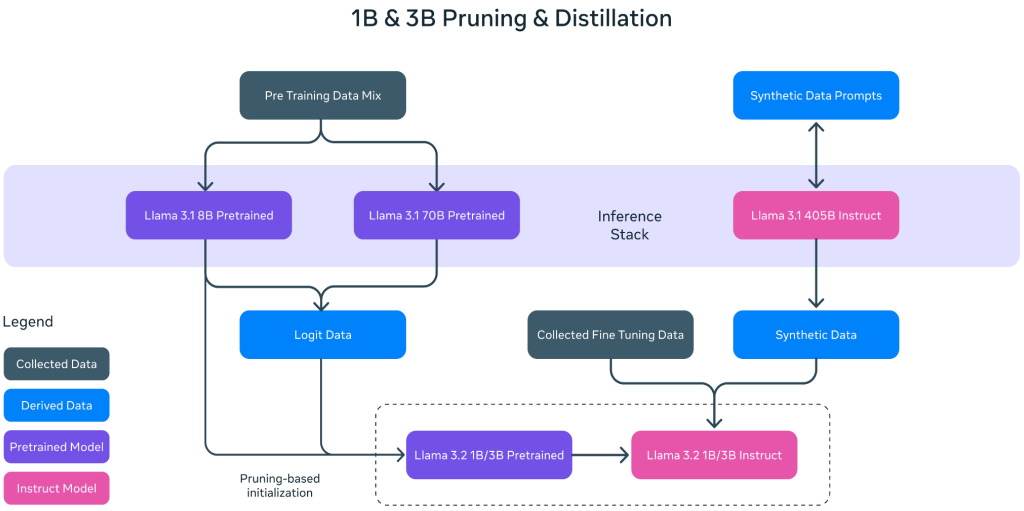

Figure 3.1 : 1B & 3B Pruning & Distillation [7]

The development of **Llama 3.2** lightweight models is done in a structured workflow. Data will first undergo **Llama 3.1** pretrained models, 8B and 70B, whose outputs or logits form the process to train a smaller model in place. Structured pruning in Figure 3.1 is applied to the 8B model, reducing its size while retaining knowledge. Fine-tuning combines alignment techniques and synthetic data to refine the 1B and 3B models, which are further aligned to create instruction-tuned variants, ensuring compact, high-performance models for edge and mobile applications.

Why Choose 1B ?

The 1B lightweight model is perfect for our project, in which we want to handle structured teacher details and generate accurate, real-time responses through fine-tuning. Its compact size and efficient performance make it ideal for deployment on mobile with limited resources. This will not only serve to maximize the accuracy of the model but also retain minimal computational overhead, achieved through pruning and distillation. We can fine-tune it so that it precisely understands and responds to domain-specific tasks-specifically, summarizing answering queries, and generating quick, reliable outputs.

Comparison of Llama 3.2 Models

Table 3.1 : Comparison of Llama 3.2 Models

Feature	1B & 3B Lightweight	11B & 90B Multimodal	Llama 3.1 Models
Parameters	1B, 3B	11B, 90B	8B, 70B
Focus	Text-only, Edge	High-Resolution Image	General NLP Tasks
Context Length	128K Tokens	64K Tokens	32K Tokens
Deployment	Edge Devices	Server-Level	General
Multilingual	Yes	Limited	Yes
Pruning	Yes	No	No

3.1.3. Phi-3.5

Phi-3.5-mini is a state-of-the-art language model featuring **3.8 billion parameters**. Designed for tasks that require multilingual support and long-context handling, it stands out for its 128K token context length. This allows the model to handle complex tasks such as document summarization, multi-turn conversations, and information retrieval efficiently. Through supervised fine-tuning and advanced optimization, including proximal policy optimization, **Phi-3.5-mini** excels at reasoning, coding, and logical tasks.

Key Features

- **3.8 Billion Parameters**: A compact size with high performance.
- **128K Token Context Length**: Handles long documents seamlessly.
- **Efficient Performance**: Comparable to larger models, but optimized for lower memory and compute requirements.

Performance and Efficiency

Unlike traditional large models that require heavy computational resources, **Phi-3.5-mini** is designed to be lightweight, delivering similar performance to larger models. Its optimization allows it to handle tasks that demand high computational power, such as long-context analysis and coding, but with a smaller footprint. This makes it an ideal choice for deployment in resource-constrained environments..

Ideal for Fine-tuning Projects

Phi-3.5-mini's compact size and specialized focus make it particularly suitable for fine-tuning tasks. Whether the project involves processing large, detailed conversations or extracting data from long documents, **Phi-3.5-mini** excels in scenarios where multilingual support and long-context handling are essential. Additionally, its advanced safety features ensure that the model can be deployed in sensitive applications, providing a more reliable and ethical option for developers.

3.2. Comparison of different LLMs

Table 3.2 : Comparison of different LLMs

Feature	Phi-3.5-mini	Llama 3.2	GPT-4	PaLM 2	LaMDA
Parameter Size	3.8B	1B–90B	High (Not disclosed)	Small & Large Variants	Optimized for NLP (Not disclosed)
Token Limit	128K	Varies	Up to 32K	Up to 100K	Not specified
Open Source	Yes	Yes	No	No	No
Strengths	Long-context Multilingual	Flexible deployment Vision-reasoning	Advanced text generation	Adaptability Multilingual	Conversational AI-Dialogues

3.3. Fine Tuning

Fine-tuning adapts a pre-trained model to a specific task by training it further on domain-specific data. This process refines the model's weights, ensuring higher accuracy for specialized tasks like recognizing unique terminologies or accents. It leverages the model's existing knowledge, requiring less data and computation for optimal performance.

3.3.1. Fine Tuning LLMs

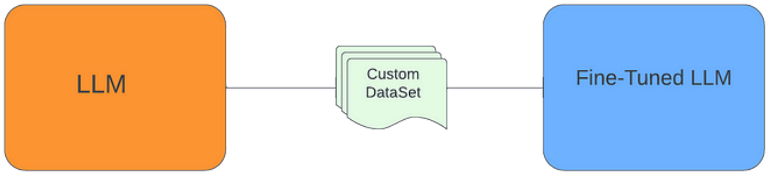

Figure 3.2 : Introduction to Fining Tuning LLMs [6]

Fine-tuning a large language model means just fine-tuning a pre-trained model for specific tasks or domains by further training it on a smaller, domain-specific dataset see Figure 3.2. The pre-trained models, such as the GPT series developed by OpenAI or Meta's **LLaMA** models, are typically trained on massive corpora so that they can gain general patterns and structures of the language. Fine-tuning rests on that base and adjusts the parameters by using specific task data. This approach is more computationally friendly compared to training from scratch and enables the model to perform better in specialized applications, such as sentiment analysis, summarization, translation, or domain-specific question-answering systems.

Steps for Fine Tuning

In fine-tuning, the major steps include: selecting the best pre-trained model, gathering and preprocessing a significant relevant dataset for training the model

on curated data.

Let W_0 denote the weights of the pre-trained model. The model updates its weights W in fine-tuning to better suit the specific task:

$$W = W_0 + \Delta W$$ ---------- eq i

where ΔW denotes the target-domain-specific updates learned.

The procedure retains the general linguistic knowledge from pretraining and combines it with task-specific nuances. Even though the method is effective, training on all model parameters incurs large computational costs. This has prompted researchers to work on methods that are more parameter-efficient than full fine-tuning, such as **Low-Rank Adaptation (LoRA)** and **Quantized LoRA (QLoRA)**. These advances allow for fine-tuning to be performed even with very modest hardware configurations and allows for greater accessibility and utilisation in real-world applications.

3.3.2. Parameter Efficient Fine Tuning - LORA & QLORA

PEFT

Large Language Models (LLMs) with billions of parameters demand substantial computational resources for full fine-tuning, making it challenging to deploy these models on resource-constrained hardware. **Low-Rank Adaptation (LoRA)** and **Quantized LoRA (QLoRA)** address this issue by significantly reducing memory and computational requirements, enabling efficient fine-tuning for task-specific applications.

Low-Rank Adaptation (LoRA)

LoRA introduces a novel approach to fine-tuning by freezing the pre-trained model weights (W_0) and introducing two smaller matrices, A and B, to

approximate the weight updates (ΔW):

$$\Delta W = BA \qquad \text{---------- eq ii}$$

Here:

- B has dimensions $d \times r$,
- A has dimensions $r \times k$,
  where r (rank) is much smaller than d (output size) and k (input size).

This reduces the number of trainable parameters to:

$$\text{Trainable Parameters} = (d + k) \cdot r \qquad \text{--------- eq iii}$$

Compared to full fine-tuning, which requires d · k parameters, this method is far more efficient. During inference, the updated weights are computed as:

$$W = W_0 + \Delta W = W_0 + BA \qquad \text{---------- eq iv}$$

The original model weights (W_0) remain unchanged, ensuring the base model is reusable across multiple tasks. This prevents "catastrophic forgetting," where the model loses its general language understanding, while allowing for lightweight task-specific adaptation (often in MBs).

Quantized LoRA (QLoRA)

QLoRA extends **LoRA** by incorporating weight quantization, further reducing memory requirements. In **QLoRA**, the pre-trained model weights are quantized to lower precision (e.g., 4-bit) to minimize storage without sacrificing performance. Let Q(W_0) represent the quantized version of W_0. The computation for inference:

$$W = Q(W_0) + \Delta W = Q(W_0) + BA \qquad \text{---------- eq v}$$

This method not only retains the efficiency of **LoRA** but also substantially decreases the memory footprint to fine-tune billion-scale models on resource-constrained hardware.

Comparison and Use Cases

Scenarios in which multiple domain-specific models must be built have been well-suited for **LoRA** and **QLoRA** because they allow freezing the base model and storing lightweight adapters only. This is effective in fine-tuning various models for different tasks with minimal storage and compute requirements. The further optimizations in QLoRA make it deployable on edge devices or GPUs, even with limited memory, enabling cost-effective and scalable customization of LLMs for applications such as chatbots, sentiment analysis, and domain-specific text generation. These methods represent a significant leap in making LLM fine-tuning accessible, efficient, and versatile for diverse real-world use cases.

3.3.4. Unsloth

Unsloth is a novel framework to fine-tune **Large Language Models** in the most optimized fashion. It increases the computations order and utilizes Triton kernels from OpenAI toward improved memory usage and training time, but with similar accuracy. It simplifies the matrix operations and removes redundant steps to be able to find efficiency that's valuable enough for such resource-hungry tasks like adapting LLMs to domain-specific applications.

Traditional Fine-Tuning vs. Unsloth

Comparing the Traditional Fine-Tuning with Unsloth In traditional fine-tuning, the computation of output involves two distinct matrix multiplications namely the pre-trained model weights, W_0, and LoRA matrices, A and B. It is often written as:

$$\text{Output} = XW_0 + X(BA) \quad \text{---------- eq vi}$$

Here:

- X: Input matrix
- W_0: Pre-trained weights
- A, B: LoRA matrices

The computation becomes redundant because the operations on W_0 and BA occur separately. Unsloth minimizes this by compressing multiple operations into one step:

$$\text{Output} = XW_0 + X(BA) \quad \text{---------- eq vii}$$

Direct update merging with the frozen weights, W_0, of **Unsloth** reduces the memory and computational overhead. This optimized operation bypasses unnecessary intermediate steps that would be consumed during its execution, ensuring speed without performance penalties.

