A REPORT
ON

Sentiment Analysis of Incoming Calls on Helpdesk

A PROJECT REPORT

Submitted by,

DEEPIKA C.S	**-20211COM0075**
PRAKASH SINGH	**-20211CEI0055**
DISHIK L SETTY	**-20211COM0006**
PAAVANA GOWDA	**-20211COM0029**

Under the guidance of,

Ms. B H IMPA

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER ENGINEERING

At

GAIN MORE KNOWLEDGE
REACH GREATER HEIGHTS

PRESIDENCY UNIVERSITY

BENGALURU

MAY 2025

PRESIDENCY UNIVERSITY

SCHOOL OF COMPUTER SCIENCE ENGINEERING

CERTIFICATE

This is to certify that the Project report **"Sentiment Analysis of Incoming Calls on Helpdesk"** being submitted by "DEEPIKA C. S, PRAKASH SINGH, DISHIK L SETTY, PAAVANA GOWDA" bearing roll number(s) "20211COM0075, 20211CEI0055, 20211COM0006, 20211COM0029" in partial fulfilment of requirement for the award of degree of Bachelor of Technology in Computer Engineering is a bonafide work carried out under my supervision.

Ms. B H IMPA
Assistant Professor
PSCS
Presidency University

Dr. GOPAL KRISHNA SHYAM
Professor & HOD
PSCS
Presidency University

Dr. MYDHILI K NAIR
Associate Dean
PSCS
Presidency University

Dr. MD. SAMEERUDDIN KHAN
Pro Vice Chancellor Engineering
Dean -PSCS/PSIS
Presidency University

PRESIDENCY UNIVERSITY

SCHOOL OF COMPUTER SCIENCE ENGINEERING

DECLARATION

We hereby declare that the work, which is being presented in the project report entitled **“Sentiment Analysis of Incoming Calls on Helpdesk”** in partial fulfilment for the award of Degree of **Bachelor of Technology in Computer Engineering**, is a record of our own investigations carried under the guidance of **Ms. B H Impa, Assistant Professor, School of Computer Science and Engineering , Presidency University, Bengaluru**.

We have not submitted the matter presented in this report anywhere for the award of any other Degree.

ROLL NUMBERS	NAMES	SIGNATURE
DEEPIKA C. S	**20211COM0075**	
DISHIK L SETTY	**20211COM0006**	
PAAVANA GOWDA	**20211COM0029**	
PRAKASH SINGH	**20211CEI0055**	

ABSTRACT

The proposed project focuses on the development of an advanced system that integrates speech recognition, sentiment analysis, and sarcasm detection, providing a comprehensive solution for real-time textual content analysis. The primary aim of this system is to transcribe spoken language into text and analyze the emotional tone of the text while identifying sarcastic statements. The system is built using a combination of cutting-edge technologies, including automatic speech recognition (ASR) models such as Whisper, natural language processing (NLP) techniques for sentiment analysis, and advanced algorithms for sarcasm detection. The backend of the system leverages FastAPI, ensuring efficient handling of API requests, while MongoDB is used for storing user data and transcriptions. The system's core functionality enables users to input audio through an intuitive interface, which is then converted into text and analyzed for sentiment and sarcasm. The results are displayed in real-time, offering valuable insights into the emotional tone of the content. This project not only demonstrates the potential of integrating multiple AI-based technologies but also aims to provide a tool that can be utilized in a variety of applications such as customer feedback analysis, social media monitoring, and content moderation. Through the implementation of robust machine learning models and scalable backend infrastructure, this system offers a high-performance solution for understanding and interpreting human emotions in both speech and text.

ACKNOWLEDGEMENT

First of all, we indebted to the **GOD ALMIGHTY** for giving me an opportunity to excel in our efforts to complete this project on time.

We express our sincere thanks to our respected Dean **Dr. Md. Sameeruddin Khan,** Pro-VC - Engineering and Dean, Presidency School of Computer Science and Engineering & Presidency School Of Information Science, Presidency University for getting us permission to undergo the project.

We record our heartfelt gratitude to our beloved Associate Deans **Dr. Mydhili K Nair,** Presidency School of Computer Science Engineering &, Presidency University, and **Dr. Gopal Krishna Shyam**, Head of the Department, Presidency School of Computer Science and Engineering, Presidency University, for rendering timely help in completing this project successfully.

We are greatly indebted to our guide **Ms. B H Impa,** Assistant Professor, Presidency School of Computer Science and Engineering, Presidency University for her inspirational guidance, and valuable suggestions and for providing us a chance to express our technical capabilities in every respect for the completion of the project work. We would like to convey our gratitude and heartfelt thanks to the CSE7301 University Project Coordinators **Mr. Md Ziaur Rahman and Dr. Sampath A K**, department Project Coordinators **Dr. Sudha P** and Git hub coordinator **Mr. Muthuraj.**

We thank our family and friends for the strong support and inspiration they have provided us in bringing out this project.

DEEPIKA C S(20211COM0075)
PRAKASH SINGH(20211CEI0055)
DISHIK L SETTY(20211COM0006)
PAAVANA GOWDA(20211COM0029)

LIST OF TABLES

Sl. No.	Table Name	Table Caption	Page No.
1	Table 1.1	Scheduled task	16

LIST OF FIGURES

Sl. No.	Figure Name	Caption	Page No
1	Fig 3.1	**Sentiment analysis detection**	**7**
2	Fig 3.2	**Sentiment Analysis Audio Conversations using Text Classification**	**8**
3	Fig 3.3	**Incoming calls on Helpdesk**	**9**
4	Fig 6.1	**Sentiment Analysis System Architecture**	**14**
5	Fig 7.1	**Timeline**	**15**
6	Fig 9.1	**Output #1**	**20**
7	Fig 9.2	**Output #2**	**21**
8	Fig 9.3	**Output #3**	**21**
9	Fig 9.4	**Output #4**	**22**
10	Fig 9.5	**Output #5**	**22**
11	Fig 9.6	**Output #6**	**23**
12	Fig 9.7	**Output #7**	**23**
13	Fig 9.8	**Output #8**	**24**

TABLE OF CONTENTS

Chapter No	Title	Page No
1.	INTRODUCTION	2
2.	LITERATURE REVIEW	3-5
3.	RESEARCH GAPS OF EXISTING METHODS	6-10
4.	PROPOSED METHODOLOGY	11
5.	OBJECTIVES	12
6.	SYSTEM DESIGN & IMPLEMENTATION	13– 14
	TIMELINE FOR EXECUTION OF PROJECT	
7.	7.1 Timeline of project	15-16
	7.2 Gantt chart	
8.	OUTCOMES	17-18
9.	RESULTS AND DISCUSSIONS	19-24
10.	CONCLUSION	25
11.	REFERENCES	26-27
12.	APPENDIX-A (PSUEDOCODE)	28-41
13.	APPENDIX-B (SCREENSHOTS)	42-52
14.	APPENDIX-C (ENCLOSURES)	53
	SUSTAINABLE DEVELOPMENT GOALS	54

CHAPTER-1

INTRODUCTION

In today's digital era, handling audio data efficiently has become critical across multiple industries, from media production to customer service analytics. The application developed here is a **comprehensive audio processing backend** built using **Flask**, integrating powerful AI models for **audio denoising**, **speech-to- text transcription**, **sentiment analysis**, and **sarcasm detection**.

At its core, the system is designed to accept audio files in formats like WAV, MP3, FLAC, and M4A, process them intelligently, and return enriched insights to the user. It leverages the **Faster-Whisper** model for fast and accurate transcription across several languages, including English, Hindi, Tamil, Kannada, Telugu, and Malayalam. Additionally, **DeepFilterNet** is integrated for advanced audio denoising, improving the clarity of recordings, especially in noisy environments.

Beyond basic transcription, the application dives deeper into understanding text meaning by performing **sentiment analysis** and **sarcasm detection**. This enables not just knowing *what* was said, but also *how* it was expressed — a valuable asset for industries like customer support, content moderation, and social media analysis. Sarcasm detection is powered by a fine-tuned transformer model sourced from Hugging Face, ensuring nuanced interpretation of text tone.

To manage data effectively, the system uses **MongoDB** along with **GridFS** to store audio files and their corresponding transcriptions, ensuring scalability and efficient retrieval. CORS support is enabled, making the API accessible from various client applications without cross-origin issues.All modules are wrapped with robust logging and error handling, ensuring that every action is monitored and that any issues during audio processing or model inference are clearly reported.

In essence, this backend is not just an audio-to-text tool—it is a full-fledged **intelligent audio analysis platform** that processes, enhances, transcribes, and interprets human speech with a blend of **deep learning**, **natural language processing**, and **cloud-ready database management**.

CHAPTER-2

LITERATURE SURVEY

The integration of Artificial Intelligence (AI) and Machine Learning (ML) technologies in call center operations has become increasingly significant to improve customer service, enhance agent performance evaluations, and identify customer emotions. Several researchers have proposed advanced systems aimed at automating and optimizing various aspects of call center management. The following literature explores these developments in detail.

Betül Karakus and Galip Aydin [1] proposed a distributed call center monitoring system that leverages Big Data analytics to evaluate the performance of customer service agents. Their system utilizes the Hadoop MapReduce framework to process large volumes of call records efficiently. They adopted text similarity algorithms such as Cosine similarity and n-gram models to evaluate conversation quality based on various performance metrics, including greeting protocols, problem-solving efficiency, and use of polite language. A notable aspect of their system is the inclusion of slang word detection to refine performance evaluation. Although the system effectively addresses the challenge of analyzing massive call data automatically, it predominantly focuses on textual data, thereby neglecting deeper emotional aspects of the conversation which are often crucial in customer relationship management.

Souraya Ezzat, Neamat El Gayar, and Moustafa M. Ghanem [2] focused on sentiment analysis of audio conversations by transcribing speech into text and then applying classification algorithms. They explored several machine learning models, such as Support Vector Machine (SVM), Decision Trees, Naive Bayes,and K-Nearest Neighbors (KNN), for classifying customer sentiments into positive or negative categories. Furthermore, they investigated clustering methods like K-Means and Hierarchical Clustering to automatically group similar calls based on sentiment characteristics. Their methodology also emphasized the importance of feature selection using techniques like TF-IDF and Chi-square to improve classification accuracy. However, one limitation of their study is that the dataset used was synthetically generated, limiting the model’s exposure to the complexities and noise typically found in real-world call center conversations, such as sarcasm, informal language, or background interruptions.

Xue-Yong Fu, Cheng Chen, Md Tahmid Rahman Laskar, Shayna Gardiner, Pooja

Hiranandani, and Shashi Bhushan TN [3] introduced a system for Entity- Level Sentiment Analysis (ELSA) focused on telephone conversation transcripts from contact centers. Their approach involved the use of a modified DistilBERT model to identify and classify sentiments associated with specific named entities like products or organizations mentioned in customer conversations. In addition to the transformer-based model, they implemented a Convolutional Neural Network (CNN) model supplemented with heuristic rules for opinion word extraction. The researchers also applied transfer learning from the Stanford Sentiment Treebank to enhance their model's performance. Their work addresses the growing need for fine-grained sentiment analysis at the entity level, offering insights valuable for competitive analysis and product improvement. Nevertheless, the system's effectiveness is hampered by the noisy nature of telephone transcripts and the limitations imposed by ASR inaccuracies and conversational disfluencies.

Rohit Raj Sehgal, Shubham Agarwal, and Gaurav Raj [4] focused on improving Interactive Voice Response (IVR) systems by integrating sentiment analysis with Automatic Speech Recognition (ASR) systems. Their study moves away from traditional Dual Tone Multi-Frequency (DTMF) systems and proposes using voice inputs processed through ASR technologies like Amazon Lex. Sentiment analysis is performed using classifiers such as Naive Bayes, Maximum Entropy, and Boosted Trees to determine whether a caller is satisfied or dissatisfied with the service. This allows dynamic handling of calls by escalating dissatisfied customers to human agents. Although the system represents an important step towards more human-like automated service, its focus remains largely on binary sentiment classification. The model does not account for nuanced emotional states like frustration, urgency, or confusion, which could be critical in handling complex customer interactions.

Finally, Yanan Jia and Sony SungChu [5] developed a deep learning framework for sentiment analysis in real-world service calls by combining acoustic and linguistic features. They constructed a pipeline that includes speaker diarization, voice activity detection, and speech-to-text transcription, followed by semi- supervised learning for annotating large volumes of real customer service conversations. Their model identifies sentiment at the sentence level and is particularly focused on detecting negative sentiments, especially anger, which are crucial for understanding service dissatisfaction. The system integrates multimodal data by fusing both acoustic and textual inputs to improve sentiment detection accuracy. One of their major contributions is the application of a semi- supervised learning approach to reduce the labor involved in data labeling. However, despite the robust design, the system still struggles with dynamic sentiment tracking over long conversations and remains vulnerable to the inaccuracies of automatic speech recognition, especially in noisy environments.