Advantages of Unsloth

- **Improved Training Speed**

  Unsloth achieves up to **8.8x faster training times**. For instance, fine-tuning an LLM on the Alpaca dataset with Unsloth reduced the training time from 23 hours to just 2.5 hours on an NVIDIA T4 GPU.
- **Reduced Memory Usage**

  The restructured computation pipeline minimizes memory requirements, making it possible to fine-tune billion-parameter models even on resource-constrained hardware.
- **Triton Kernels for GPU Optimization**

  By rewriting PyTorch modules with Triton, Unsloth ensures maximum efficiency for GPU operations, enabling faster matrix computations and better utilization of hardware resources.
- **Compatibility with Existing Tools**

  Fully integrated with frameworks like Hugging Face's **PEFT** and **TRL**, Unsloth supports a wide range of NVIDIA GPUs, from consumer-grade cards to high-performance accelerators.

3.4. Retrieval Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) combines a retriever and a generative model to provide accurate, context-based answers. The retriever fetches relevant information, which the generative model uses to create informed responses. This approach ensures up-to-date and domain-specific accuracy, making it ideal for applications like chatbots and search systems.

3.4.1. Vector Database

Vector Embeddings

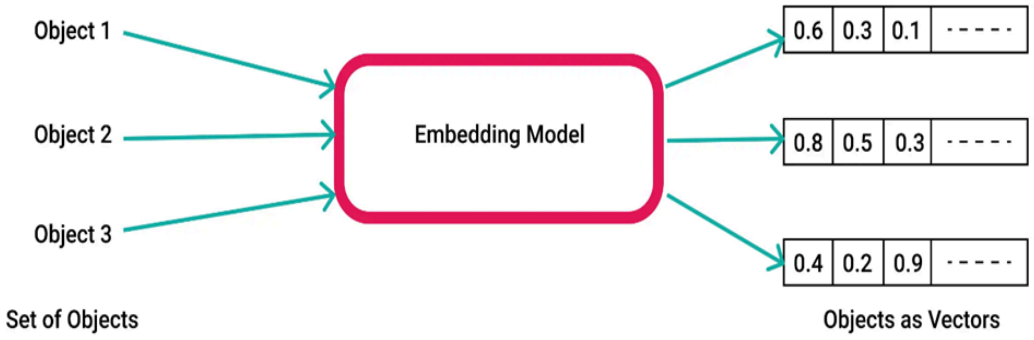

Figure 3.3 : Vector Embeddings [8]

Vector Embeddings represent data through numerical forms, such as text, images, or audio, in a multi-dimensional space see Figure 3.3, thereby capturing the semantic meaning. It is further achieved by models like transformers. These vectors are highly important in enabling machines to observe patterns, relationships, and context-relationships in the given data. Applications of vector embeddings can be found very vividly in applications related to semantic search, natural language processing, etc. Unlike other traditional data types, embeddings are of high dimensionality and need special systems for storage, efficient querying, and scaling.

Vector Databases

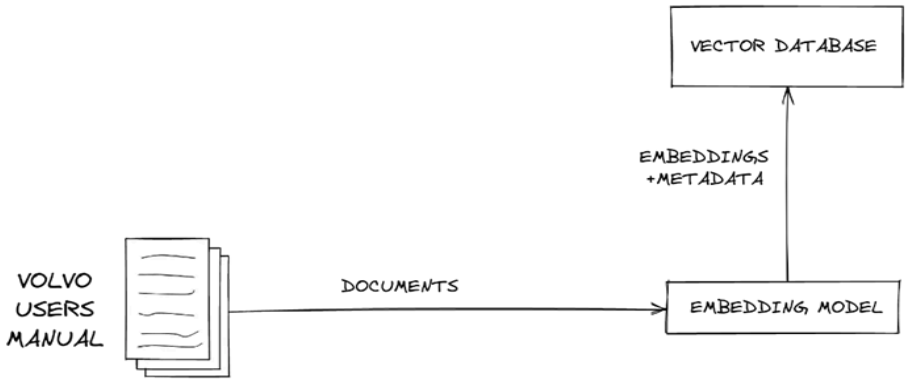

Figure 3.4 : Vector Databases [9]

Vector Databases are particular structures for the storage and management of these embeddings, allowing for efficient similarity-based searching by making use of **Approximate Nearest Neighbor (ANN)** methods. Since traditional databases rely on exact matchings, vector databases are designed to identify the similarity between embeddings using similarity measures, which becomes an important aspect in current AI applications. Applications such as semantic search and recommendations require optimized querying, scalability, and real-time performance.

3.4.2. How Vector Databases Work

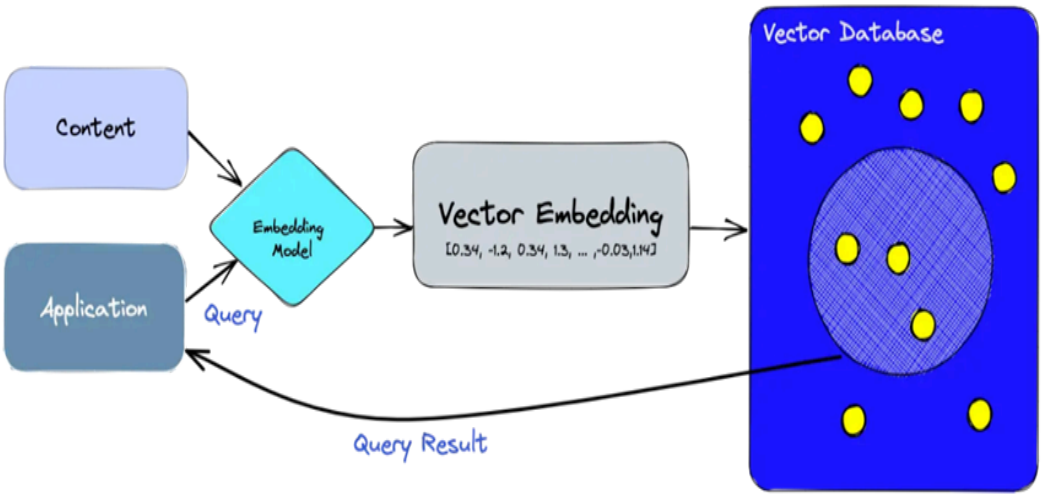

Figure 3.5 : How Vector Databases Work [10]

In Figure 3.5, the operation of vector databases involves several key steps. First, raw data is transformed into vector embeddings, typically using models such as **BERT** or **Gemini APIs**. These embeddings are then indexed, alongside relevant metadata, for easier and faster retrieval. The similarity metrics, cosine similarity, Euclidean distance, or dot product enables a database to compute the embedding of a query and compare it with the stored vectors and retrieve the nearest neighbors. Cosine similarity measures the angle between two vectors whereby values closer to 1 indicate identical vectors. Euclidean distance computes the straight-line distance between the vectors, therefore detecting differences in magnitude. On the other hand, the dot product measures the geometric overlap between vectors , assessing their contribution.

3.4.3. Pinecone Vector Database

Pinecone is a managed **vector database** for fast similarity search and easy integration with AI pipelines. It is particularly well-suited for semantic search, question answering, and recommendation systems applications. With the help of large language models, pinecone transforms raw data into vector embeddings for semantic comparisons. It provides customizable indexing where users can fix dimensions and similarity metrics such as **cosine similarity** or **dot product**, besides ingestion of embeddings with metadata for filtering and multi-tenancy purposes, providing scalable data handling.

The database further provides similarity search to retrieve semantically similar vectors, which proves indispensable for document retrieval purposes. Performance is optimized with hybrid search capabilities, metadata filtering, and reranking, which improve query accuracy. Pinecone delivers a fast and scalable solution with low latency while being able to achieve high recall even for huge datasets, and ensures enterprise-grade security certifications, like SOC 2 and HIPAA. Its flexibility extends further to integrations with popular cloud platforms such as AWS, GCP, and Azure and frameworks such as **LangChain** and **OpenAI**.

3.4.4. Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) is an advanced AI technique that combines the power of **Large Language Models (LLMs)** with a retrieval mechanism, thereby enhancing the accuracy and relevance of generated content. **RAG** does this by drawing contextually relevant data from external databases at generation time, especially incorporating real-time and domain-specific information that ensures the output is both precise as well as context-aware. Thus, applications in legal research, customer support, or technical troubleshooting can benefit from **RAG**.

Overcoming LLM Limitations with RAG

Large language models (LLMs) like **ChatGPT** or **Claude** are powerful but have limitations, including reliance on static, pre-trained knowledge, lack of real-time data, high operational costs, and the potential for generating inaccurate responses ("hallucinations"). **RAG (Retrieval-Augmented Generation)** overcomes these challenges by integrating external databases, providing real-time, relevant context during generation. This reduces hallucinations and ensures the inclusion of up-to-date, domain-specific data. Moreover, **RAG** improves transparency by allowing source citation and makes the output more verifiable and reliable.

RAG Architecture

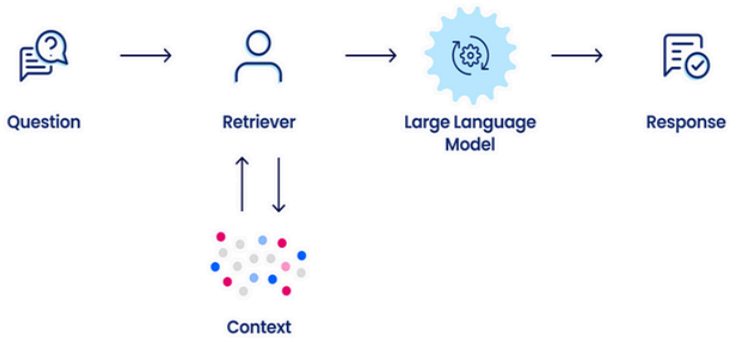

Figure 3.6 : RAG Architecture [11]

RAG comprises two main components, which are the retriever and the generator. The retriever is designed to fetch relevant information from an external database using various types of retrievers. Dense retrieval in the form of neural networks picks up deep semantic relationships, whereas sparse retrieval (keyword based) excels at exact matches. A hybrid approach that balances good performance and accuracy combines both dense and sparse methods. The generator uses the retrieved context to generate a very accurate, detailed response enriched with relevant external data.

Retrieval-Augmented Generation System Workflow

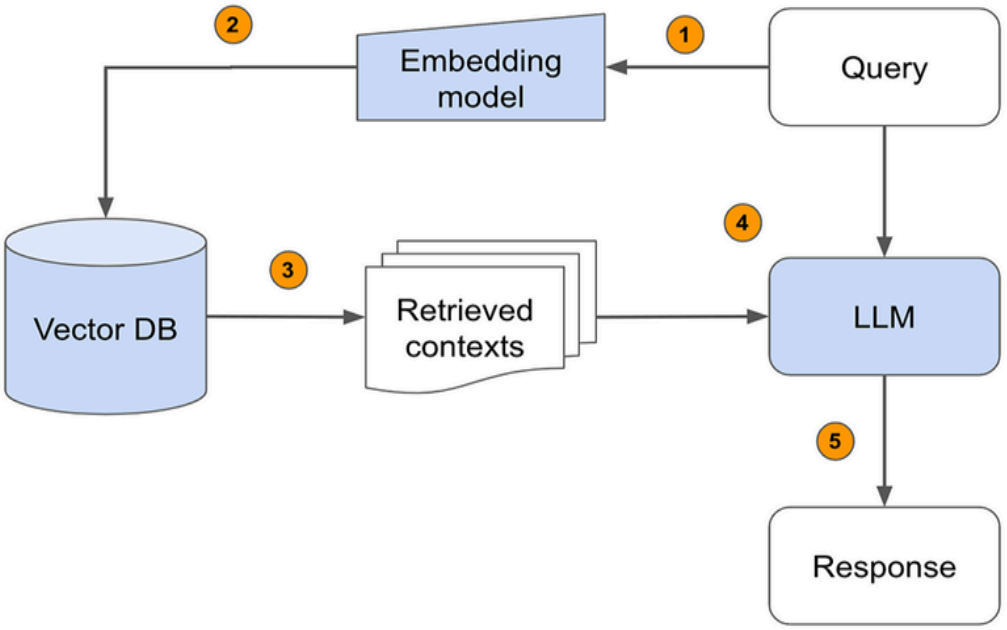

Figure 3.7 : RAG System Flow [12]

In Figure 3.7, steps involved in RAG System:-

1. **Query Processing**: It begins with a user query, for ex., a question or prompt.
2. **Embedding Model**: Converts the query into a vector representation for computation and searches in a vector database using the query vector to find the contexts most similar to the query.
3. **Context Retrieval**: Retrieves passes to the **LLM**, which generates responses for inputted queries.

4. **LLM Response Generation**: Combines query and retrieved contexts to produce an accurate and context-enriched response
5. **Final Output**: The system generates a response that has been enhanced through use of external data for more precision and relevance.

3.4.5. Selecting a Retriever

The choice between dense and sparse retrievers depends on the nature of the data and the expected queries. While computationally more expensive, dense retrievers are better at capturing deep semantic relationships and understanding context. Sparse retrievers are faster and do their job better at finding exact term matches for simpler or very specific queries.

Chapter 4

Implementation Details

This section outlines the comprehensive workflow and steps involved in building Disha. It covers the dataset preparation, fine-tuning methods, Retrieval-Augmented Generation (RAG) techniques, integration of final models, and evaluation using performance metrics, providing a detailed understanding of the system's design and functionality.

4.1. Flowchart

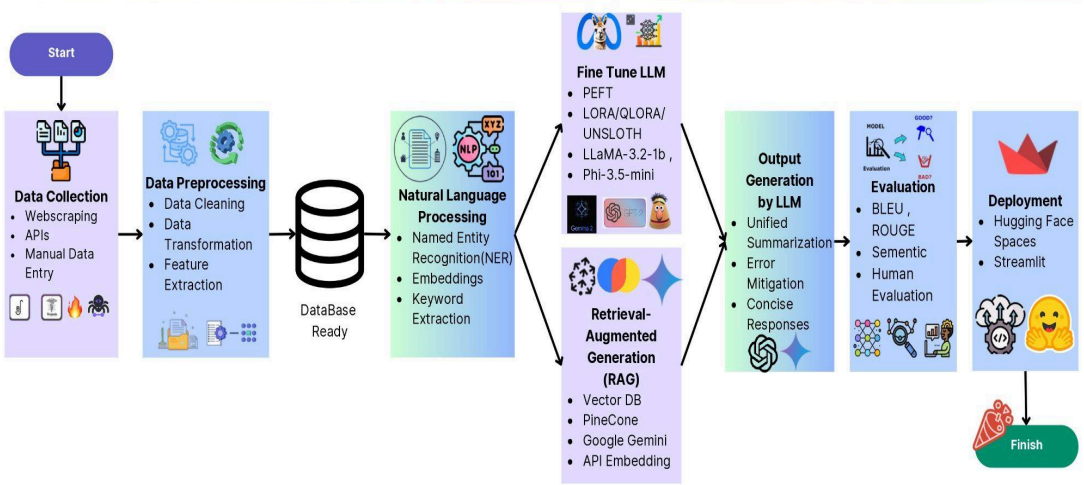

Figure 4.1 : Disha Workflow

Steps Involving in Disha :-

- **Data Collection**:
  The process begins with gathering raw data through multiple sources such as web scraping, APIs, and manual data entry. This ensures the chatbot's database covers a wide range of relevant information from the IIIT Nagpur domain, including textual and visual data from websites.

- **Data Preprocessing**:

  Collected data is cleaned, transformed, and enriched with features to create a structured, high-quality dataset. This step ensures the data is noise-free and formatted appropriately, which is crucial for downstream tasks like embeddings and language model fine-tuning.

- **Natural Language Processing (NLP)**:

  The refined data is processed using **NLP** techniques, including **Named Entity Recognition (NER)** for extracting meaningful entities, embeddings for vector representation, and keyword extraction to highlight relevant information. This step converts raw text into actionable insights.

- **Fine-Tuning LLMs**:

  Pre-trained language models such as **LLaMA-3.2-1B** and **Phi-3.5-mini** are fine-tuned using techniques like **LoRA**, **QLoRA**, or **Unsloth**. Parameter-efficient fine-tuning methods ensure the chatbot can adapt to domain-specific tasks with reduced computational preserving accuracy.

- **Retrieval-Augmented Generation (RAG)**:

  A **retrieval-based** approach combines vector database systems like Pinecone with API embeddings to integrate external knowledge dynamically. This allows the chatbot to provide accurate, contextually relevant responses

- **Output Generation by LLM**:

  The **fine-tuned LLM** generates responses that are concise, error-mitigated, and unified in summarization. The focus is on delivering high-quality, informative answers to user queries, improving overall user experience.

- **Evaluation**:

  The generated outputs are evaluated using metrics such as **BLEU**, **ROUGE**, and **Semantic Similarity**, alongside **human evaluation**.

- **Deployment**:

  The final **RAG Version** of chatbot is deployed using platforms like **Hugging Face Spaces** and **Streamlit**.

4.2. Dataset

4.2.1. Data Acquisition Process

The primary step of the dataset formation process was to acquire the most diverse set of information from official sources like the official website of IIIT Nagpur, PDF files, and images. Using **OCR** tools and manual entry. The acquired data included many distinctive pieces of information related to personal details, experience, educational qualifications, research work and publications for all departments, CSE, ECE, and BS.

4.2.2. Data Categorization and Structuring

The collected data was categorized into multiple predefined fields to standardize information. For faculty, they included name, department, position, contact details, qualifications, teaching experience, research interests, and supervision records. Timetable data was reformatted into a structured JSON format that included lab names, schedules, and additional information. PDFs containing rules and regulations, academic calendars, and project details were converted into machine-readable structured JSON files, ensuring consistency and usability.

```
print(df.iloc[0]["Name"])
print(df.iloc[0]["Details"][:320])
```

```
cse Dr Prerna Mishra
{
"name": "Dr Prerna Mishra",
"department": "Computer Science and Engineering",
"position": "Adjunct Assistant Professor",
"phone no": "9834485669",
"email": "pmishra@iiitn.ac.in",
"joining date": "03-01-2023",
"education": [],
"teaching experience": [
{
"institution": "Indian Institute of Information Technology Nagpur
```

```
print(df.iloc[68]["Name"])
print(df.iloc[68]["Details"][:320])
```

```
cse Dr Mangesh Ramaji Kose
{
"name": "Dr Mangesh Ramaji Kose",
"department": "Computer Science and Engineering",
"position": "Assistant Professor",
"phone no": "7224931314",
"email": "mkose@iiitn.ac.in",
"joining date": "2024-06-30",
"education": [
{
"degree year": "2015",
"college name": "Prof. Ram Meghe Institute of Technology and Research",
"
```

4.2.3. Data Preprocessing

Preprocessing involved cleaning the data to remove inconsistencies and clutter. Techniques such as string normalization, typo correction, and deduplication were used. Missing information such as positions and degrees was addressed. Text data was pre-processed utilizing **NLP techniques**, including keyword extraction and named entity recognition in order to enrich the dataset and make it more valuable. This ensured that all entries in the data conformed to a form of orderliness.

4.2.4. Conversion to JSON Format

All preprocessed data were serialized into a standardized **JSON format** so that it can easily be integrated into downstream applications. Each record comprised a whole **JSON object** with all the information about the faculty member or the institution. For instance, for single faculty members, the JSON files contained nested structures for degrees, projects, and publications among others. Lab timetables were encoded into **JSON objects** where schedule details could be mapped to individual days and times.

```
print(df.iloc[162]["Name"])
print(df.iloc[162]["Details"][:320])
```

```
Library
{
"libraryDetails": {
"libraryName": "Central Library",
"description": "The Central Library is a Reading Hall where 50 readers can be
accommodated at a time, with one stack room and a collection of about 3191 books and 144
CDs. The library has a rich collection of 362 titles in the field of Electronics and
Communicatio
```

```
print(df.iloc[176]["Name"])
print(df.iloc[176]["Details"])
```

```
head of departments ( HoD )
{
"departmentHeads": [
{
"departmentName": "Department of Basic Science and Engineering",
"headName": "Dr. Prasad V. Joshi",
"headTitle": "Assistant Professor & HOD",
"phoneNumber": "9893195909",
"email": "pjoshi@iiitn.ac.in"
},
{
"departmentName": "Department of Computer Science and Engineering",
"headName": "Dr. Nishat A. Ansari",
"headTitle": "Assistant Professor & HOD",
"phoneNumber": "9766223703",
"email": "nishat.ansari@iiitn.ac.in"
},
{
"departmentName": "Department of Electronics and Communication Engineering",
"headName": "Dr. Harsh Goud",
"headTitle": "Assistant Professor & HOD",
"phoneNumber": "9827554326",
"email": "hgoud@iiitn.ac.in"
}
]
}
```

4.2.5. Dataset Compilation

Once the individual **JSON files** were created, they were consolidated into a master **CSV file**, ***`iiitn.csv`***, with two columns: Name and Details containing the **JSON objects**. The dataset consisted of 285 rows, which showed wide variability in terms of individuals and institutional data. In addition, the test dataset ***`iiitn_test.csv`*** was prepared by randomly sampling 50 rows from the main dataset for testing and validation purposes to be used during model training and evaluation.

4.2.6. Comprehensive Details

The faculty and institutional data were covered under the dataset completely. Faculty profiles accounted for academic and industrial experience, contributions in development of laboratories, projects, workshops, fellowships, and other details regarding publications. Data on supervision for undergraduate and postgraduate dissertations was also included. Institutional data included rules, regulations, achievements, schedules of the academic calendars, and detailed schedules for lab management to improve resource planning.

```
print(df.iloc[220]["Name"])
print(df.iloc[220]["Details"][:320])
```

```
timetable Signal Processing Lab
{
"Lab Name": "Signal Processing Lab",
"Schedule": {
"Monday": {
"10:00-12:00": "S&S (ECE-II-A2)",
"14:00-16:00": "HDL (ECE-III-B2)",
"16:00-18:00": "S&S (ECE-II-B2)"
},
"Tuesday": {
"10:00-12:00": "S&S (ECE-II-A1)",
"14:00-16:00": "S&S (ECE-II-B1)",
"16:00-18:00": "AS&S (IoT-II-A1)"
},
"Wednesday": {
"10:00-12:00": "D
```

```
print(df.iloc[223]["Name"])
print(df.iloc[223]["Details"][:320])
```

```
kshitij
{
"event name": "KSHITIJ",
"description": "More than a mere sports festival, KSHITIJ is a dynamic platform where
athletes spotlight their prowess, teamwork takes center stage, and the college spirit
soars. Our goal is to cultivate unity, sportsmanship, and healthy competition among
students - an experience that transce
```

4.2.7. Data Validation and Verification

After compilation, the dataset was validated to check for accuracy and consistency of the information. Random entries were cross-checked against the original data sources with no critical pieces of information being missed. The dataset was then purged of duplicate entries, and great care was taken to ensure each **JSON** object complied with the necessary schema. This made the dataset reliable and prepared it for downstream tasks.