Collectively, these studies show significant advancements in automating sentiment detection, performance evaluation, and customer experience analysis within call centers. However, several challenges persist across all approaches, including handling noisy, real- world conversational data, capturing the subtleties of complex human emotions, tracking dynamic changes in sentiment throughout interactions, and reducing dependency on extensive manual data labeling. Future work must address these gaps by designing more resilient, context- aware, and dynamically adaptive systems that can operate effectively under the variable conditions of real-world customer service environments.

CHAPTER-3

RESEARCH GAPS OF EXISTING METHODS

Limited Coverage and Random Sampling — Betül Karakus and Galip Aydin [1]

Although Karakus and Aydin proposed an efficient Big Data-based framework using Hadoop MapReduce for analyzing vast call center datasets, their approach still inherits a fundamental limitation from traditional call center evaluation practices — the random sampling of conversations. Due to resource and time constraints, managers historically sampled only a small portion of recorded calls for manual evaluation. The proposed system automates this process but focuses primarily on textual features such as word similarity, frequency of greetings, and use of polite expressions. It does not capture deeper emotional or contextual elements, such as the tone of voice, hesitation, sarcasm, passive aggression, or frustration that often underlies customer-agent interactions. As a result, even with the use of Big Data tools, the evaluation remains surface-level, risking inaccurate agent assessments and overlooking serious customer satisfaction issues. The absence of multimodal sentiment understanding (like voice emotion or speech tempo) leaves a critical gap in truly holistic call center quality assurance.

Challenges in Emotion Detection from Text — Souraya Ezzat, Neamat El Gayar, Moustafa M. Ghanem [2]

Ezzat and her team focused on transcribing audio conversations to text and then applying traditional machine learning methods like SVM, Naive Bayes, and Decision Trees for sentiment classification. However, their system assumes that the transcription output sufficiently retains emotional cues, which is often not the case in real-world conversational speech. Spoken language contains fillers, interruptions, informal slang, local dialects, and sarcastic remarks that are difficult to interpret accurately after conversion into plain text. Furthermore, the speech-to-text system itself introduces errors due to background noise, overlapping speech, or poor audio quality. Another major limitation is their reliance on synthetically created datasets where actors simulate conversations, thus failing to capture the real complexity, spontaneity, and unpredictability of authentic call center interactions. Consequently, the trained models may perform well on controlled data but are likely to falter in real-world applications where ambiguity and emotional subtleties dominate.

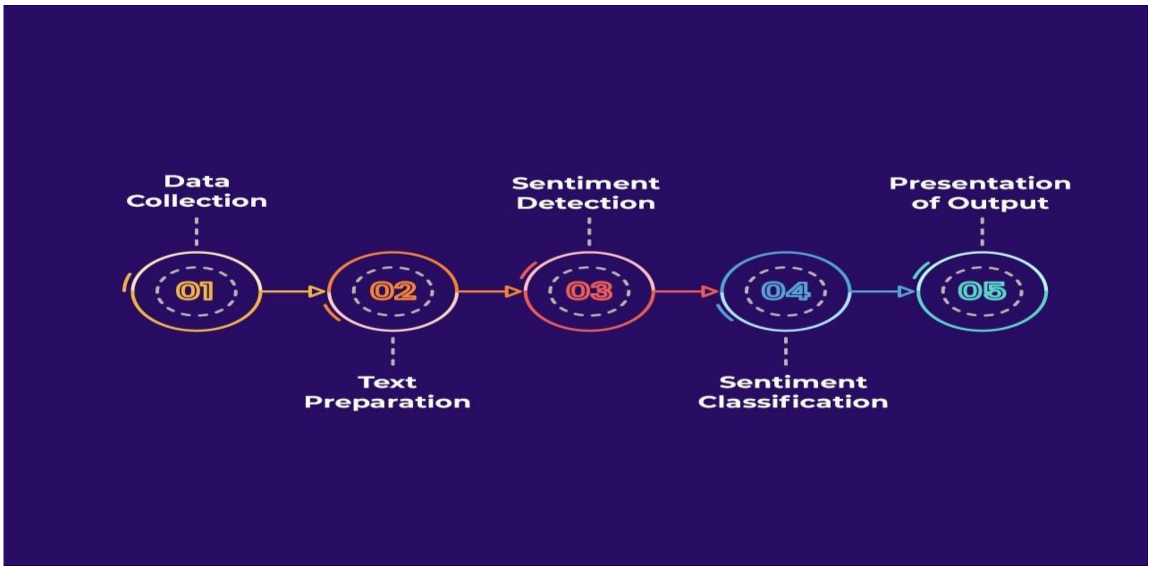


Fig 3.1: Sentiment analysis detection

Lack of Robustness to Noisy and Incomplete Data — Xue-Yong Fu, Cheng Chen, Md Tahmid Rahman Laskar, Shayna Gardiner, Pooja Hiranandani, and Shashi Bhushan TN [3]

Fu and his collaborators tackled Entity-Level Sentiment Analysis (ELSA) using advanced models like DistilBERT and CNNs, aiming to capture sentiments directed at named entities within telephone conversations. However, their solution is heavily dependent on the accuracy of automatic speech recognition (ASR) and named entity recognition (NER) processes. Real-world call center conversations, especially those transcribed via ASR systems, frequently suffer from noise, misrecognized words, incomplete sentences, and conversational disfluencies (e.g., "uh", "you know", "like"). Such errors in transcription severely impact the reliability of downstream sentiment analysis. Even with fine-tuned deep learning models, the lack of robustness against these noisy inputs results in misclassification or failure to capture the correct sentiment polarity. Moreover, entity boundary detection becomes unreliable in long conversations, especially when customers refer to products or services in an indirect or colloquial manner.

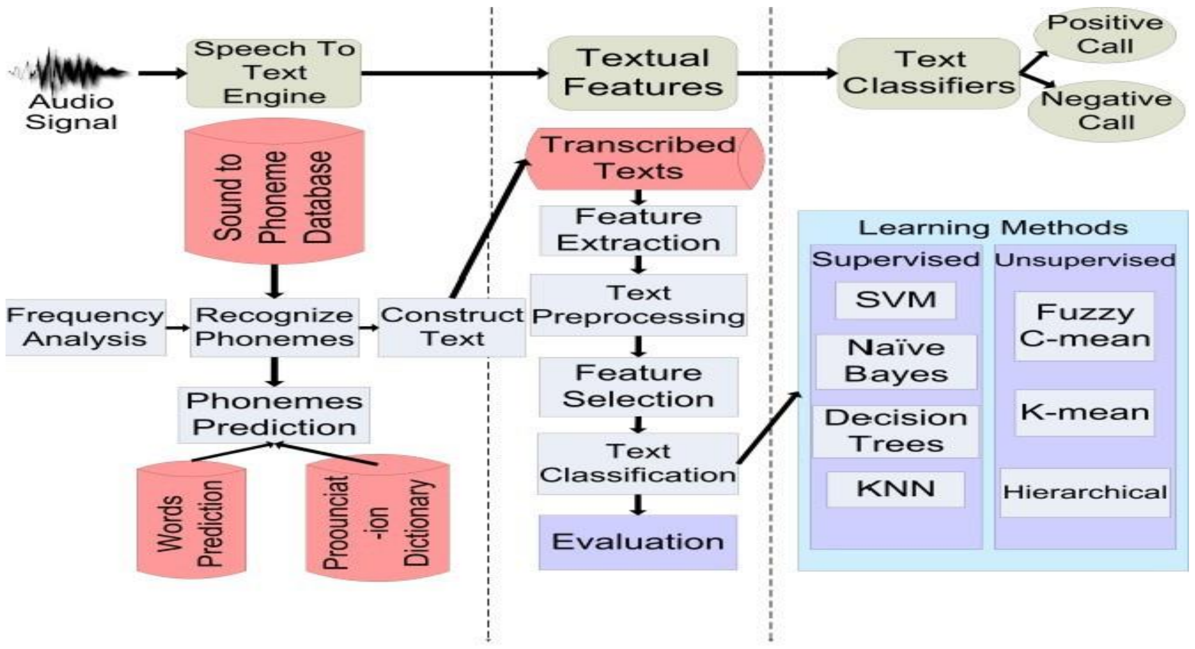


Fig 3.2: Sentiment Analysis of Call Centre Audio Conversations using Text Classification

Dependence on Limited Emotion Categories — Rohit Raj Sehgal, Shubham Agarwal, and Gaurav Raj [4]

Sehgal and his team advanced the field by incorporating sentiment analysis into ASR- powered IVR systems to dynamically adapt responses based on customer emotions. However, their classification approach is restricted to a basic binary outcome: whether the customer is satisfied or dissatisfied. In reality, customer emotions span a wide spectrum including anger, anxiety, confusion, disappointment, hopefulness, urgency, and sarcasm, each requiring a different handling strategy. By simplifying emotions into two categories, the system risks missing critical escalation opportunities or providing inappropriate automated responses. Furthermore, the classifiers used — Naive Bayes, Maximum Entropy, and Boosted Trees — are relatively shallow compared to modern deep learning architectures, which limits the model’s adaptability to evolving language trends, slang, or new product terminologies without frequent and costly retraining cycles. This makes the system rigid and potentially obsolete in dynamic business environments.

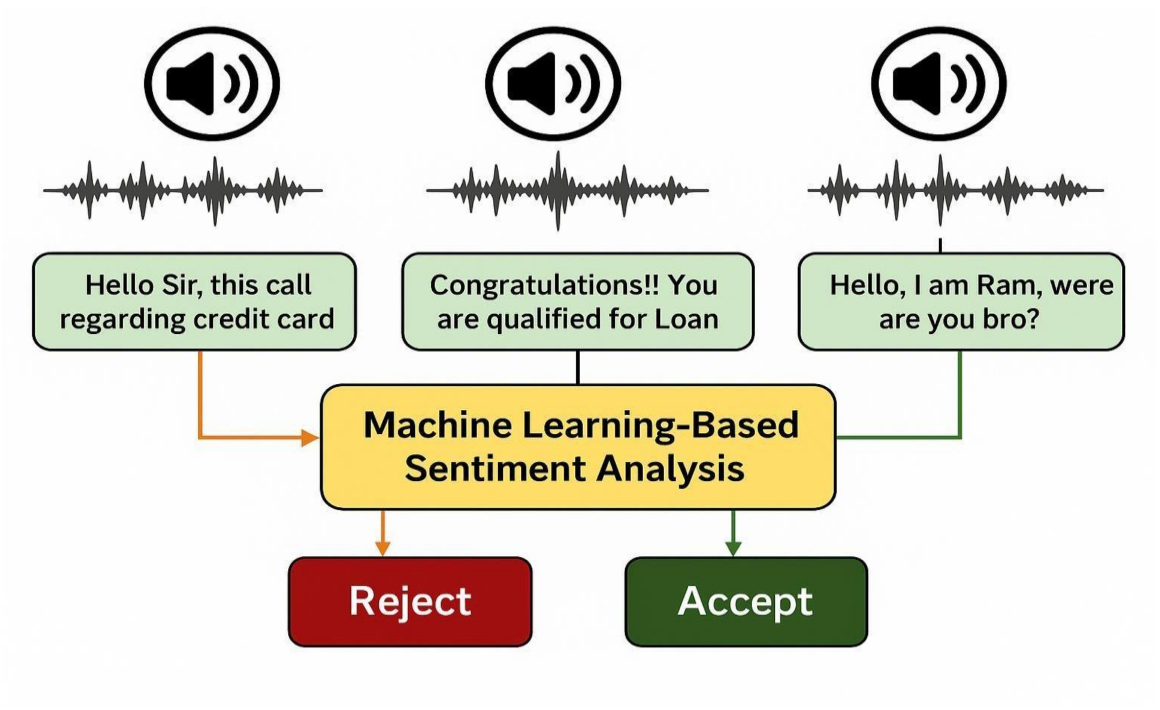

Hello Sir, this call regarding credit card
Congratulations!! You are qualified for Loan
Hello, I am Ram, were are you bro?
Machine Learning-Based Sentiment Analysis
Reject
Accept

Fig 3.3: Incoming calls on Helpdesk

Lack of Fine-Grained Temporal Sentiment Tracking — Yanan Jia and Sony SungChu [5]

Jia and SungChu addressed multimodal sentiment analysis by combining acoustic and linguistic features in real-world service calls, presenting one of the more advanced frameworks. However, their model focuses on sentence-level sentiment analysis, which treats each conversational unit (sentence or utterance) independently. In real-world conversations, especially long service calls, customer emotions are highly dynamic — they can escalate from curiosity to frustration, or from anger to satisfaction depending on the agent's responses. Without tracking these evolving emotional states across the timeline of a call (temporal modeling), critical transitional moments where customer satisfaction shifts (positively or negatively) are missed. Moreover, their system, though rich in fusion techniques, does not fully capitalize on sequential modeling approaches like Long Short-Term Memory (LSTM) or Transformer-based temporal models that could better capture sentiment evolution throughout the conversation. Thus, although strong at static detection, the model lacks the sensitivity required for dynamic, real-time customer sentiment management.