	Name	Details
111	Academic Programme	aculty incharge qip": { "name": "Dr. Amol Bhopale", "title": "Assistant Professor", "department": "Department of Computer Science & Engineering", "email": "qip@iiitn.ac.in" } } }
112	admission	reless Sensor Network, and IoT", "Microwave and Antennas", "Signal, Image & Video Processing", "Process Control system", "Intelligent Systems"] } }, "additional info": "NA" }
113	director Dr Prem Lal Patel	rica", "Netherlands", "China", "Italy", "Japan", "Dubai"], "awards": ["Member of climate changing working group of IAHR dealing with Fluvial Mechanism"], "additional info": [] }
114	Phd Admission	from the date of joining until thesis submission for EX-A, IN-AS, IN-U, IN-ES categories.", "Industry Candidates": { "With Masters": "3 years", "With Bachelors": "4 years" } } } }
115	registrar Shri Kailas N Dakhale	f Technology (IIT), Jodhpur"], "additional info": { "previous positions": ["Deputy Registrar at IIT Kanpur", "Joint Registrar at VNIT, Nagpur"], "industry experience": "7 years" } }
116	TNP	garwal for their achievement and wish all the best for future endeavors." }], "Placements BTech Batch 2020": { "Highest Package": "20 LPA", "Average Package": "8.5 LPA" } }
117	admission and clubs	" } }, "Flagship Annual Event at IIITN": { "ABHIVYAKTI": "The Annual Cultural Festival", "TANTRAFIESTA": "National Level Technical Event by IIITN", "additional info": "NA" } } }
118	sionCancellationAndRefundPolicy	decision is final on all cases.", "4": "The Institute may modify this policy without notice.", "5": "Legal disputes will be settled in Nagpur, Maharashtra." }, "additional info": "NA" } }
119	Anti Ragging Notice	], "contact information": { "antiragging helpline email": "helpline@antiragging.in", "antiragging helpline number": "1800-180-5522" }, "additional info": "24x7 Toll Free Number." }
120	fee details	me of account": "Indian Institute of Information Technology, Nagpur Hostel", "account no": "41759639981", "ifsc code": "SBIN0006702", "branch name": "VRCE BRANCH" } } }
121	hostel fee details	payable at the start of the On-Campus Semester.", "Students residing for the full year in hostel would have to pay full fees as applicable for 2nd/3rd Year for 4th year also." } } }
122	important documents	writing Guidelines", "link": "https://api.iiitn.ac.in/data/IIITN/2024/August/ContentDocs/CN ID 1724998344705jMGpH9Hv5 95bNdk/1724998259697kxEvPWr32PSAKaC.pdf" } } }
123	internship program1	y of faculty members offering internships after testing and verifying the grades and technical skills. Applying to the internship does not guarantee the selection of students." } }}
124	internship program2	rnshipApplication": "Application for joining internship Download", "InternshipReportGuidelines": "Internship Report Submission Guideline Download" } }, "additional info": "NA" }
125	officials staffs	n.ac.in", "phone": "8805813037" }, { "name": "Shri Ashok Dongare", "department": "Admin", "designation": "Member", "email": "jecivil@iiitn.ac.in", "phone": "9689009639" }] }] }
126	officials staffs2	n & Redressal) Act, 2013.", "link": "https://iiitn.ac.in/18092024/Policy%20of%20Prevention%20of%20Sexual%20Harassment%20of%2 0Women%20at%20Workplace.pdf" }] }
127	postgraduate	hD": { "Regular Candidates": "3 years from the date of joining till the submission of thesis", "Industry Candidates": { "With Masters": "3 years", "With Bachelors": "4 years" } } } }
128	reporting and cutoff	Caste Validity is required for Maharashtra candidates.", "Fees Structures might not be updated please check the college website for exact fees structure."], "additional info": [] }
129	right to inform	Shri Kailas N. Dakhale", "position": "Registrar", "organization": "Indian Institute of Information Technology, Nagpur", "phone": "8087983449", "email": "registrar@iiitn.ac.in" } } } }
130	rules and regulations	discipline and restraint in interactions.", "Avoid visiting junior's rooms or inviting juniors to senior rooms.", "Do not force juniors to perform under the guise of talent search." } } }
131	undergraduate	Communication Engineering", "Total Seat Intake": 150 }, { "Program Name": "B. Tech. ECE (Internet of Things)", "Total Seat Intake": 66 }], "Total Undergraduate Seats": 637 }
132	admission and clubs	" } }, "Flagship Annual Event at IIITN": { "ABHIVYAKTI": "The Annual Cultural Festival", "TANTRAFIESTA": "National Level Technical Event by IIITN", "additional info": "NA" } } }
133	sionCancellationAndRefundPolicy	decision is final on all cases.", "4": "The Institute may modify this policy without notice.", "5": "Legal disputes will be settled in Nagpur, Maharashtra." }, "additional info": "NA" } }
134	Anti Ragging Notice	], "contact information": { "antiragging helpline email": "helpline@antiragging.in", "antiragging helpline number": "1800-180-5522" }, "additional info": "24x7 Toll Free Number." }
135	fee details	me of account": "Indian Institute of Information Technology, Nagpur Hostel", "account no": "41759639981", "ifsc code": "SBIN0006702", "branch name": "VRCE BRANCH" } } }
136	gostel fee details	payable at the start of the On-Campus Semester.", "Students residing for the full year in hostel would have to pay full fees as applicable for 2nd/3rd Year for 4th year also." } } }
137	important documents	writing Guidelines", "link": "https://api.iiitn.ac.in/data/IIITN/2024/August/ContentDocs/CN ID 1724998344705jMGpH9Hv5 95bNdk/1724998259697kxEvPWr32PSAKaC.pdf" } } }

Figure 4.2 : Dataset

4.2.8. Final Dataset

The dataset was finalised as a complete, well-structured resource and forms the core of training and evaluating the chatbot. The structured **JSON** format and **CSV** aggregation make it versatile and easy to integrate with **NLP pipelines** for data preprocessing and model fine-tuning. This dataset captures all the critical aspects of faculty and institutional details in IIIT Nagpur, so the chatbot will be capable of delivering précised and detailed responses.

4.3. Fine Tuning

4.3.1. Libraries

- **Pip3-autoremove :** `**pip3-autoremove**` is a tool used to clean up unused Python packages, ensuring a clutter-free environment. It was used here to remove outdated PyTorch libraries, preventing compatibility issues during model fine-tuning.
- **torch, torchvision, torchaudio :** PyTorch (`torch`) is essential for deep learning tasks, providing tensor operations and neural network tools. `**torchvision**` aids image-based tasks, and `**torchaudio**` supports audio, though it's less relevant here. GPU-enabled versions optimize training speed for large models like **LLaMA-3.2-1B**.
- **Xformers :** `xformers` is crucial for memory-efficient Transformer model training. It optimizes attention mechanisms, reducing memory usage and improving training speed, which is essential for fine-tuning large-scale models like LLaMA and Phi.
- **Unsloth :** `unsloth` simplifies fine-tuning by providing tools for data loading, preprocessing, and integrating advanced techniques like LoRA.

4.3.2. Dependencies

- **Unsloth (FastLanguageModel, train_on_responses_only, is_bfloat16_supported) :** The `**unsloth**` library facilitates efficient training and inference of language models with tools like `**FastLanguageModel**`. It supports advanced optimizations such as bfloat16 precision, and features like `train_on_responses_only` streamline fine-tuning on dialogue datasets.

- **torch : PyTorch (`torch`)** is a foundational library for deep learning. It powers tensor computations and backpropagation for training language models like **LLaMA** and **Phi**. It's critical for GPU-accelerated fine-tuning.

- **Pandas :** `**pandas**` simplifies data manipulation and analysis. It's used to load and preprocess datasets from CSV files..

- **unsloth.chat_templates (get_chat_template, standardize_sharegpt) :** The `**unsloth.chat_templates**` module helps format and preprocess chat datasets. Functions like `**get_chat_template**` and `**standardize_sharegpt**` ensure alignment with training requirements for dialogue-based models.
- **datasets (load_dataset, Dataset) :** The `datasets` library provides tools for loading and processing datasets from diverse formats.
- **trl (SFTTrainer) :** `**trl**` or Transformers Reinforcement Learning, includes tools like `**SFTTrainer**` to fine-tune language models using supervised datasets. It simplifies training alignment for instruction-following and conversational models.
- **transformers (TrainingArguments, DataCollatorForSeq2Seq) :** The `transformers` library is key for working with pre-trained models. `TrainingArguments` handles hyperparameter configuration, while `DataCollatorForSeq2Seq` manages tokenization, padding during fine-tuning.
- **peft (PeftModel, PeftConfig) :** The `**peft**` library optimizes training through **parameter-efficient fine-tuning (PEFT)**. It reduces computational demands.
- **transformers (AutoModelForCausalLM, AutoTokenizer, TextStreamer) :** `AutoModelForCausalLM` and `AutoTokenizer` enable working with causal language models, while `TextStreamer` facilitates streaming outputs, particularly useful during inference and generation tasks.

4.3.3. Workflow

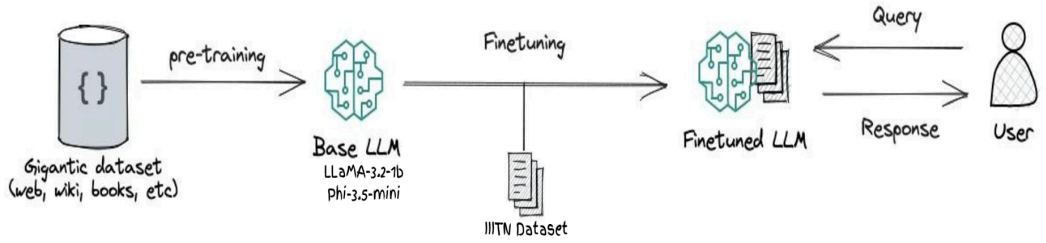

Figure 4.3 : Fine Tuning Workflow

4.3.4. Model Preparation

Model preparation involves curating data, fine-tuning algorithms and optimizing parameters to tailor a machine learning model for a specific task.

4.3.4.1. LLaMA-3.2-1b

We fine-tuned the **LLaMA-3.2-1B-Instruct** model from the **`unsloth`** library using 4-bit quantization (`load_in_4bit=True`) to minimize memory usage without sacrificing performance. A `max_seq_length` of 2048 supports long contexts, enhanced by **RoPE** scaling for extended positional embeddings. Automatic precision detection ensures compatibility, defaulting to `float16` for Tesla T4 GPUs or `bfloat16` for Ampere architectures.