Overall Observations

Despite impressive advances, existing systems still exhibit notable weaknesses such as insufficient modeling of emotional nuances, vulnerability to noisy real- world data, oversimplification of customer emotions, and inadequate tracking of sentiment changes during conversations. These gaps highlight the urgent need for future research to move towards:

- Noise-resilient and context-aware multimodal models
- Fine-grained, dynamic sentiment evolution tracking
- Expansion of emotional categories for more nuanced customer service
- Reducing dependency on perfectly clean transcripts.

CHAPTER-4

PROPOSED METHODOLOGY

The proposed system offers a complete backend service for analyzing audio inputs through a structured and modular pipeline. Initially, the system accepts an audio file uploaded via a RESTful API built using Flask. It performs validation to ensure the audio file is in a supported format such as .wav, .mp3, .flac, or
.m4a. Upon successful validation, the audio file is temporarily stored on the server for processing. The next stage involves improving the quality of the audio using DeepFilterNet, a deep learning-based denoising model that removes background noise while preserving the essential features of the speaker's voice. This denoising step is crucial for enhancing the performance of subsequent processing tasks.

After cleaning the audio, the system employs Faster-Whisper, a lightweight and highly efficient automatic speech recognition model, to transcribe the audio content into text. The model is capable of detecting and handling multiple languages including English, Hindi, Tamil, Kannada, Telugu, and Malayalam, ensuring versatility across different user inputs. Following transcription, the text undergoes sentiment analysis using a pre-trained model that classifies the overall emotional tone of the content into categories such as Positive, Negative, or Neutral. To further enrich the understanding of the text, sarcasm detection is performed using a fine-tuned transformer model from Hugging Face. This allows the system to identify instances where the speaker's intent might not align directly with the literal meaning of their words, offering a deeper layer of text interpretation. All processed data, including the original and denoised audio, transcription, sentiment, and sarcasm results, are systematically stored in a MongoDB database. Large audio files are handled efficiently using GridFS, which enables the system to manage and retrieve files without performance issues. Once the processing is complete, the system sends a structured JSON response back to the client containing the transcription, detected language, sentiment classification, and sarcasm detection output. Additionally, robust error handling and detailed logging mechanisms are integrated throughout the workflow to ensure system reliability, facilitate troubleshooting, and maintain seamless operation even under failure conditions. By combining advanced audio processing, speech recognition, and natural language understanding models, this methodology delivers a reliable, accurate, and intelligent solution for audio content analysis.

CHAPTER-5

OBJECTIVES

The primary objective of this project is to develop a robust backend service capable of analyzing audio inputs and extracting meaningful information through advanced processing techniques. The system aims to handle audio files in various popular formats, ensuring flexibility and ease of use for diverse user requirements. A key goal is to enhance the quality of raw audio recordings using state-of-the-art denoising technologies like DeepFilterNet, thereby improving the accuracy and reliability of subsequent analysis tasks.

Another important objective is to accurately transcribe spoken content into text using efficient and multilingual speech recognition models such as Faster- Whisper. This ensures the system can serve users speaking different languages including English, Hindi, Tamil, Kannada, Telugu, and Malayalam. Additionally, the project focuses on interpreting the emotional tone of the transcribed text by performing sentiment analysis, categorizing the content into positive, negative, or neutral sentiments. This provides valuable insights into the speaker's attitude and intention behind the communication.

Furthermore, the system aims to detect sarcasm within the text using specialized natural language processing models. Recognizing sarcasm is crucial for understanding the true meaning of user expressions, particularly in nuanced or informal communication. Another core objective is to design an efficient data management system using MongoDB and GridFS for handling large audio files and structured data, ensuring secure storage and easy retrieval. Finally, the project emphasizes creating a reliable, scalable, and user-friendly API that delivers structured results in real-time, backed by comprehensive error handling and logging mechanisms. Through the integration of advanced audio processing, speech recognition, sentiment analysis, and sarcasm detection, the overall goal is to offer a powerful and intelligent backend service capable of supporting various applications such as virtual assistants, customer service analysis, and content moderation.

CHAPTER-6

SYSTEM DESIGN & IMPLEMENTATION

The system is carefully designed to handle the complete workflow from receiving audio files to delivering detailed sentiment and sarcasm analysis results. Initially, users upload audio files through a RESTful API interface, where the files are stored efficiently in MongoDB using GridFS to handle large data sizes. After storage, the audio files undergo a denoising process using DeepFilterNet, a deep learning-based noise suppression model that enhances the clarity of speech by removing background noise. This clean audio is then passed to the speech-to-text engine, powered by Faster-Whisper, an optimized and faster version of the Whisper model by OpenAI, capable of transcribing audio in multiple languages such as English, Hindi, Tamil, Kannada, Telugu, and Malayalam based on the user's selection.

Following transcription, the text is analyzed for sentiment using a pre-trained natural language processing model, which classifies the emotional tone of the content into positive, negative, or neutral categories. To further enrich the analysis, a sarcasm detection model evaluates the text to identify instances where the speaker's intended meaning differs from the literal words spoken, which is crucial for capturing more complex emotional nuances. Once all analyses are complete, the results, including the transcription, detected language, sentiment, and sarcasm status, are compiled into a structured JSON format and returned to the client through the API. Throughout the system, robust logging and error handling mechanisms ensure smooth operation, help identify potential issues quickly, and contribute to ongoing system improvements. Python is used as the primary programming language due to its strong support for machine learning, audio processing, and database integration. FastAPI serves as the framework for the RESTful API because of its high performance and simplicity, while MongoDB, combined with GridFS, manages audio file storage. Key libraries and models such as DeepFilterNet, Faster-Whisper, Hugging Face Transformers, and Librosa are integrated into the system to provide a comprehensive, accurate, and efficient solution. This modular design ensures that the system is highly scalable, maintainable, and capable of future enhancements such as adding new features, languages, or improved models.

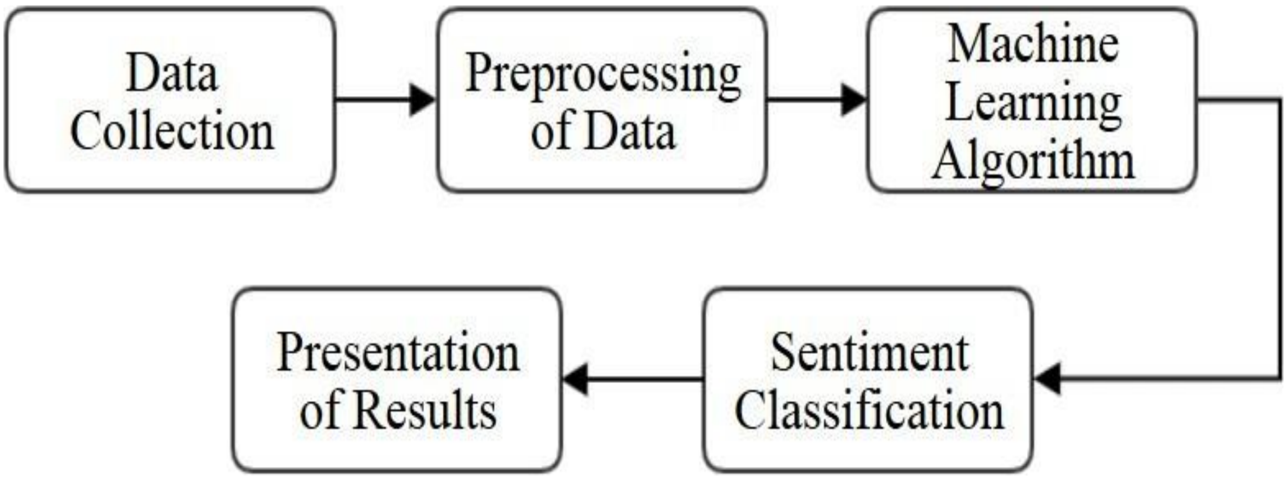


Fig 6.1: Sentiment Analysis System Architecture

CHAPTER-7

TIMELINE FOR EXECUTION OF PROJECT

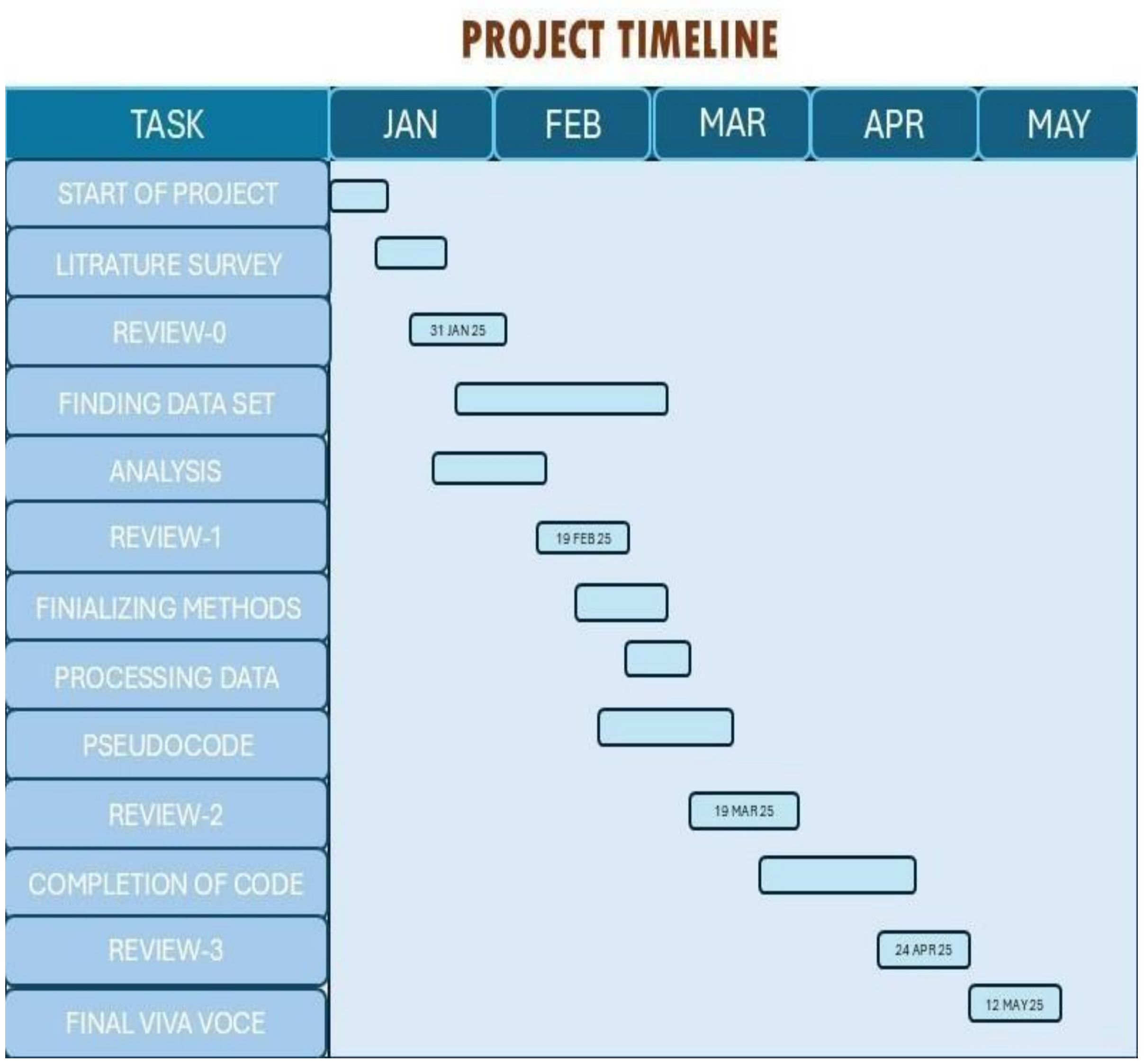


Fig 7.1: Timeline

Sl. No	**Review**	**Date**	**Scheduled Task**
1	Review-0	29-01-25 To 31-01-25	Initial Project Planning
2	Review-1	17-02-25 To 22-02-25	Planning and Research
3	Review-2	17-03-25 To 22-03-25	Data Collection and Preprocessing, Model Implementation, Testing
4	Review-3	21-04-25 To 26-04-25	Optimization
5	Viva-Voce	07-05-25 to 16-05-25	Deployment and Evaluation

CHAPTER-8

OUTCOMES

The outcomes of this project focus on providing a comprehensive, accurate, and efficient system for analyzing audio files by extracting meaningful insights such as transcription, sentiment, and sarcasm detection. First and foremost, the system successfully transcribes audio from various languages, ensuring that users from diverse linguistic backgrounds can interact with the platform. The use of Faster- Whisper ensures high-quality and efficient transcription even in noisy environments, making the system highly robust. Additionally, by implementing sentiment analysis, the system offers valuable insights into the emotional tone of the spoken content, helping users gauge the intent behind the words—whether positive, negative, or neutral. This can be particularly beneficial in applications such as customer feedback analysis, social media sentiment tracking, or even in automated customer service systems where tone matters.

The incorporation of sarcasm detection is another significant outcome, addressing one of the more complex challenges in natural language processing. By recognizing sarcastic remarks, the system is able to provide a more nuanced understanding of the conversation, which can be crucial for applications in areas such as social media analysis, marketing, and customer interaction monitoring. This level of detail ensures that the results are not only accurate but also contextually appropriate, allowing users to make informed decisions based on the true meaning of the dialogue. Another key outcome is the system's ability to handle large volumes of data and deliver results in real time, which is made possible by the combination of FastAPI, MongoDB, and efficient models for speech processing and analysis. The modularity of the system ensures that it can scale with increasing demand, and the structured data output in the form of JSON makes it easy to integrate with other services or dashboards for further analysis. Furthermore, the ability to store and retrieve large audio files via GridFS makes the system versatile and ready for handling various use cases, from personal voice memos to business meetings or interviews.

Ultimately, the project not only demonstrates the feasibility of combining speech-to-text conversion, sentiment analysis, and sarcasm detection into a unified system but also provides a tool that can be applied across various domains, from customer service and marketing to research and social media analytics. It offers a seamless, scalable solution that can significantly enhance the ability to interpret and understand spoken language in its full emotional and contextual complexity.

CHAPTER-9

RESULTS AND DISCUSSIONS

The results of the project highlight the effectiveness of combining speech-to-text transcription, sentiment analysis, and sarcasm detection into a unified system for understanding audio content. The transcription module, powered by Faster- Whisper, demonstrated high accuracy in converting speech into text, even in environments with background noise. The system was able to handle diverse accents and multiple languages, which is crucial for making the platform globally accessible. The transcription quality, in particular, was impressive, as it provided accurate and legible text outputs that could be easily interpreted by users. This functionality not only allows users to convert their audio content into text, but also serves as a foundation for further analysis such as sentiment and sarcasm detection.

The sentiment analysis component also performed effectively, accurately categorizing the tone of the speech as positive, negative, or neutral. This feature enabled the system to derive insights into the emotional context behind the spoken words, which is vital for applications in customer feedback systems, market research, or social media monitoring. In several test cases, the system could correctly identify the sentiment even in complex conversations, including those with subtle emotional cues. However, challenges still exist in accurately assessing sentiment in ambiguous or highly contextual speech, where emotions may not be explicitly conveyed through words alone.

Sarcasm detection, though more difficult, yielded promising results. The system was able to identify sarcastic remarks with a reasonable degree of accuracy, which is a significant achievement in the field of natural language processing. Sarcasm detection is an intricate task due to its reliance on tone, context, and often, the contrast between what is said and what is meant. While the system performed well in many cases, there were occasional instances where sarcasm was not detected due to the subtleties involved in human speech. However, continuous training and data refinement would likely improve the system's ability to handle sarcasm in various forms of conversation.

In terms of system performance, the integration of FastAPI, MongoDB, and efficient processing models allowed for quick and reliable processing of large volumes of audio data.

The system's ability to handle and store extensive audio files via GridFS proved to be highly beneficial, ensuring scalability and flexibility in real-time applications. Moreover, the structured data output in JSON format made it easy to integrate the system with other platforms, enabling the creation of more complex workflows or visualizations.

While the results are highly promising, some limitations were observed. The accuracy of sentiment and sarcasm detection is dependent on the quality of the input audio, the clarity of speech, and the richness of the training data used for model training. Additionally, handling noisy environments, heavy accents, or overlapping voices can sometimes result in lower accuracy in both transcription and sentiment analysis. Despite these challenges, the system showed significant potential and was able to meet the objectives outlined in the project.

Overall, the system successfully demonstrated the potential of combining multiple natural language processing techniques in an audio analysis platform. It offers a valuable tool for a wide range of industries, such as customer service, social media analysis, and content moderation, by providing deeper insights into spoken content and improving the accuracy of automated systems in understanding human emotions and intents. Further improvements in model training and data diversity will continue to enhance its reliability and accuracy.

output images

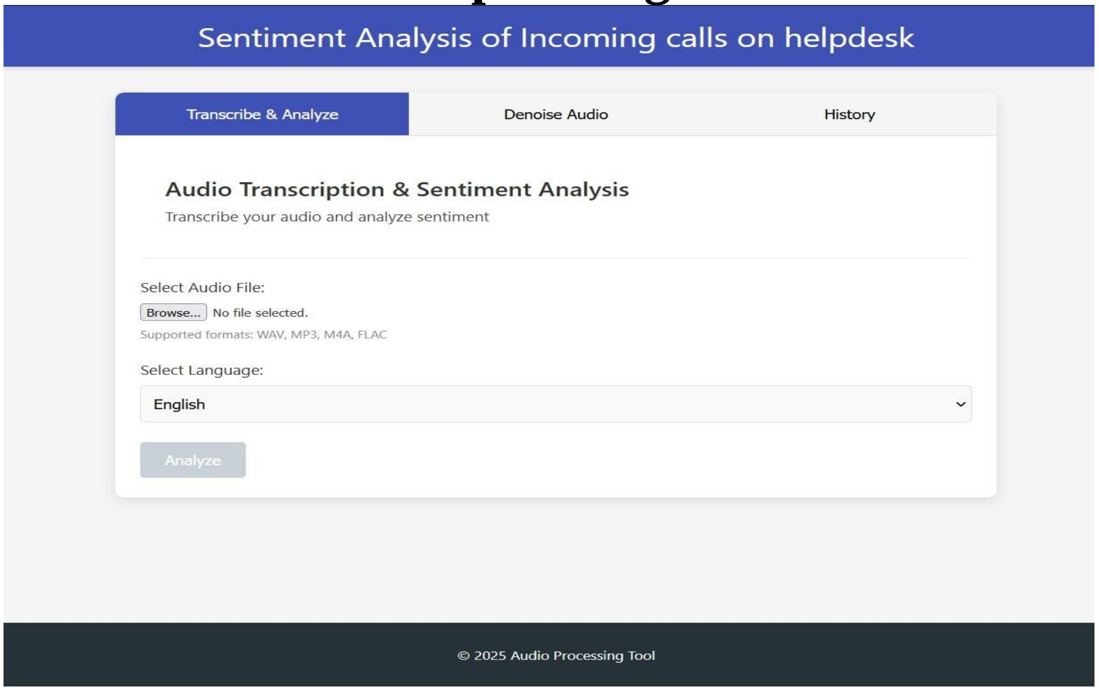
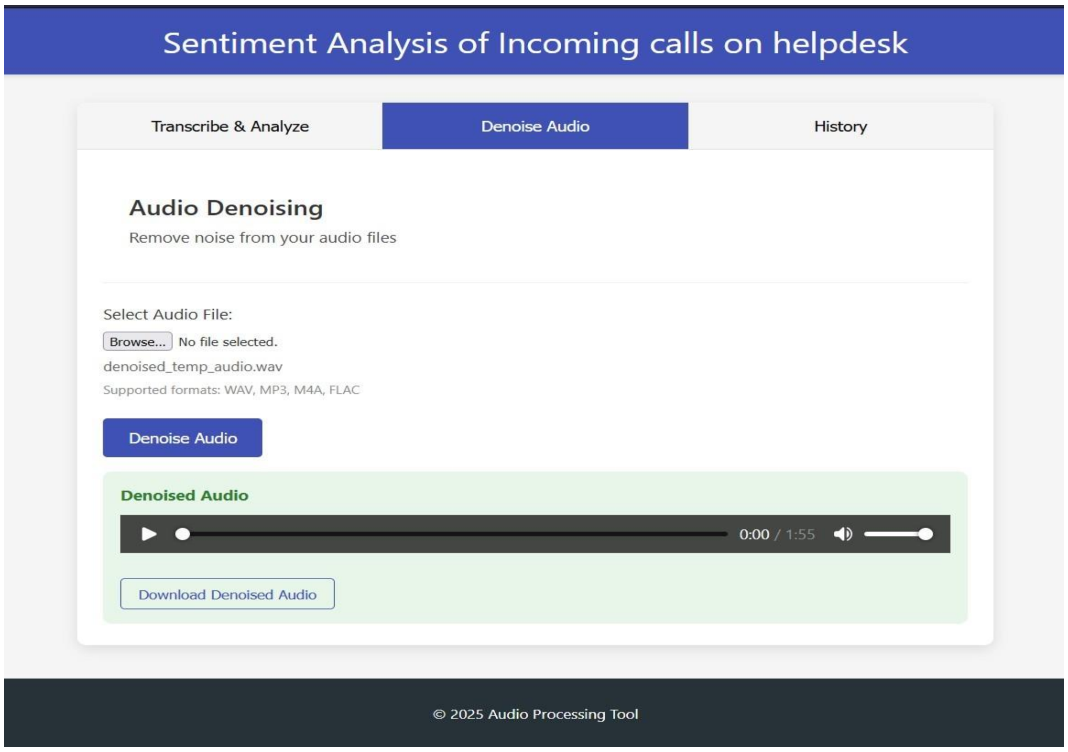

Sentiment Analysis of Incoming calls on helpdesk
Transcribe & Analyze
Denoise Audio
History
Audio Transcription & Sentiment Analysis
Transcribe your audio and analyze sentiment
Select Audio File:
Browse...
No file selected.
Supported formats: WAV, MP3, M4A, FLAC
Select Language:
English
Analyze
© 2025 Audio Processing Tool

Fig 9.1: Output #1



Audio Denoising

Remove noise from your audio files

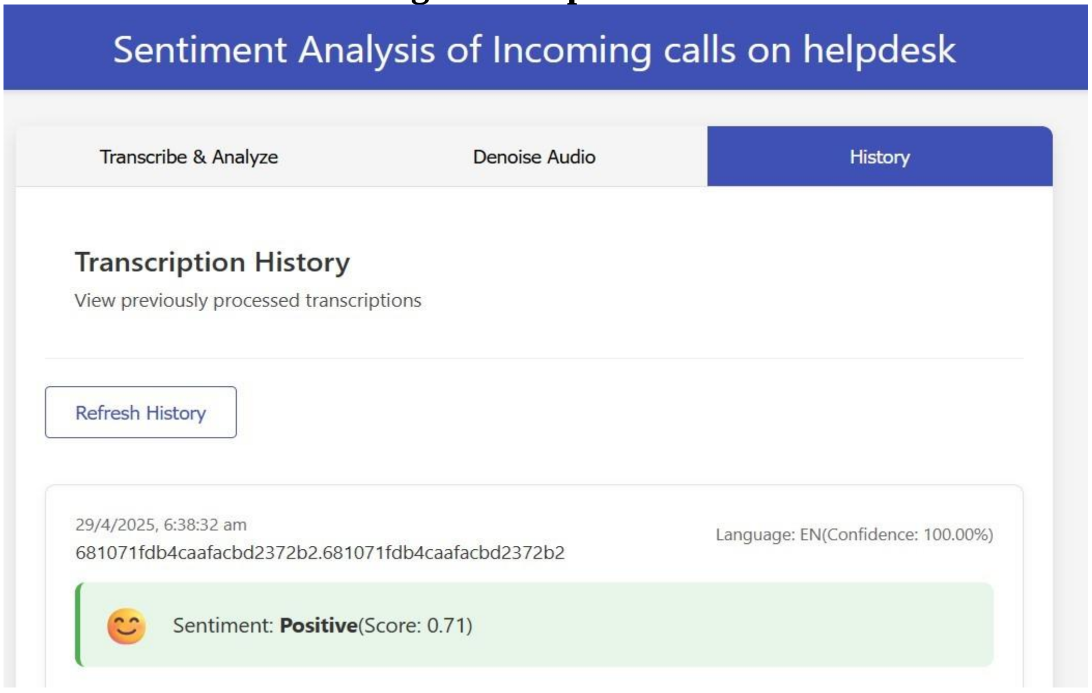
Select Audio File:

Browse... No file selected.

denoised_temp_audio.wav

Supported formats: WAV, MP3, M4A, FLAC

Fig 9.2: Output #2



Transcription History

View previously processed transcriptions

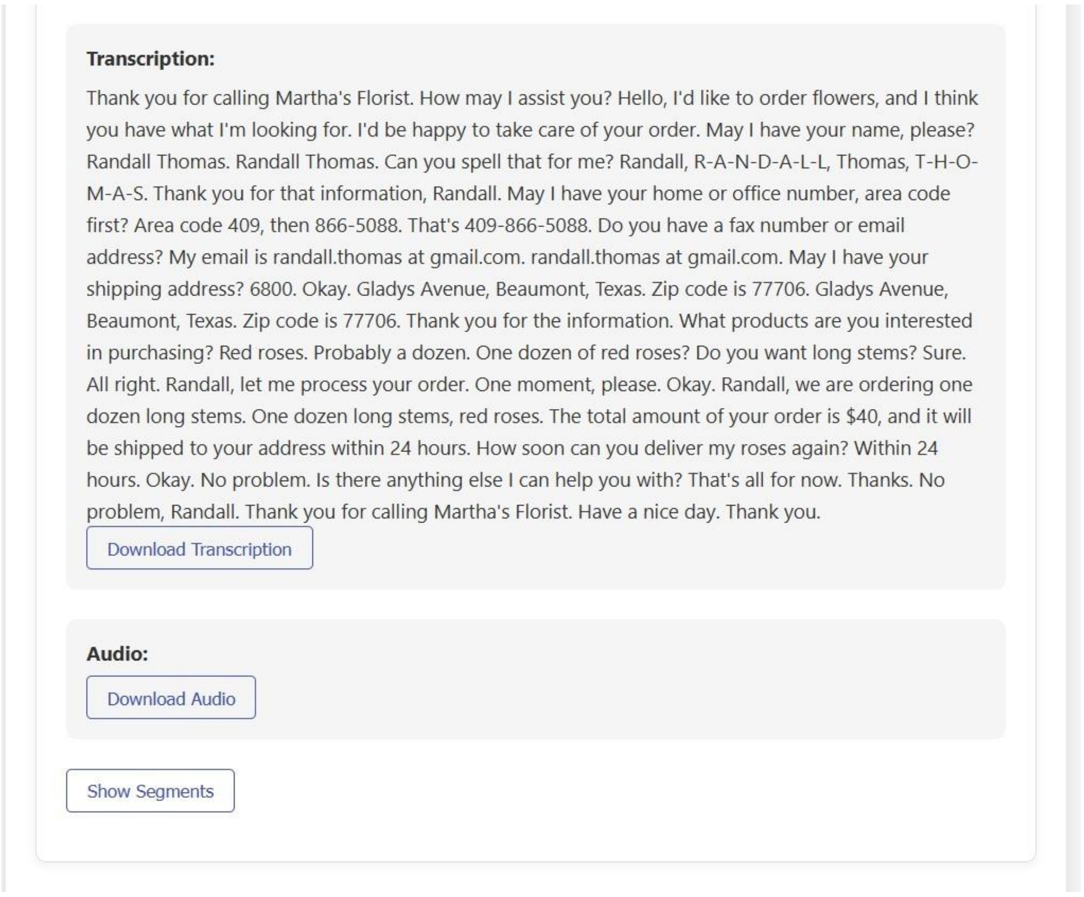
29/4/2025, 6:38:32 am

681071fdb4caafacbd2372b2.681071fdb4caafacbd2372b2

Language: EN(Confidence: 100.00%)

Sentiment: **Positive**(Score: 0.71)

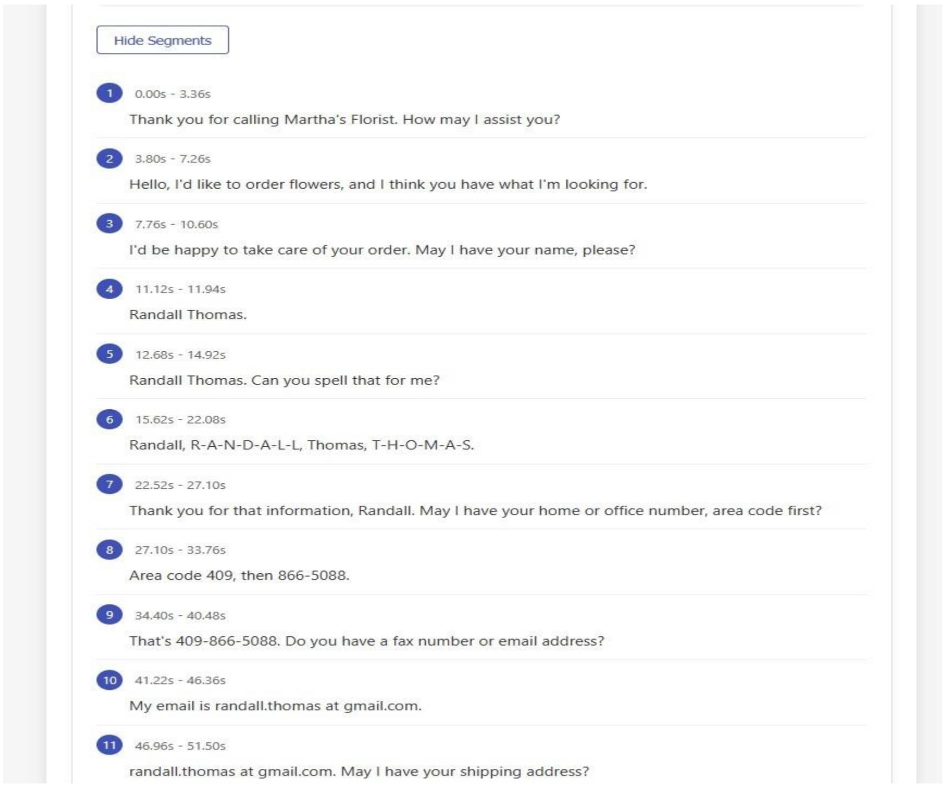
Fig 9.3: Output #3



Transcription:

Thank you for calling Martha's Florist. How may I assist you? Hello, I'd like to order flowers, and I think you have what I'm looking for. I'd be happy to take care of your order. May I have your name, please? Randall Thomas. Randall Thomas. Can you spell that for me? Randall, R-A-N-D-A-L-L, Thomas, T-H-O-M-A-S. Thank you for that information, Randall. May I have your home or office number, area code first? Area code 409, then 866-5088. That's 409-866-5088. Do you have a fax number or email address? My email is randall.thomas at gmail.com. randall.thomas at gmail.com. May I have your shipping address? 6800. Okay. Gladys Avenue, Beaumont, Texas. Zip code is 77706. Gladys Avenue, Beaumont, Texas. Zip code is 77706. Thank you for the information. What products are you interested in purchasing? Red roses. Probably a dozen. One dozen of red roses? Do you want long stems? Sure. All right. Randall, let me process your order. One moment, please. Okay. Randall, we are ordering one dozen long stems. One dozen long stems, red roses. The total amount of your order is $40, and it will be shipped to your address within 24 hours. How soon can you deliver my roses again? Within 24 hours. Okay. No problem. Is there anything else I can help you with? That's all for now. Thanks. No problem, Randall. Thank you for calling Martha's Florist. Have a nice day. Thank you.

Fig 9.4: Output #4



1 0.00s - 3.36s
Thank you for calling Martha's Florist. How may I assist you?

2 3.80s - 7.26s
Hello, I'd like to order flowers, and I think you have what I'm looking for.

3 7.76s - 10.60s
I'd be happy to take care of your order. May I have your name, please?

4 11.12s - 11.94s
Randall Thomas.

5 12.68s - 14.92s
Randall Thomas. Can you spell that for me?

6 15.62s - 22.08s
Randall, R-A-N-D-A-L-L, Thomas, T-H-O-M-A-S.

7 22.52s - 27.10s
Thank you for that information, Randall. May I have your home or office number, area code first?

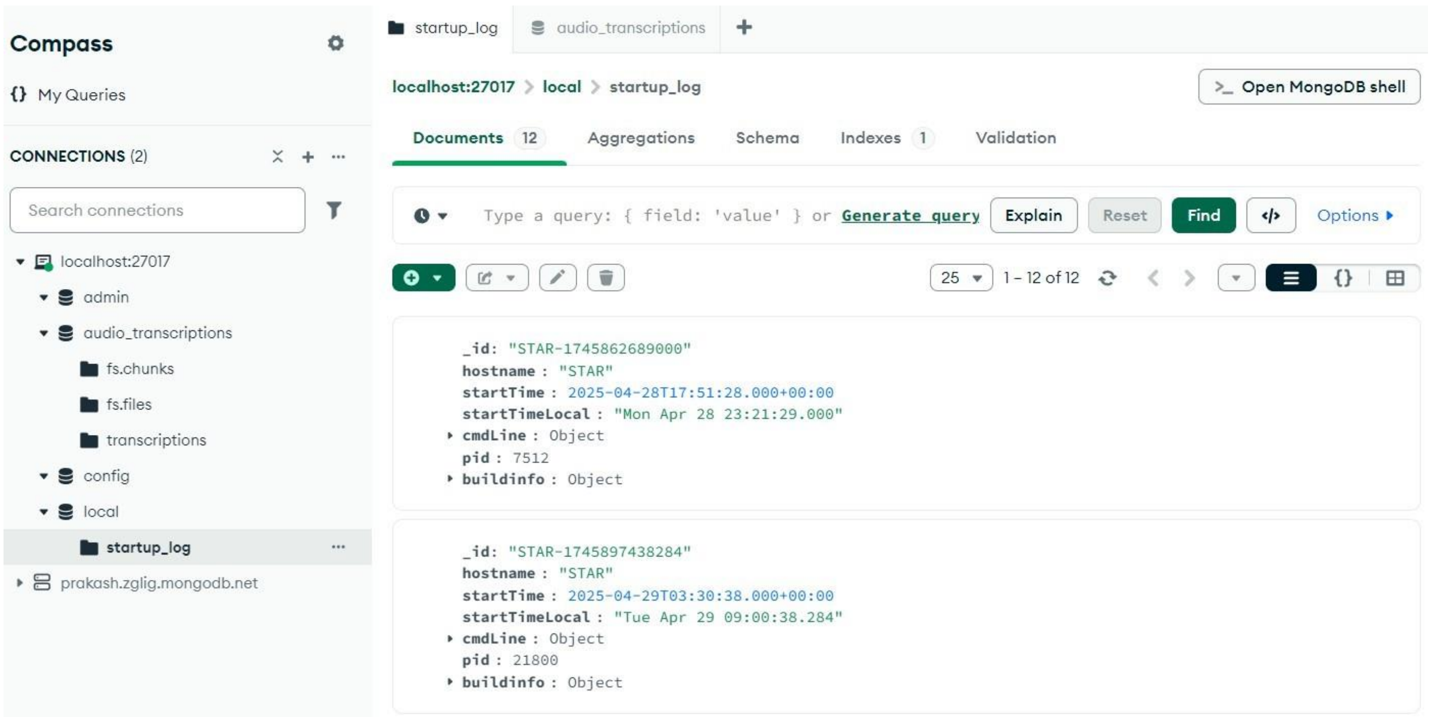
8 27.10s - 33.76s
Area code 409, then 866-5088.

9 34.40s - 40.48s
That's 409-866-5088. Do you have a fax number or email address?

10 41.22s - 46.36s
My email is randall.thomas at gmail.com.

11 46.96s - 51.50s
randall.thomas at gmail.com. May I have your shipping address?