Optimization leverages **Parameter-Efficient Fine-Tuning (PEFT)** with **Low-Rank Adaptation (LoRA)**, targeting critical modules like **`q_proj`** and **`k_proj`**. Configurations like **`r=8/16/32`**, zero dropout (`lora_dropout=0`), and alpha scaling (`lora_alpha=16`) balance efficiency and performance. Gradient checkpointing reduces VRAM usage, supporting larger batch sizes and extended context training, while features like **RSLORA** and **LoFTQ** improve stability and precision.

```
max_seq_length = 2048
dtype = None
load_in_4bit = True

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name="unsloth/Llama-3.2-1B-Instruct",
    max_seq_length=max_seq_length,
    dtype=dtype,
    load_in_4bit=load_in_4bit,
)

model = FastLanguageModel.get_peft_model(
    model=model,
    r=16,
    target_modules=["q_proj", "k_proj", "v_proj", "o_proj",
    lora_alpha=16,
    lora_dropout=0,
    bias="none",
    use_gradient_checkpointing="unsloth",
    random_state=3407,
    use_rslora=False,
    loftq_config=None,
)
```

4.3.4.2. Phi-3.5-mini

Phi-3.5-Mini is a lightweight, high-performance model suitable for fine-tuning tasks where memory and computational efficiency are key. The max sequence length is set to 2048, enabling long-context understanding, while dynamic data types (dtype = None) are leveraged to automatically select precision suitable for the GPU in use. **4-bit quantization** reduces memory usage and accelerates downloads without compromising model performance.

```
max_seq_length = 2048
dtype = None
load_in_4bit = True

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name="unsloth/Phi-3.5-mini-instruct",
    max_seq_length=max_seq_length,
    dtype=dtype,
    load_in_4bit=load_in_4bit,
)

model = FastLanguageModel.get_peft_model(
    model=model,
    r=16,
    target_modules=["q_proj", "k_proj", "v_proj", "o_proj",
    lora_alpha=16,
    lora_dropout=0,
    bias="none",
    use_gradient_checkpointing="unsloth",
    random_state=3407,
    use_rslora=False,
    loftq_config=None,
)
```

Pre-quantized models like **Phi-3.5-Mini-Instruct** allow seamless access to pre-tuned weights optimized for instruction-following tasks. Fine-tuning with **LoRA (Low-Rank Adaptation)** improves model efficiency, focusing only on specific layers (q_proj, k_proj, etc.) to reduce computational load. Gradient checkpointing is set to "unsloth" for efficient memory management during training. Together, these configurations ensure **Phi-3.5-Mini** is **fine-tuned** effectively while maintaining low resource demands.

4.3.5. Chat Template

4.3.5.1. LLaMA-3.2-1b

"**Name**" and "**Details**" that are transformed into a conversation format. The Name is assigned as input from the "**human**" side, and Details as output from the "**gpt**" side, creating a structured conversation between a user and **GPT**. The dataset is then converted into a Hugging Face Dataset and formatted using the get_chat_template function, which prepares the input for fine-tuning **LLaMA-3.1**. The function formatting_prompts_func applies the chat template to each conversation, ensuring the text is appropriately tokenized for model training. Lastly, the dataset is standardized and mapped for processing.

```
import pandas as pd
from datasets import Dataset
from unsloth.chat_templates import get_chat_template, standardize_sharegpt

df = pd.read_csv("/kaggle/input/iiitn-disha/iiitn.csv")

df['conversations'] = df.apply(lambda row: [
    {"from": "human", "value": row['Name']},
    {"from": "gpt", "value": row['Details']}
], axis=1)

df = df[['conversations']]

dataset = Dataset.from_pandas(df)

print(dataset[0])

tokenizer = get_chat_template(tokenizer, chat_template="llama-3.1")

def formatting_prompts_func(examples):
    convos = examples["conversations"]
    texts = [tokenizer.apply_chat_template(convo, tokenize=False, add_generation_prompt=Fa
    return {"text": texts}

dataset = standardize_sharegpt(dataset)
dataset = dataset.map(formatting_prompts_func, batched=True)
```

4.3.5.2. Phi-3.5-mini

"**Name**" and "**Details**", which are transformed into a conversation format where Name is labeled as "**human**" and Details as "**gpt**". This structure prepares data for fine-tuning a language model, where each row represents an

exchange between a human and an AI. The dataset is then converted into a Hugging Face Dataset and formatted using a chat template for the **Phi model**.

The formatting_prompts_func function processes the conversation data by applying the specified chat template, which aligns the roles and content. The dataset is standardized using standardize_sharegpt for further use in training.

```
import pandas as pd
from unsloth.chat_templates import get_chat_template, standardize_sharegpt
from datasets import Dataset

df = pd.read_csv("/kaggle/input/iiitn-disha/iiitn.csv")

df['conversations'] = df.apply(lambda row: [
    {"from": "human", "value": row['Name']},
    {"from": "gpt", "value": row['Details']}
], axis=1)

df = df[['conversations']]

dataset = Dataset.from_pandas(df)

print(dataset[0])

tokenizer = get_chat_template(
    tokenizer,
    chat_template="phi-3",
    mapping={"role": "from", "content": "value", "user": "human", "assistant": "gpt"}
)

def formatting_prompts_func(examples):
    convos = examples["conversations"]
    texts = [tokenizer.apply_chat_template(convo, tokenize=False, add_generation_prompt=F
    return {"text": texts}

dataset = standardize_sharegpt(dataset)
dataset = dataset.map(formatting_prompts_func, batched=True)
```

4.3.6. Training

Runs a training loop for fine-tuning a language model using the **SFTTrainer** class from the **Unsloth** library. The model, tokenizer, and dataset are passed to the trainer, along with configurations for the training parameters. The TrainingArguments object specifies key settings such as the batch size, number of epochs, learning rate, gradient accumulation steps, and weight decay. Additionally, mixed precision training is enabled based on the support for bfloat16 or fp16, optimizing memory usage and computation.

```
trainer = SFTTrainer(
    model = model,
    tokenizer = tokenizer,
    train_dataset = dataset,
    dataset_text_field = "text",
    max_seq_length = max_seq_length,
    data_collator = DataCollatorForSeq2Seq(tokenizer = tokenizer),
    dataset_num_proc = 2,
    packing = False,
    args = TrainingArguments(
        per_device_train_batch_size = 4,
        gradient_accumulation_steps = 4,
        warmup_steps = 5,
        num_train_epochs = 50,
        learning_rate = 0.001,
        fp16 = not is_bfloat16_supported(),
        bf16 = is_bfloat16_supported(),
        logging_steps = 1,
        optim = "adamw_8bit",
        weight_decay = 0.01,
        lr_scheduler_type = "linear",
        seed = 3407,
        output_dir = "outputs",
        report_to = "none",
    )
)
trainer_stats = trainer.train()
```

```
==((====))==  Unsloth - 2x faster free finetuning | Num GPUs = 1
   \\   /|    Num examples = 222 | Num Epochs = 50
O^O/ \_/ \    Batch size per device = 4 | Gradient Accumulation steps = 4
\        /    Total batch size = 16 | Total steps = 700
 "-____-"     Number of trainable parameters = 29,884,416
```

[700/700 7:41:50, Epoch 50/50]

Figure 4.4 : SFT Trainer

4.3.7. Model Saving

The model and tokenizer are saved locally using the `save_pretrained` method to ensure they can be reused for future tasks such as further training, inference, or deployment. By saving them to the `lora_model` directory, all necessary files (like model weights, configuration, special tokens, and tokenizer files) are stored in a way that they can be easily loaded back into memory for continued work or shared across different platforms. This approach ensures that the fine-tuned model and tokenizer are preserved, allowing for efficient use without retraining, making them ready for deployment or integration into applications.

```
model.save_pretrained("lora_model") # Local saving
tokenizer.save_pretrained("lora_model")
# model.push_to_hub("your_name/lora_model", token = "...") # Online saving
# tokenizer.push_to_hub("your_name/lora_model", token = "...") # Online saving
```

```
('lora_model/tokenizer_config.json',
 'lora_model/special_tokens_map.json',
 'lora_model/tokenizer.model',
 'lora_model/added_tokens.json',
 'lora_model/tokenizer.json')
```

4.3.8. Use the saved model

```
    from unsloth import FastLanguageModel
    model, tokenizer = FastLanguageModel.from_pretrained(
        model_name = "lora_model", # YOUR MODEL YOU USED FOR TRAINING
        max_seq_length = max_seq_length,
        dtype = dtype,
        load_in_4bit = load_in_4bit,
    )
    FastLanguageModel.for_inference(model) # Enable native 2x faster inference

messages = [
    {"from": "human", "value": "Hostel Rules"},
]
inputs = tokenizer.apply_chat_template(
    messages,
    tokenize = True,
    add_generation_prompt = True, # Must add for generation
    return_tensors = "pt",
).to("cuda")

from transformers import TextStreamer
text_streamer = TextStreamer(tokenizer, skip_prompt = True)
_ = model.generate(input_ids = inputs, streamer = text_streamer, max_new_tokens = 1000, use_cache = True)
```

4.4. RAG

Retrieval-Augmented Generation (RAG) is an advanced framework in natural language processing that combines the strengths of retrieval-based and generation-based models. **RAG** retrieves relevant context or knowledge from external sources, such as databases or documents, and integrates this information into the text generation process. This hybrid approach enhances the model's ability to produce accurate, contextually informed, and detailed responses, making it especially useful for tasks like question answering. **RAG** ensures grounded outputs, reducing hallucinations and improving reliability.

4.4.1. Workflow

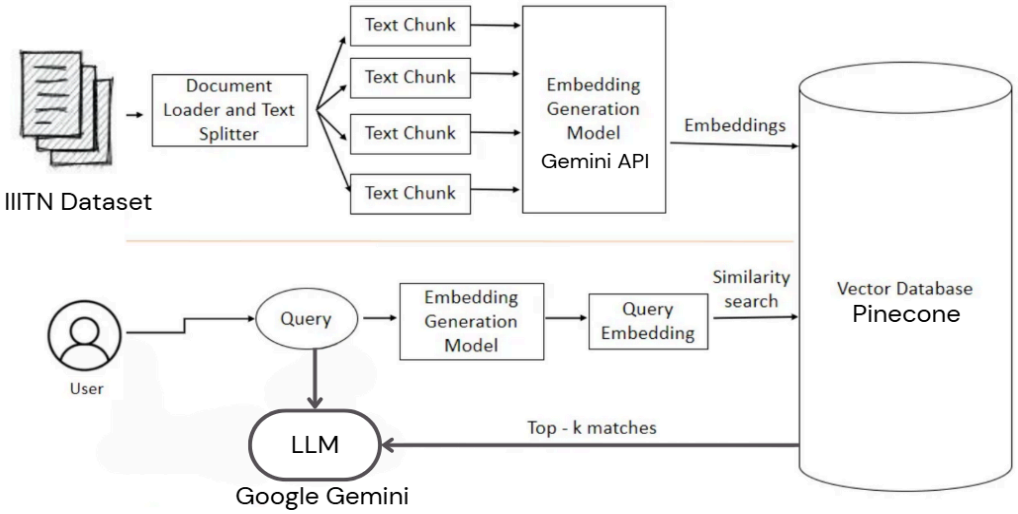

Figure 4.5 : RAG Workflow

4.4.2. Pinecone

We initialise the **Pinecone** client with an API key and sets up an index for storing embeddings. It checks if an index named "iiitn" already exists; if not, it creates one with a specified dimension of 768 and a cosine similarity metric. The index is set to be serverless, running on AWS in the us-east-1 region. This function allows for seamless setup of a Pinecone index for efficient vector search, ensuring that the necessary infrastructure is in place for storing and querying embeddings related to the project.

```
def setup_pinecone():
    pc = Pinecone(api_key=os.getenv("PINECONE_API_KEY"))
    index_name = "iiitn"

    if index_name not in pc.list_indexes().names():
        pc.create_index(
            name=index_name,
            dimension=768,
            metric="cosine",
            spec=ServerlessSpec(cloud='aws', region='us-east-1')
        )
    return index_name, pc
```