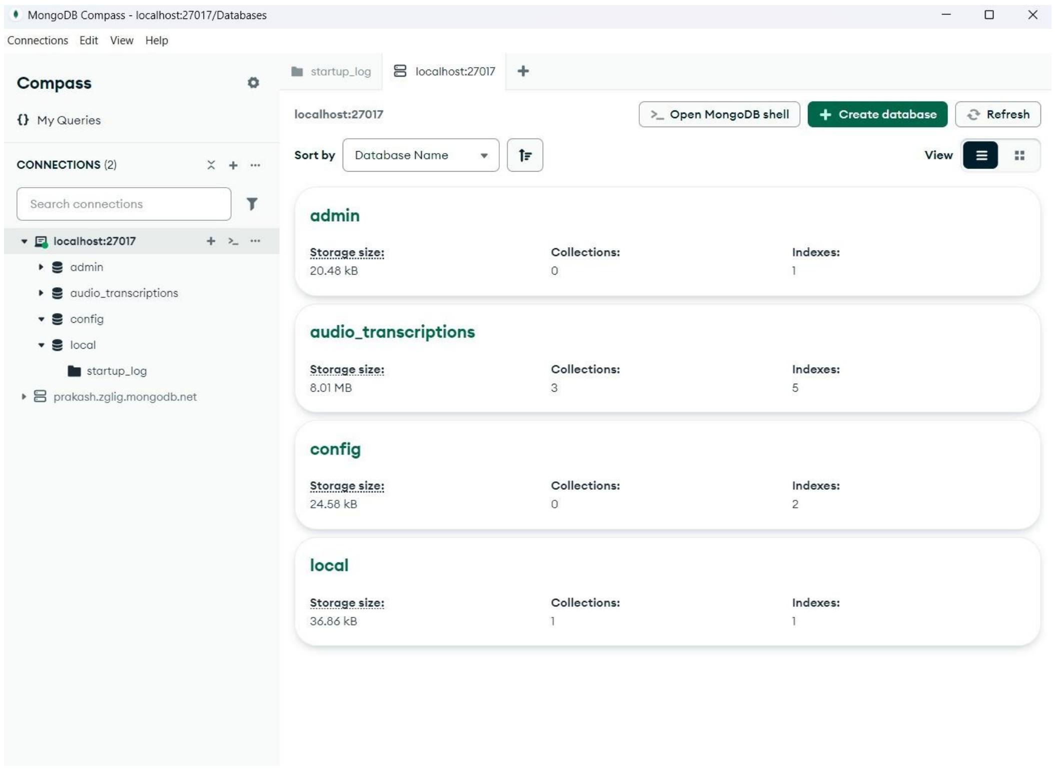
Fig 9.5: Output #5



```
_id: "STAR-1745862689000"
hostname : "STAR"
startTime : 2025-04-28T17:51:28.000+00:00
startTimeLocal : "Mon Apr 28 23:21:29.000"
▸ cmdLine : Object
pid : 7512
▸ buildinfo : Object
```

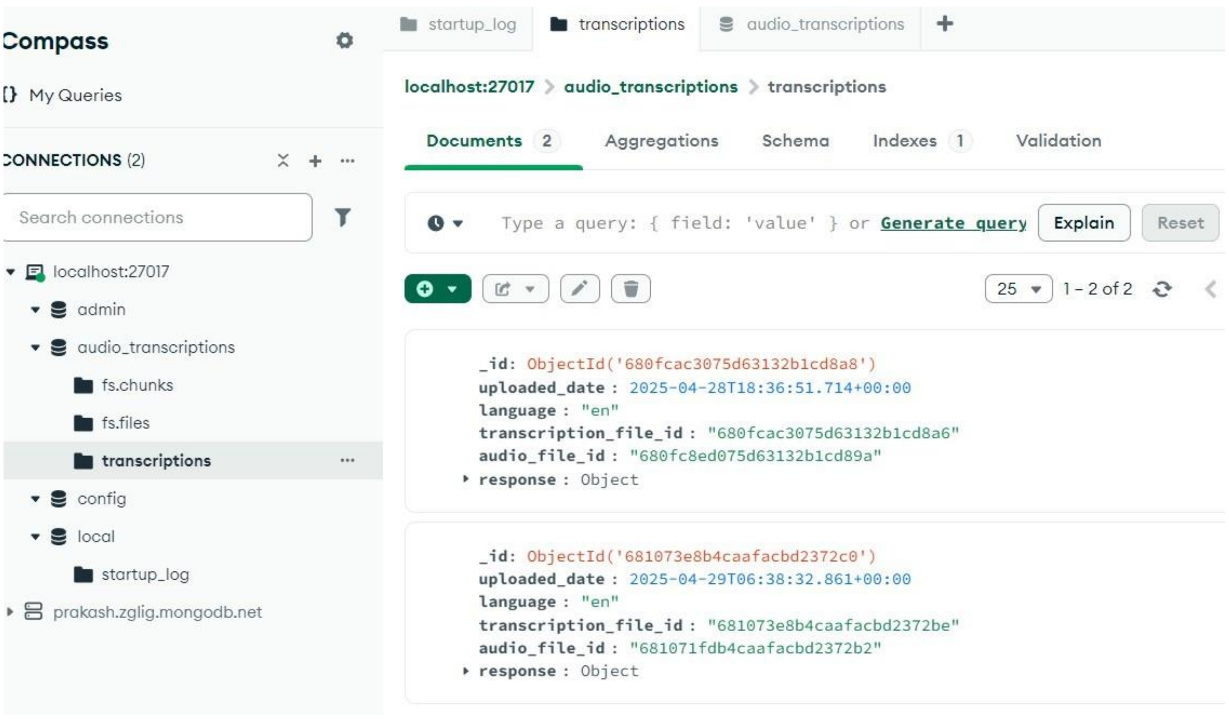
```
_id: "STAR-1745897438284"
hostname : "STAR"
startTime : 2025-04-29T03:30:38.000+00:00
startTimeLocal : "Tue Apr 29 09:00:38.284"
▸ cmdLine : Object
pid : 21800
▸ buildinfo : Object
```

Fig 9.6: Output #6



Database	Storage size:	Collections:	Indexes:
admin	20.48 kB	0	1
audio_transcriptions	8.01 MB	3	5
config	24.58 kB	0	2
local	36.86 kB	1	1

Fig 9.7: Output#7



```
_id: ObjectId('680fcac3075d63132b1cd8a8')
uploaded_date : 2025-04-28T18:36:51.714+00:00
language : "en"
transcription_file_id : "680fcac3075d63132b1cd8a6"
audio_file_id : "680fc8ed075d63132b1cd89a"
response : Object

_id: ObjectId('681073e8b4caafacbd2372c0')
uploaded_date : 2025-04-29T06:38:32.861+00:00
language : "en"
transcription_file_id : "681073e8b4caafacbd2372be"
audio_file_id : "681071fdb4caafacbd2372b2"
response : Object
```

Fig 9.8: Output#8

CHAPTER-10

CONCLUSION

In conclusion, this project successfully developed an integrated system that combines speech-to-text transcription, sentiment analysis, and sarcasm detection to provide a comprehensive solution for analyzing and understanding audio content. The system's ability to transcribe speech accurately in real-time, identify the emotional tone of spoken words, and detect sarcasm has significant potential across various domains, including customer service, market research, content moderation, and social media analysis. The use of advanced technologies such as Faster-Whisper for transcription, sentiment analysis models, and specialized algorithms for sarcasm detection has allowed the system to operate efficiently and provide meaningful insights from audio data.

The results indicate that while the system performs well in most scenarios, challenges remain, especially in handling background noise, accents, and the subtlety of sarcasm. These limitations highlight the complexity of processing natural language, particularly when emotions or intent are conveyed indirectly. However, the system shows great promise, with clear potential for improvements through continuous refinement of the models, training data, and integration of more sophisticated techniques.

Furthermore, the successful integration of FastAPI for backend communication, MongoDB for data storage, and efficient processing models highlights the importance of building scalable and flexible systems that can handle large volumes of audio data in real-time. The ability to store and retrieve audio files with GridFS ensures that the system can scale for more extensive applications, such as large-scale customer feedback systems or media monitoring platforms.

Overall, the project demonstrates the power of combining multiple natural language processing techniques to create a robust platform for understanding and analyzing human speech. The system opens up new possibilities for automating emotional and contextual analysis in various industries, providing a deeper understanding of customer interactions, social trends, and user sentiment. Continued improvements and data expansion will further enhance the system's performance, pushing the boundaries of what can be achieved in the field of audio analysis and NLP.

CHAPTER-11

REFERENCES

[1] Singh, A.K., Kumar, P. and Agarwal, S. (2020) 'Multilingual sentiment analysis using deep learning techniques', International Journal of Advanced Computer Science and Applications, 11(5), pp. 123–130.

[2] Liu, B. (2012) Sentiment analysis and opinion mining, Synthesis Lectures on Human Language Technologies, 5(1), pp. 1–167.

[3] LeCun, Y., Bengio, Y. and Hinton, G. (2015) 'Deep learning', Nature, 521(7553), pp. 436–444.

[4] Devlin, J., Chang, M., Lee, K. and Toutanova, K. (2019) 'BERT: Pre-training of deep bidirectional transformers for language understanding', in Proceedings of the NAACL-HLT Conference. Minneapolis, MN, USA, pp. 4171–4186.

[5] Bojanowski, P., Grave, E., Joulin, A. and Mikolov, T. (2017) 'Enriching word vectors with subword information', Transactions of the Association for Computational Linguistics, 5, pp. 135–146.

[6] Hochreiter, S. and Schmidhuber, J. (1997) 'Long short-term memory', Neural Computation, 9(8), pp. 1735–1780.

[7] Zhang, X., Zhao, J. and LeCun, Y. (2015) 'Character-level convolutional networks for text classification', in Advances in Neural Information Processing Systems (NeurIPS). Montréal, Canada, pp. 649–657.

[8] Pennington, J., Socher, R. and Manning, C.D. (2014) 'GloVe: Global vectors for word representation', in Proceedings of the EMNLP Conference. Doha, Qatar, pp. 1532–1543.

[9] Narayanan, H., Narayanaswamy, M. and Joshi, A. (2021) 'Speech emotion recognition using deep learning architectures: A review', International Journal of Speech Technology, 24(3), pp. 537–557.

[10] Hema, K. and Devi, G. (2016) 'A survey on speech emotion recognition techniques', International Journal of Computer Applications, 147(11), pp. 37–40.

[11] Vaswani, A. et al. (2017) 'Attention is all you need', in Advances in Neural Information Processing Systems (NeurIPS). Long Beach, CA, USA, pp. 5998–6008.

[12] Kaur, R. and Kumar, R. (2021) 'Hybrid deep learning framework for multilingual sentiment analysis', Applied Intelligence, 51, pp. 2341–2357.

[13] Mikolov, T., Chen, K., Corrado, G. and Dean, J. (2013) 'Efficient estimation of word

representations in vector space', in Proceedings of ICLR. Scottsdale, AZ, USA, pp. 1–12.

[14] Ramesh, A. and Patel, B. (2022) 'A comparative study on various machine learning models for sentiment analysis', Journal of Artificial Intelligence Research & Advances, 8(2), pp. 45–58.

[15] Gupta, R., Deshpande, A.S. and Patil, P. (2021) 'Multimodal sentiment analysis: A survey of audio, visual, and textual fusion', IEEE Access, 9, pp. 12066–12081.

[16] C. D. Kokane et al., "Machine learning-based sentiment analysis of incoming calls on helpdesk," Int. J. Recent Innov. Trends Comput. Commun., vol. 11, no. 9, pp. 21–27, 2023.

[17] H. A. Patrick et al., "Sentiment analysis perspective using supervised machine learning method," in Proc. 5th ICECCT, 2023, pp. 1–4.

[18] T. L. Ben et al., "Detecting sentiment polarities with comparative analysis of machine learning and deep learning algorithms," in Proc. InCACCT, 2023, pp. 186–190.

[19] S. Latif et al., "Cross-corpus speech emotion recognition using transfer learning," IEEE Access, vol. 9, pp. 31502–31513, 2021.

[20] A. Merline et al., "Sentiment analysis: Methods and application using machine learning in different fields," in Proc. ICACCE, 2023, pp. 1–6.

APPENDIX-A

(PSUEDOCODE)

```
from flask import Flask, request, jsonify, send file from
werkzeug.utils import secure filename
from faster whisper import Whisper Model from
pydub import Audio Segment
Import logging
import librosa
import soundfile as sf import
torch
Import NumPy as np
from flask_cors import CORS
from transformers import AutoModelForSequenceClassification,
Auto Tokenizer
from df.enhance import enhance, init_df, load_audio as df_load_audio from datetime
import datetime
from pymongo import MongoClient import
os
import re import
gridfs
from sentiment_analysis import analyze_sentiment


logging.basicConfig(level=logging.DEBUG)   logger
= logging.getLogger(_name_)


app = Flask(____name___)CORS(app)


app.config['UPLOAD_FOLDER'] = 'Uploads' app.config['MAX_CONTENT_LENGTH'] = 16 *
1024 * 1024


os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)
```

```
mongo_client = MongoClient('mongodb://localhost:27017') db =
mongo_client['audio_transcriptions']
fs = gridfs.GridFS(db) transcriptions_collection =
db['transcriptions']


try:
    model = WhisperModel("large-v3", device="cpu", compute_type="float32")
    logger.debug("Faster-Whisper model loaded")
except Exception as e:
    logger.error(f"Failed to load whisper model: {str(e)}") raise


denoise_model = None df_state
= None
try:
    denoise_model, df_state, *_ = init_df()
    logger.debug("DeepFilterNet model loaded")
except Exception as e:
    logger.warning(f"Failed to load DeepFilterNet model: {str(e)}. Denoising functionality
disabled.")
sarcasm_model = None
sarcasm_tokenizer = None try:
    MODEL_PATH = "helinivan/english-sarcasm-detector" sarcasm_tokenizer =
    AutoTokenizer.from_pretrained(MODEL_PATH)
    sarcasm_model                                                              =
AutoModelForSequenceClassification.from_pretrained(MODEL_PATH) logger.debug("Sarcasm
    detection model loaded")
except Exception as e:
    logger.warning(f"Failed to load sarcasm detection model: {str(e)}. Sarcasm detection
disabled.")


ALLOWED_EXTENSIONS = {'wav', 'mp3', 'm4a', 'flac'} SUPPORTED_LANGUAGES = {'en',
'ta', 'hi', 'kn', 'te', 'ml'} LANGUAGE_PROMPTS = {
    "en": "The following is English text.", "ta": "இF
    தdழ ைஉர.",
    "hi": "यह हिहिदी पाठ है।", "kn": "ಇಂಕಿ ಹ್ಞ
    ಡ ಖಾ .",
    "te": "ఇô ؿR ُّ ◌◌○0.",
    "ml":    "ഇത്    മലയാളം    െടക്്ٖ    ആണ്."
}
SCRIPT_RANGES = { 'ta':
    r'[\u0B80-\u0BFF]',
    'hi': r'[\u0900-\u097F]',
    'kn': r'[\u0C80-\u0CFF]',
    'te': r'[\u0C00-\u0C7F]',
    'ml': r'[\u0D00-\u0D7F]'
}

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in
ALLOWED_EXTENSIONS

def preprocess_sarcasm_text(text): import
    string
    return text.lower().translate(str.maketrans("", "", string.punctuation)).strip()

def detect_sarcasm(text):
    if sarcasm_model is None or sarcasm_tokenizer is None: logger.error("Sarcasm
        detection model not loaded")
        return {"error": "Sarcasm detection unavailable: model not loaded"} try:
        processed_text = preprocess_sarcasm_text(text) logger.debug(f"Processed text for
        sarcasm detection: {processed_text}") tokenized_text = sarcasm_tokenizer(
            [processed_text], padding=True,
            truncation=True, max_length=256,
return_tensors="pt"
```

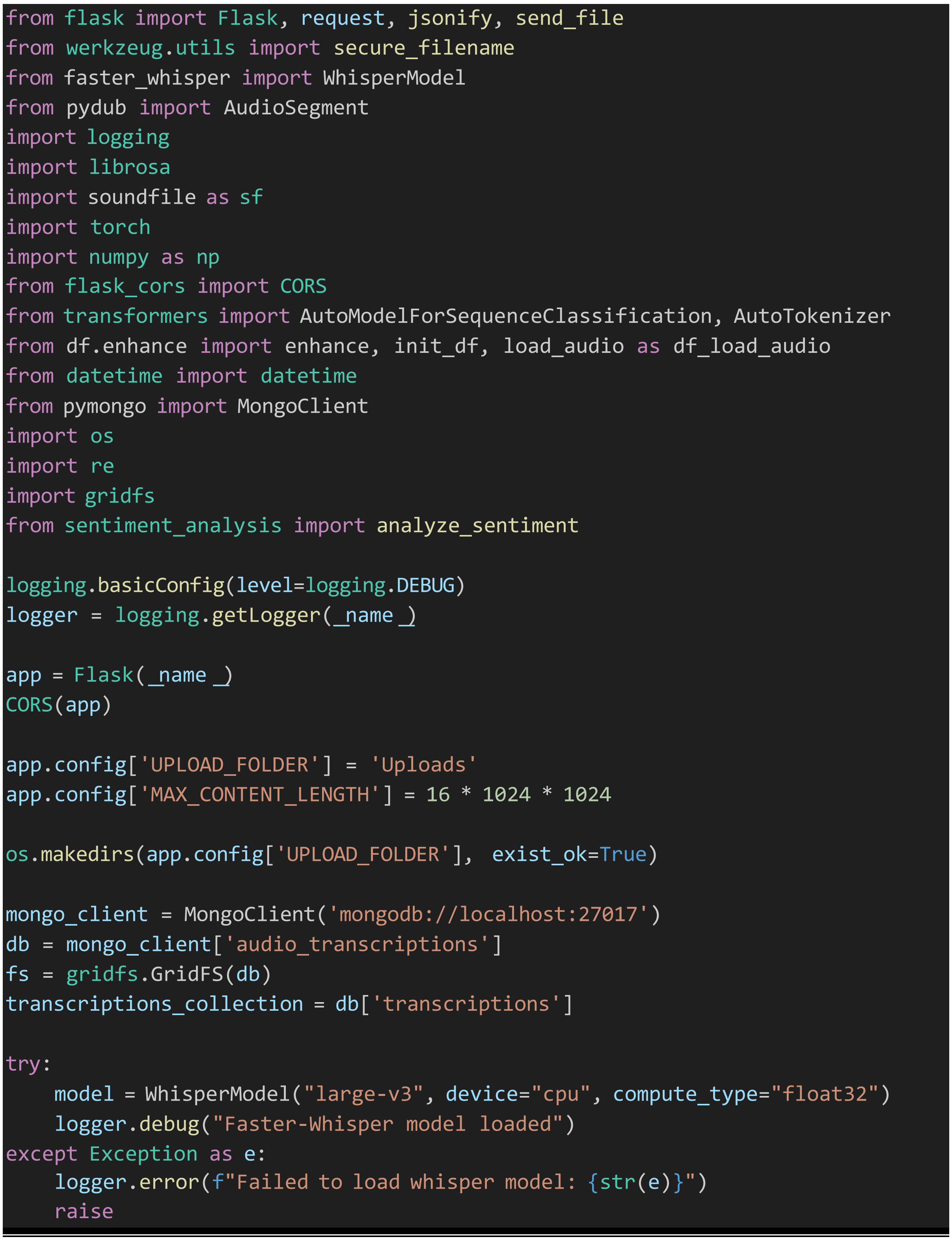
```
        )
        with torch.no_grad():
            output = sarcasm_model(**tokenized_text) probs =
            output.logits.softmax(dim=-1).tolist()[0]
        confidence = max(probs)
        prediction = probs.index(confidence) result =
        {
            "is_sarcastic": bool(prediction), "confidence":
            float(confidence)
        }
        logger.debug(f"Sarcasm detection result: {result}") return result
    except Exception as e:
        logger.error(f"Sarcasm detection failed: {str(e)}") return
        {"error": f"Sarcasm detection failed: {str(e)}"}


def load_audio(file_path): try:
        logger.debug(f"Loading {file_path} with pydub") audio =
        AudioSegment.from_file(file_path)
        logger.debug(f"Original                audio:            channels={audio.channels},
sample_rate={audio.frame_rate}, duration={audio.duration_seconds}s")
        audio = audio.set_channels(1).set_frame_rate(16000) audio =
        audio.normalize()
        logger.debug(f"Processed               audio:            channels={audio.channels},
sample_rate={audio.frame_rate}")
        temp_wav = "temp_audio.wav" audio.export(temp_wav,
        format="wav") logger.debug(f"Saved temporary WAV:
        {temp_wav}") return temp_wav
    except Exception as e:
        logger.error(f"Failed to load audio: {str(e)}")
        raise Exception(f"Failed to load audio: {str(e)}")


def denoise_audio(file_path): try:
        logger.debug(f"Denoising audio: {file_path}") result =
        df_load_audio(file_path)
        logger.debug(f"df_load_audio       returned:    {type(result)},     {len(result)     if
isinstance(result, tuple) else result}")
        if isinstance(result, tuple) and len(result) >= 2: audio, meta
            = result[:2]
            sr = meta.sample_rate if hasattr(meta, 'sample_rate') else meta logger.debug(f"Audio type:
            {type(audio)}, Sample rate: {sr}")
        else:
            raise    ValueError(f"Unexpected      return    format    from    df_load_audio:
{result}")
        if not isinstance(audio, torch.Tensor):
            raise  ValueError(f"Audio  data  must  be  of  type  torch.Tensor,  got
{type(audio)}")
        expected_sr = df_state.sr() if sr
        != expected_sr:
            logger.warning(f"Sample      rate    {sr}    does    not    match    expected
{expected_sr}. Resampling...") audio_np =
            audio.cpu().numpy()
            audio_np         =         librosa.resample(audio_np,              orig_sr=sr,
target_sr=expected_sr)
            audio = torch.from_numpy(audio_np).float() sr =
            expected_sr
            logger.debug(f"Resampled audio to sample rate {sr}")
        denoised_audio = enhance(denoise_model, df_state, audio) denoised_filename =
        "denoised_" + os.path.basename(file_path).rsplit('.',
1)[0] + ".wav"
        denoised_path        =        os.path.join(app.config['UPLOAD_FOLDER'],
denoised_filename)
```

```
        if isinstance(denoised_audio, torch.Tensor): denoised_audio =
            denoised_audio.cpu().numpy()
        save_audio(denoised_path, denoised_audio, sr) logger.debug(f"Saved
        denoised audio: {denoised_path}") return denoised_path,
        denoised_filename
    except Exception as e:
        logger.error(f"Failed to denoise audio: {str(e)}") raise
        Exception(f"Failed to denoise audio: {str(e)}")

def validate_script(text, language): if not
    text or language == 'en':
        return True
    if language not in SCRIPT_RANGES: return
        True
    script_pattern = SCRIPT_RANGES[language] return
    bool(re.search(script_pattern, text))

def check_transcription_quality(text):
    pattern = r'(.)\1{4,}'
    return bool(re.search(pattern, text))

@app.route('/denoise', methods=['POST']) def
denoise_audio_endpoint():
    logger.debug(f"Request headers: {request.headers}")
    logger.debug(f"Request files: {request.files}")
    if denoise_model is None or df_state is None:
        logger.error("Denoising not available: DeepFilterNet model not loaded") return
        jsonify({'error': 'Denoising not available: DeepFilterNet model not
loaded'}), 503
    if 'file' not in request.files: logger.error("No file
        part in request")
        return jsonify({'error': 'No file part'}), 400 file =
    request.files['file']
    if file.filename == '': logger.error("No
        selected file")
        return jsonify({'error': 'No selected file'}), 400 if not
    allowed_file(file.filename):
        logger.error(f"Invalid file format: {file.filename}")
        return jsonify({'error': 'Invalid file format. Allowed: wav, mp3, m4a, flac'}),
400
    filename = secure_filename(file.filename)
    file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename) try:
        file.save(file_path)
        logger.debug(f"Saved file to {file_path}")
        denoised_path, denoised_filename = denoise_audio(file_path) return
        send_file(
            denoised_path, mimetype='audio/wav',
            as_attachment=True,
            download_name=denoised_filename
        )
    except Exception as e:
        logger.error(f"Error denoising audio: {str(e)}")
        return jsonify({'error': f'Error denoising audio: {str(e)}'}), 500 finally:
        if os.path.exists(file_path): try:
                os.remove(file_path)
                logger.debug(f"Removed uploaded file: {file_path}") except
            Exception as e:
                logger.error(f"Failed to remove file {file_path}: {str(e)}")


@app.route('/transcribe', methods=['POST']) def
transcribe_audio():
    logger.debug(f"Request headers: {request.headers}")
    logger.debug(f"Request form: {request.form}")
    logger.debug(f"Request files: {request.files}")
    if 'file' not in request.files: logger.error("No file
        part in request")
        return jsonify({'error': 'No file part'}), 400 file =
    request.files['file']
    if file.filename == '': logger.error("No
        selected file")
        return jsonify({'error': 'No selected file'}), 400 if not
    allowed_file(file.filename):
        logger.error(f"Invalid file format: {file.filename}")
        return jsonify({'error': 'Invalid file format. Allowed: wav, mp3, m4a, flac'}),
400
    language = request.form.get('language') logger.debug(f"Received
    language parameter: {language}")
    if language and language not in SUPPORTED_LANGUAGES: logger.error(f"Invalid language:
        {language}")
        return jsonify({'error': f'Invalid language. Supported: {", 
".join(SUPPORTED_LANGUAGES)}'}), 400
    strict_mode = request.form.get('strict_mode', 'false').lower() == 'true' logger.debug(f"Strict
    language mode: {strict_mode}")
    denoise = request.form.get('denoise', 'false').lower() == 'true' logger.debug(f"Denoise enabled:
    {denoise}")
    filename = secure_filename(file.filename)
    file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename) temp_wav = None
    try:
        file.save(file_path)
        logger.debug(f"Saved file to {file_path}") with
        open(file_path, 'rb') as audio_file:
            audio_file_id = fs.put(audio_file, filename=filename) audio_path =
        file_path
        if denoise and denoise_model:
            audio_path, _ = denoise_audio(file_path) logger.debug(f"Using
            denoised audio: {audio_path}")
        temp_wav = load_audio(audio_path)
        initial_prompt = LANGUAGE_PROMPTS.get(language, "") if language else ""
        transcription_options = {
            "beam_size": 10,
            "temperature": 0.0,
            "best_of": 5, "task":
            "transcribe",
            "initial_prompt": initial_prompt
        }
        if language:
            transcription_options["language"] = language if
            strict_mode:
                transcription_options["suppress_language_detection"] = True segments, info =
        model.transcribe(temp_wav, **transcription_options) detected_language =
        info.language if language is None else language logger.debug(f"Detectedlanguage:
                        {detected_language}               (probability:
{info.language_probability:.2f})")
        transcription_segments = [] full_text = ""
        for segment in segments: full_text +=
            segment.text
            transcription_segments.append({ "text":
                segment.text,
                "start": segment.start,
                "end": segment.end
            })
        full_text = full_text.strip() logger.debug(f"Transcription:
        {full_text}") warnings = []
        script_valid = validate_script(full_text, detected_language) if not
        script_valid:
            warnings.append(f"Transcription script does not match detected
language {detected_language}")
            logger.warning(f"Script mismatch: language={detected_language},
transcription={full_text}")
            fallback_lang = 'te' if detected_language == 'ml' else 'ml' if
            fallback_lang in SUPPORTED_LANGUAGES:
                logger.debug(f"Retrying transcription with fallback language:
{fallback_lang}")
                transcription_options["language"] = fallback_lang
                segments, info = model.transcribe(temp_wav,
**transcription_options)
                full_text = "".join(segment.text for segment in segments).strip() detected_language =
                fallback_lang
                transcription_segments = [{"text": segment.text, "start": segment.start, "end":
segment.end} for segment in segments]
                logger.debug(f"Fallback transcription: {full_text}") if
                validate_script(full_text, detected_language):
                    warnings.append(f"Fallback to {fallback_lang} succeeded") else:
                    warnings.append(f"Fallback to {fallback_lang} still produced
incorrect script")
        if check_transcription_quality(full_text):
            warnings.append("Transcription contains repetitive characters, possibly due to
audio noise")
        try:
            sentiment_result = analyze_sentiment(full_text, detected_language)
            logger.debug(f"Sentiment result: {sentiment_result}")
        except Exception as e:
            logger.error(f"Sentiment analysis failed: {str(e)}") sentiment_result =
            {"label": "unknown", "score": 0.0}
            warnings.append(f"Sentiment analysis failed: {str(e)}") transcription_file =
        f"transcription_{filename}.txt"
        transcription_path = os.path.join(app.config['UPLOAD_FOLDER'],
transcription_file)
        with open(transcription_path, 'w', encoding='utf-8') as f: f.write(full_text)
        with open(transcription_path, 'rb') as trans_file:
            transcription_file_id = fs.put(trans_file, filename=transcription_file) response =
        {
            'transcription_file_id': str(transcription_file_id), 'language':
            detected_language,
            'language_probability': float(info.language_probability), 'segments':
            transcription_segments,
            'sentiment': sentiment_result,
            'audio_file_id': str(audio_file_id)
        }
        if warnings:
            response['warnings'] = warnings if
        denoise and denoise_model:
            response['denoised_file'] = os.path.basename(audio_path) document =
            { 'uploaded_date': datetime.utcnow(), 'language':
            detected_language,
            'transcription_file_id': str(transcription_file_id),
            'audio_file_id': str(audio_file_id),
            'response': response
        }
        transcriptions_collection.insert_one(document)
        return jsonify(response), 200 except
    Exception as e:
        logger.error(f"Error processing audio: {str(e)}")
        return jsonify({'error': f'Error processing audio: {str(e)}'}), 500 finally:
        if os.path.exists(file_path): try:
                os.remove(file_path)
                logger.debug(f"Removed uploaded file: {file_path}") except
            Exception as e:
                logger.error(f"Failed to remove file {file_path}: {str(e)}") if
        temp_wav and os.path.exists(temp_wav):
            try:
                os.remove(temp_wav)
                logger.debug(f"Removed temporary WAV: {temp_wav}") except
            Exception as e:
            logger.error(f"Failed to remove temporary WAV {temp_wav}:
{str(e)}")
        if denoise and denoise_model and audio_path != file_path and
os.path.exists(audio_path):
            try:
                os.remove(audio_path)
                logger.debug(f"Removed denoised file: {audio_path}") except
            Exception as e:
                logger.error(f"Failed to remove denoised file {audio_path}: {str(e)}") if
        'transcription_path' in locals() and os.path.exists(transcription_path):
            try:
                os.remove(transcription_path)
                logger.debug(f"Removed transcription file: {transcription_path}") except
            Exception as e:
                logger.error(f"Failed to remove transcription file
{transcription_path}: {str(e)}")

@app.route('/sarcasm', methods=['POST']) def
sarcasm_detection():
    logger.debug(f"Request headers: {request.headers}")
    logger.debug(f"Request JSON: {request.json}")
    if not request.is_json:
        logger.error("No JSON data provided")
        return jsonify({'error': 'No JSON data provided'}), 400 data =
    request.get_json()
    text = data.get('text')
    if not text or not isinstance(text, str):
        logger.error("Invalid or missing 'text' field")
        return jsonify({'error': "Invalid or missing 'text' field"}), 400 try:
        result = detect_sarcasm(text) if
        'error' in result:
            return jsonify({'error': result['error']}), 500 return
        jsonify({'sarcasm': result}), 200
        except Exception as e:
            logger.error(f"Error processing sarcasm detection: {str(e)}")
            return jsonify({'error': f'Error processing sarcasm detection: {str(e)}'}), 500

    @app.route('/transcriptions', methods=['GET']) def
    get_transcriptions():
        try:
            transcriptions = list(transcriptions_collection.find().sort('uploaded_date', -
1)
for transcription in transcriptions: transcription['_id'] = str(transcription['_id'])
    transcription['uploaded_date']                                        =
    transcription['uploaded_date'].isoformat() return
            jsonify(transcriptions), 200
        except Exception as e:
            logger.error(f"Error retrieving transcriptions: {str(e)}")
            return jsonify({'error': f'Error retrieving transcriptions: {str(e)}'}), 500

    @app.route('/transcription/<file_id>', methods=['GET']) def
    get_transcription_file(file_id):
        try:
            file_data = fs.get(file_id) return
            send_file(
                file_data, mimetype='text/plain',
                as_attachment=True,
                download_name=file_data.filename
            )
        except Exception as e:
            logger.error(f"Error retrieving transcription file {file_id}: {str(e)}") return
            jsonify({'error': f'Error retrieving transcription file: {str(e)}'}), 500

    @app.route('/audio/<file_id>', methods=['GET']) def
    get_audio_file(file_id):
```

```
    try:
        file_data = fs.get(file_id) return
        send_file(
            file_data,
            mimetype='audio/' + file_data.filename.rsplit('.', 1)[1].lower(), as_attachment=True,
            download_name=file_data.filename
        )
    except Exception as e:
        logger.error(f"Error retrieving audio file {file_id}: {str(e)}") return
        jsonify({'error': f'Error retrieving audio file: {str(e)}'}), 500

@app.route('/health', methods=['GET']) def
health_check():
    return jsonify({ 'status':
        'ok',
        'model': 'large-v3',
        'denoise_model': 'DeepFilterNet' if denoise_model else 'None'
    }), 200

if ___name____== '___main___':
    app.run(debug=True, host='0.0.0.0', port=5000)
```