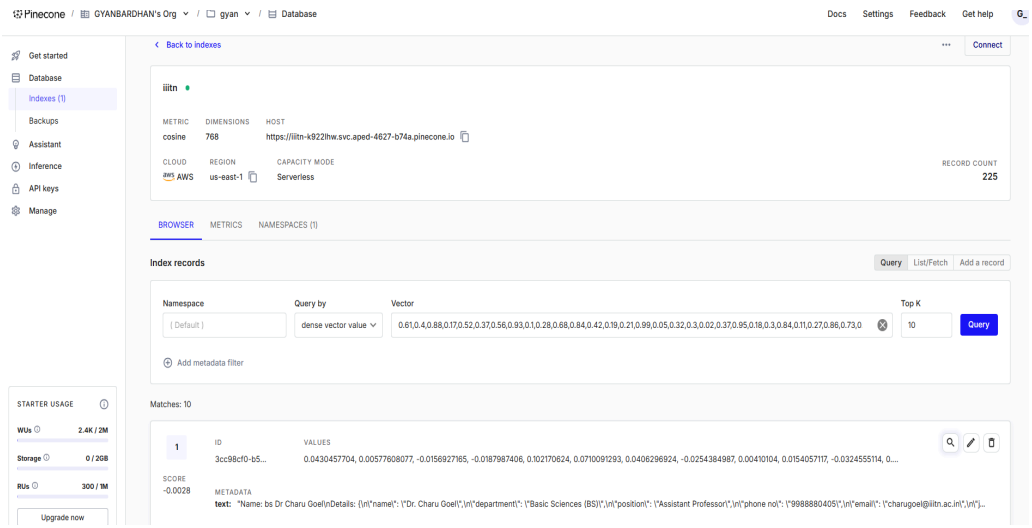

Figure 4.6 : Pinecone Index

4.4.3. Retrieval & Generation

The retrieval and generation pipeline employs Pinecone for fast vector search, as well as Google Generative AI for answering questions posed by users. Retrieval means the process of using embeddings to search for existing vectors in a pre-indexed **Pinecone** index, which contains vectorized documents. When the user puts out a query, it outputs embeddings for the query, which are used to find the most appropriate documents in the Pinecone index.

```
embeddings = GoogleGenerativeAIEmbeddings(model="models/embedding-001")

pineconekey = os.getenv("PINECONE_API_KEY")
pc = Pinecone(api_key=pineconekey)
index_name = "iiitn"

docsearch = PC.from_existing_index(index_name=index_name, embedding=embeddings)

prompt_template = """
Use the following pieces of information to answer the user's question.
If you don't know the answer, just say that you don't know, don't try to make up an answe
Context: {context}
Question: {question}
Only return the helpful answer below and nothing else.
Helpful answer:
"""

PROMPT = PromptTemplate(template=prompt_template, input_variables=["context", "question"]

llm = ChatGoogleGenerativeAI(model="models/gemini-1.5-pro-latest", temperature=0.9)

qa = RetrievalQA.from_chain_type(
    llm=llm,
    chain_type="stuff",
    retriever=docsearch.as_retriever(search_kwargs={'k': 2}),
    return_source_documents=True,
    chain_type_kwargs={"prompt": PROMPT}
)

result = qa({"query": user_input})
```

The output of these embeddings is taken into the generative aspect where the **Google Gemini** model (ChatGoogleGenerativeAI) is utilized in order to produce a response based on the retrieved documents. The RetrievalQA chain ties together the document retriever with the generative model, enabling a pipeline where relevant context is provided to the model for answering questions. The response is polished by a prompt that specifies the structure of the answer, making it clear and relevant while discouraging the model from producing irrelevant or wrong information. The core of the functionality lies in combining retrieval with generation.

In this method, the retriever fetches context from the Pinecone index, and then the generative model creates a response based on the context. This, therefore, enables contextual answering tailored to the user's query but backed by relevant information stored in the index. It can handle situations where explicit knowledge may not be in the model, relying only on the retrieved context to generate informative and correct answers.

4.5. Voice Command Incorporation

The project integrates voice command functionality using **OpenAI's Whisper model** for speech-to-text conversion. Audio input is captured via the audio_recorder_streamlit library and processed by the **Whisper `small.en` model**, which transcribes the speech into text. This transcription is seamlessly used as input for the chatbot, enabling users to interact through voice commands. This feature enhances accessibility and provides a natural, user-friendly alternative to traditional text-based input.

4.6 Web Application

We have deployed the DISHA RAG module on Hugging Face Spaces with a Streamlit UI, featuring voice command support. Hosted on a 16 GB RAM, 2vCPU setup, it ensures smooth performance.

Accessible to everyone at :
https://huggingface.co/spaces/gyanbardhan123/Disha_RAG

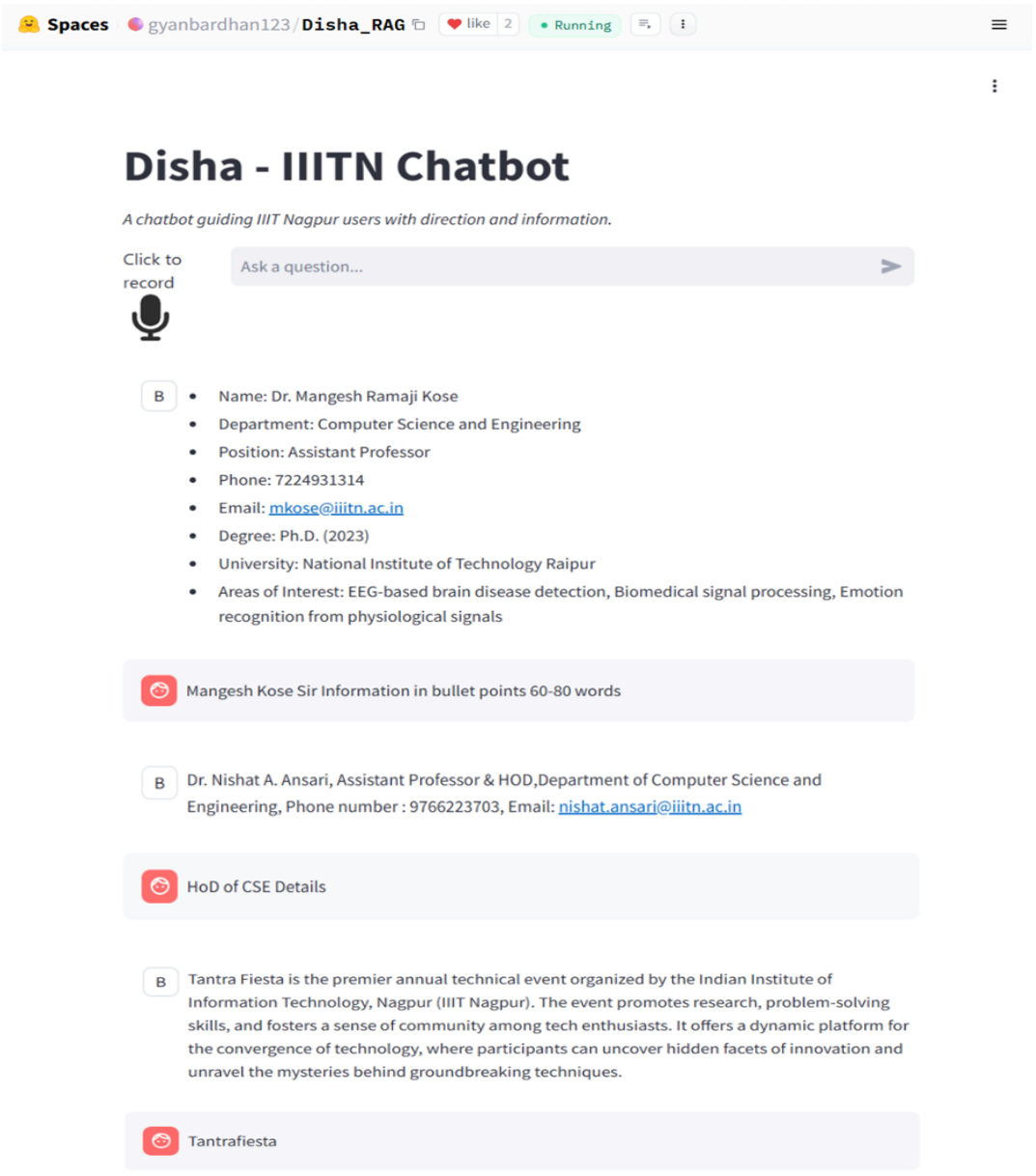

Figure 4.7 : *Disha RAG* Web Interface

4.7. Final Model - Integration

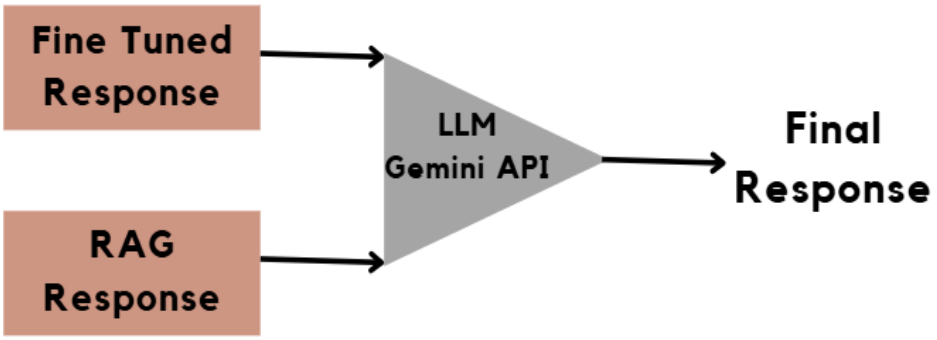

Figure 4.8 : Models Integration

A multi-step approach to generating accurate responses by combining Fine-Tuned and **RAG (Retrieval-Augmented Generation)** responses. Both methods are processed through the Gemini API, which acts as a unifying layer to evaluate and generate the final response. The inclusion of the Gemini API ensures that the strengths of both fine-tuning and **RAG** are utilized effectively, leading to a response that is not only relevant but also contextually accurate.

This approach enhances the chatbot's ability to address diverse user queries. Fine-tuning specializes in providing pre-trained accuracy, while **RAG** enriches the model with real-time and context-specific knowledge. Together, they produce a refined response by leveraging complementary strategies, ensuring a balance of precision and up-to-date relevance.

4.8. Performance Metrics

Performance metrics play a pivotal role in evaluating and refining machine-generated text for applications like machine translation, summarization, and conversational AI systems. These metrics provide a systematic way to measure the quality, relevance, and effectiveness of responses, ensuring that the system meets user expectations and project goals.

4.8.1. BLEU

BLEU (Bilingual Evaluation Understudy) is a widely used metric for evaluating the quality of machine-generated text, particularly in machine translation and natural language generation tasks. **BLEU** compares the machine-generated output (candidate text) with one or more reference texts (ground truth). It measures how much the generated text overlaps with the reference in terms of word sequences, called **n-grams**, while considering precision and penalizing overly short outputs.

Advantages of BLEU for *Disha*

1. **Speed and Efficiency :** Automates evaluation, reducing reliance on human testers.
2. **Consistency :** Provides a standardized metric for quality assessment.
3. **Language Independence :** Works for diverse phrasing and multiple languages, ensuring flexibility.

4.8.2. ROUGE

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a set of metrics used to evaluate the quality of machine-generated text, particularly in tasks like text summarization and natural language generation. Unlike **BLEU**, which focuses on precision, **ROUGE** emphasizes recall by measuring how much of the reference text is captured by the generated text.

Advantages of ROUGE for Disha

1. **Recall-Oriented :** Ensures ***Disha*** captures the critical elements of the reference response.
2. **Flexibility :** Can evaluate overlap at different granularities, from individual words to sequences.
3. **Broad Applicability :** Effective for evaluating summaries, FAQs, or chatbot responses that prioritize completeness.

4.8.3. Semantic Similarity

Semantic Similarity measures the similarity between the meanings of the machine-generated text (candidate) and the reference text (ground truth). Unlike metrics like **BLEU** and **ROUGE**, which rely on matching exact words or sequences, **Semantic Overlap** evaluates how well the generated response captures the underlying meaning, even if phrased differently. This makes it particularly valuable for natural language generation tasks.

Advantages of Semantic Overlap for *Disha*

- **Meaning-Focused :** Ensures ***Disha*'s** responses align with the intended meaning of the reference.
- **Flexibility :** Rewards valid responses that use synonyms or paraphrasing.
- **Context Sensitivity :** Accounts for the relationship between words, enhancing evaluation for complex queries.

Although the words differ slightly, semantic overlap measures the alignment in meaning, assigning a high similarity score for responses that accurately capture the intent and key details.

4.8.4. Human Evaluation

Human Evaluation is the process of assessing the quality of machine generated text by involving human judges. Unlike other automated metrics, human evaluation captures subjective aspects such as clarity, relevance, fluency and appropriateness, making it the gold standard for evaluating ***Disha***.

Advantages of Human Evaluation for *Disha*

1. **Subjective Insights :** Captures nuances like tone and user satisfaction.
2. **Flexibility :** Can adapt to specific evaluation criteria based on our goals.
3. **Context Awareness :** Considers the broader conversational flow and intent of the query.

Chapter 5

Results and Discussions

This section includes elaborate analysis of the conducted comparison of the performance of these models based on the numerous key metrics. These sections include detailed graphical comparisons along with score-based insights about each model. The parameters have been chosen with respect to critical parameters like BLEU, ROUGE-L, Semantic Similarity, and Human Evaluation that gauge linguistics accuracy, context-based relevance, and real-time use. These comparisons cover discussions of the strengths of all these models, with stress placed on their respective best attributes in particular scenarios. Lastly, the best one according to a fair decision in terms of performance efficiency, and computational complexity of this model is selected; thereby, the best fitting solution is chosen.

5.1. Results

Table 5.1 : Model Evaluation based on performance metrics

Model	BLEU	ROUGE-L	Semantic Similarity	Human Evaluation	Trained Parameters
LLaMA-3.2-1b (R=8)	0.925700	0.964550	0.998106	0.934744	12,156,928
LLaMA-3.2-1b (R=16)	0.925950	0.964757	0.998106	0.942012	24,313,856
LLaMA-3.2-1b (R=32)	0.924404	0.963656	0.998096	0.946338	48,627,712
Phi 3.5 Mini	0.785048	0.886750	0.998205	0.852504	29,884,416
RAG	0.964902	0.996087	0.995800	0.967379	0

BLEU Score

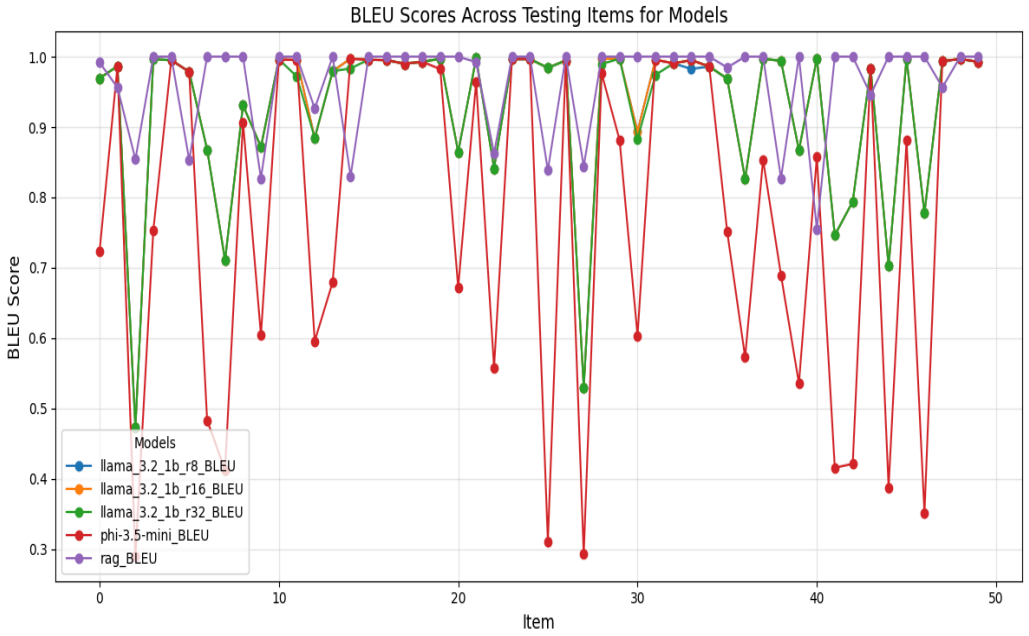

Figure 5.1 : BLEU Score based performance evaluation

ROUGE Score

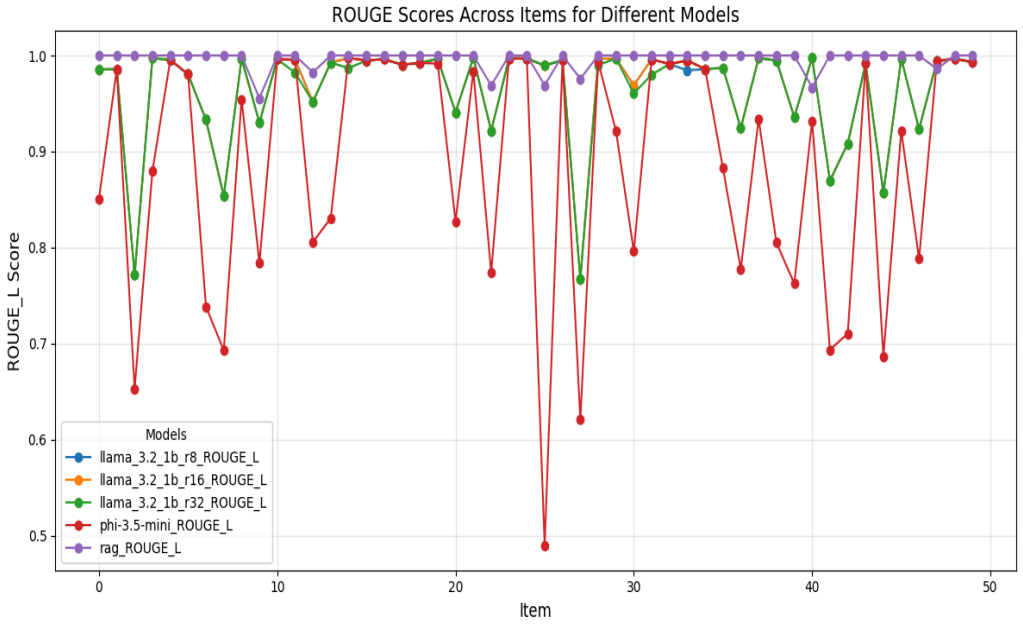

Figure 5.2 : ROUGE Score based performance evaluation

Semantic Overlap

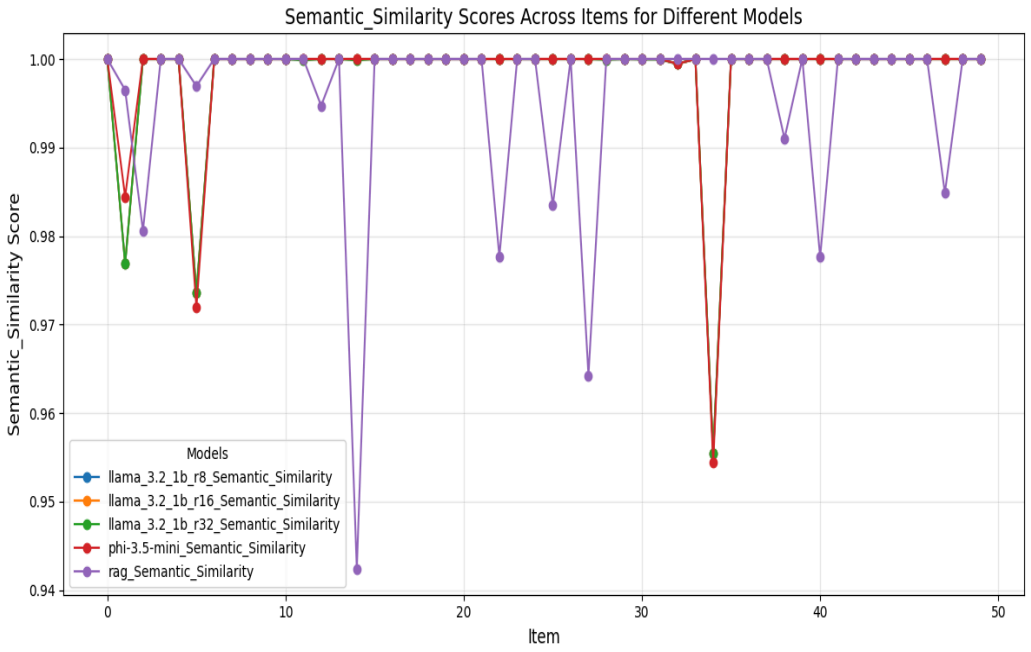

Figure 5.3 : Semantic Overlap based performance evaluation

Human Evaluation

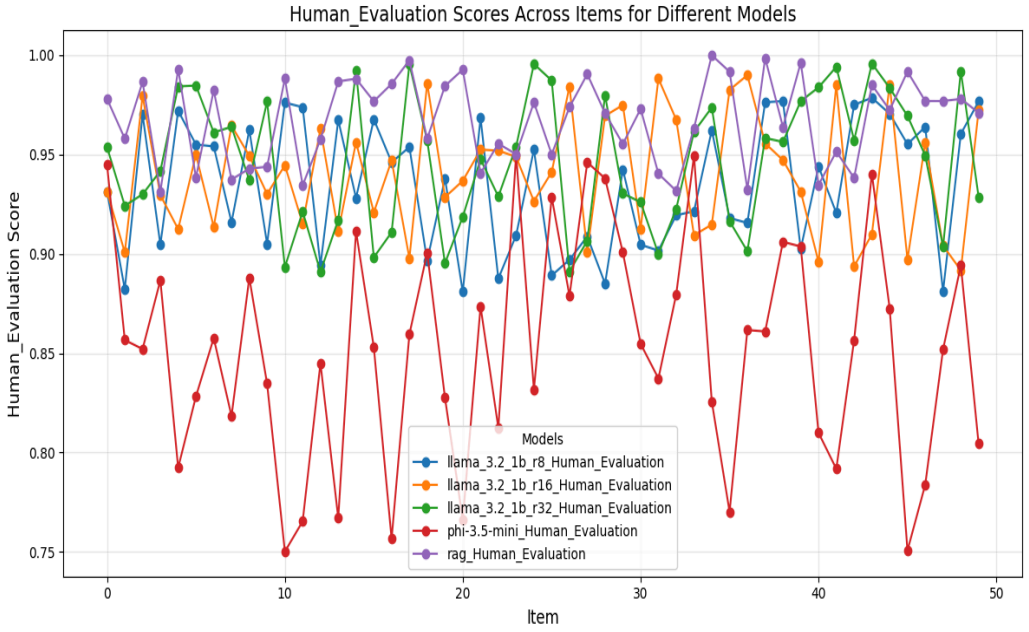

Figure 5.4 : Human Evaluation Score based performance evaluation

Trainable Parameters

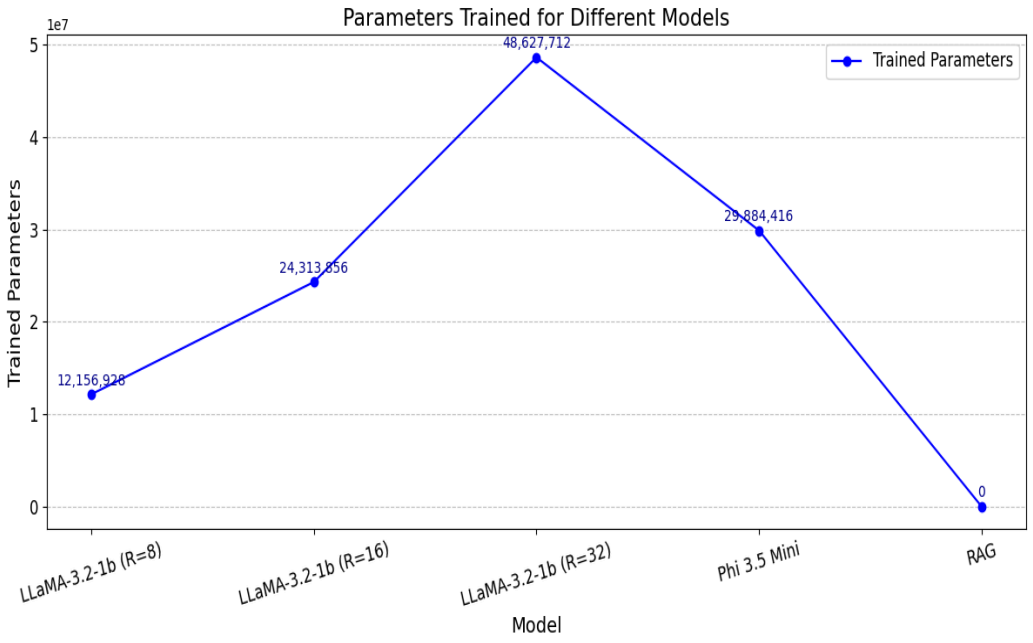

Figure 5.5 : Trained Parameters based performance evaluation

In the evaluation of various LLM models (i.e LLaMA-3.2-1b (R=8), LLaMA-3.2-1b (R=16), LLaMA-3.2-1b (R=32), Phi 3.5 Mini), see Table 5.1, **LLaMA-3.2-1b (R=16)** was selected as the optimal choice due to its superior balance between performance and efficiency. The model demonstrated **high BLEU (0.925950) and ROUGE-L (0.964757) scores** from Figure 5.1 Figure 5.2, indicating strong lexical and structural accuracy in text generation, alongside near-perfect **semantic similarity (0.998106)** in Figure 5.3 Its **human evaluation score (0.942012)** in Figure 5.4 reflects excellent alignment with user preferences, closely rivaling the more resource-intensive LLaMA-3.2-1b (R=32). With **24,313,856 trained parameters** in Figure 5.5, LLaMA-3.2-1b (R=16) achieves this high level of performance while maintaining a significantly lower computational cost compared to LLaMA-3.2-1b (R=32). Moreover, it outperforms Phi 3.5 Mini across all critical metrics, including BLEU, ROUGE-L, and human evaluation. This combination of strong performance and resource efficiency makes LLaMA-3.2-1b (R=16) the ideal choice for deployment in real-world scenarios.

RAG displays excellent performance in all significant metrics, hence making the model a strong and viable solution. The **BLEU score is 0.964902**, in Figure 5.1 which indicates syntactical correctness and fluency, but the **ROUGE-L score of 0.996087** in Figure 5.2 emphasizes contextual meaning and structure preservation by the model. The model has a **Semantic Similarity score of 0.995800**, in Figure 5.3 demonstrating the ability to produce output that closely corresponds to a human interpretation. This is supported by the high **Human Evaluation score of 0.967379** in Figure 5.4 for its utility and quality in application. RAG is quite efficient since it uses no training parameters; rather, its workings depend on the retrieval mechanism. It lowers the costs of computations with excellent results and hence a scalable option for multiple applications.

RAG outperformed LLaMA-3.2-1b (R=16) in all evaluation metrics, thus demonstrating its supremacy in different aspects. But a problem with RAG is that it relies on **cosine similarity for information retrieval**. This method can cause inaccuracies when questions are rephrased because it cannot understand the subtle context. Consequently, RAG may give shallow responses or not fully answer questions, especially when the phrasing or context is different from what it has encountered during retrieval.

To address the limitations of RAG and enhance contextual understanding, we have merged both LLaMA-3.2-1b (R=16) and RAG models. The purpose of **combining LLaMA-3.2-1b (R=16) with RAG** is to harness the strength of both models for enhanced performance. **LLaMA--3.2-1b (R=16)‘s attention mechanism** helps to understand context better, leading to the generation of more accurate responses. RAG is stronger at retrieving the right information . This hybrid overcomes the shortcomings of RAG.

We first get the outputs from both **RAG and LLaMA** and then feed these outputs to Gemini for further processing. Gemini synthesizes and generates an accurate, context-aware, and summarized response, ensuring more relevant and precise answers. This hybrid approach enhances both retrieval accuracy and contextual comprehension.

5.2. Future Improvements and Plans

To enhance ***Disha*** and make it even more user-friendly and efficient, we have outlined the following future plans:

1. **Multilingual Support:**
   - As a step toward inclusivity, we plan to add Hindi as a supported language, enabling ***Disha*** to cater to a more diverse user base.
   - In the future, additional regional languages can also be incorporated based on user needs.
2. **Optimization for Lightweight Use:**
   - ***Disha*** will be optimized to work efficiently on the college website with minimal hardware and financial requirements.
   - This involves reducing the computational footprint and utilizing lightweight models or APIs to ensure smooth performance even on resource-constrained systems.

3. **Image Output Support:**
   - We will integrate the facility for image output, such as display faculty pictures and other related images, to make the experience better and more visual and interactive.

4. **Learning and Adaptation in Real-time:**
   - Disha will thus be improved to operate real-time, thereby no further fine-tuning will be necessary when new data arrives as Disha will continue to learn based on the new information that continuously comes.

Vision for the Future

Disha will support multiple languages, integrate image outputs, optimize the usage of resources, and enable real-time adaptation for being an inclusive and sustainable solution. These improvements will enhance user experience, making Disha a reliable tool for students, faculty, and visitors while ensuring continuous improvement without frequent fine-tuning.

Chapter 6

Conclusion

In today's fast-paced digital world, navigating complex systems to find the right information can be overwhelming. ***Disha***, our conversational chatbot, was developed to address this challenge on our college website, offering a seamless and user-friendly solution for students, parents, and visitors alike. By leveraging cutting-edge technologies like Machine Learning, Natural Language Processing, and Large Language Models, ***Disha*** ensures accurate and efficient responses to queries, revolutionizing how users access information.

Throughout this project, we employed both automated metrics like BLEU, ROUGE, and Semantic Overlap, as well as Human Evaluation, to rigorously assess and refine the chatbot's performance. Each evaluation method provided unique insights, helping us improve ***Disha*'s** ability to deliver precise, meaningful, and context-aware answers.

Looking ahead, ***Disha*'s** planned enhancements, including multilingual support and resource optimization, will further broaden its accessibility and usability. These improvements will not only make ***Disha*** more versatile but also ensure that it remains a sustainable and inclusive solution for diverse users.

With ***Disha***, we have taken a significant step toward simplifying information access, fostering a more connected and user-focused digital environment. As it continues to evolve, ***Disha*** promises to be an indispensable tool, transforming the way users interact with our college's digital ecosystem.

REFERENCES

[1] Basilico, Michele. *Design, Implementation and Evaluation of a Chatbot for Accounting Firm: A Fine-Tuning Approach With Two Novel Datasets*. Master's thesis, Politecnico di Torino, Corso di laurea magistrale in Ingegneria Informatica, 2024. Advisor: Luca Ardito.

[2] Sepulveda, J. B., E. Potes, H. N. Pineda Montoya, S. Rodriguez, F. I. Orduy, J. E. Rosales Cabezas, A. Traslaviña Navarrete, D. Madrid Farfan, and S. "Towards Enhanced RAC Accessibility: Leveraging Datasets and LLMs," arXiv e-prints, Vol. 2405, 08792, 2024, doi:10.48550/arXiv.2405.08792.

[3] Kulkarni, Mandar, Praveen Tangarajan, Kyung Kim, and Anusua Trivedi. "Reinforcement Learning for Optimizing RAG for Domain Chatbots." *arXiv*, 2024, https://doi.org/10.48550/arXiv.2401.06800.

[4] Neupane, S., Hossain, E., Keith, J., Tripathi, H., Ghiasi, F., Amiri Golilarz, N., Amirlatifi, A., Mittal, S., & Rahimi, S. (2024). *From Questions to Insightful Answers: Building an Informed Chatbot for University Resources*. arXiv. https://doi.org/10.48550/arXiv.2405.08120

[5] Li, Y., Lin, X., Wang, W., Feng, F., Pang, L., Li, W., Nie, L., He, X., & Chua, T.-S. (2024). A Survey of Generative Search and Recommendation in the Era of Large Language Models. *arXiv preprint arXiv:2404.14102*.

[6] Das, S., "Fine Tune Large Language Model (LLM) on a Custom Dataset with QLoRA," *Medium*, Jan. 25, 2024.

[7] Llama 3.2, "Revolutionizing Edge AI and Vision with Open, Customizable Models," *Meta Blog*, Sept. 25, 2024.

[8] Tripathi, R., “What are Vector Embeddings,” *Pinecone Blog*, Jun. 30, 2023.

[9] Proser, Z.,“Retrieval Augmented Generation (RAG),” *Pinecone Blog*, Aug. 3, 2023.

[10] Schwaber-Cohen, R., “What is a Vector Database & How Does it Work? Use Cases + Examples,” *Pinecone Blog*, May 3, 2023.

[11] Tran, H., “Which is Better, Retrieval Augmentation (RAG) or Fine-Tuning? Both,” *Applied AI, Research*, Sept. 20, 2023, updated Oct. 11, 2024.

[12] Mohandas, G., and P. Moritz, “Building RAG-based LLM Applications for Production,” *Anyscale*, Oct. 25, 2023.

[13] Zhang, B., Liu, Z., Cherry, C., & Firat, O. (2024). When Scaling Meets LLM Finetuning: The Effect of Data, Model and Finetuning Method. *arXiv*. https://doi.org/10.48550/arXiv.2402.17193

[14] Vidivelli, S., Ramachandran, M., Dharunbalaji, A. (2024). Efficiency-driven custom chatbot development: unleashing langchain, RAG, and performance-optimized LLM fusion. *Computers, Materials & Continua*, *80(2)*, 2423-2442. https://doi.org/10.32604/cmc.2024.054360

[15] H. K. Chaubey, G. Tripathi, R. Ranjan and S. k. Gopalaiyengar, "Comparative Analysis of RAG, Fine-Tuning, and Prompt Engineering in Chatbot Development," 2024 International Conference on Future Technologies for Smart Society (ICFTSS), Kuala Lumpur, Malaysia, 2024, pp. 169-172, doi: 10.1109/ICFTSS61109.2024.10691338.