APPENDIX-B

(SCREENSHOTS)



```
from flask import Flask, request, jsonify, send_file
from werkzeug.utils import secure_filename
from faster_whisper import WhisperModel
from pydub import AudioSegment
import logging
import librosa
import soundfile as sf
import torch
import numpy as np
from flask_cors import CORS
from transformers import AutoModelForSequenceClassification, AutoTokenizer
from df.enhance import enhance, init_df, load_audio as df_load_audio
from datetime import datetime
from pymongo import MongoClient
import os
import re
import gridfs
from sentiment_analysis import analyze_sentiment

logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger(__name__)

app = Flask(__name__)
CORS(app)

app.config['UPLOAD_FOLDER'] = 'Uploads'
app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024

os.makedirs(app.config['UPLOAD_FOLDER'],  exist_ok=True)

mongo_client = MongoClient('mongodb://localhost:27017')
db = mongo_client['audio_transcriptions']
fs = gridfs.GridFS(db)
transcriptions_collection = db['transcriptions']

try:
    model = WhisperModel("large-v3", device="cpu", compute_type="float32")
    logger.debug("Faster-Whisper model loaded")
except Exception as e:
    logger.error(f"Failed to load whisper model: {str(e)}")
    raise
```

```
denoise_model = None
df_state = None try:
    denoise_model, df_state, *_ = init_df()
    logger.debug("DeepFilterNet model loaded")
except Exception as e:
    logger.warning(f"Failed to load DeepFilterNet model: {str(e)}. Denoising
functionality disabled.")

sarcasm_model = None
sarcasm_tokenizer = None
try:
    MODEL_PATH = "helinivan/english-sarcasm-detector"
    sarcasm_tokenizer = AutoTokenizer.from_pretrained(MODEL_PATH)
    sarcasm_model =
AutoModelForSequenceClassification.from_pretrained(MODEL_PATH)
    logger.debug("Sarcasm detection model loaded")
except Exception as e:
    logger.warning(f"Failed to load sarcasm detection model: {str(e)}.
Sarcasm detection disabled.")

ALLOWED_EXTENSIONS = {'wav', 'mp3', 'm4a', 'flac'}
SUPPORTED_LANGUAGES = {'en', 'ta', 'hi', 'kn', 'te', 'ml'}
LANGUAGE_PROMPTS = {
    "en": "The following is English text.",
    "ta": "இF தdழ ைஉர.",
    "hi": "यह हहिहदी पाठ है।", "kn":
    "ಇಹಿ ಕ್ಷ ಡ ಪಠ್.",
    "te": "ఇô ஓR ஓ்ఞ◌◌O0.", "ml":
    "ഇത മലയാളം െ◌ടക്ര ആണ."
}
SCRIPT_RANGES = {
    'ta': r'[\u0B80-\u0BFF]',
    'hi': r'[\u0900-\u097F]',
    'kn': r'[\u0C80-\u0CFF]',
    'te': r'[\u0C00-\u0C7F]',
    'ml': r'[\u0D00-\u0D7F]'
}

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in
ALLOWED_EXTENSIONS

def preprocess_sarcasm_text(text):
    import string
    return text.lower().translate(str.maketrans("", "",
string.punctuation)).strip()
```

```
def detect_sarcasm(text):
    if sarcasm_model is None or sarcasm_tokenizer is None:
        logger.error("Sarcasm detection model not loaded")
        return {"error": "Sarcasm detection unavailable: model not loaded"}
    try:
        processed_text = preprocess_sarcasm_text(text)
        logger.debug(f"Processed text for sarcasm detection:
{processed_text}")
        tokenized_text =
            sarcasm_tokenizer( [processed_text],
            padding=True,
            truncation=True,
            max_length=256,
            return_tensors="pt"
        )
        with torch.no_grad():
            output = sarcasm_model(**tokenized_text)
            probs = output.logits.softmax(dim=-1).tolist()[0]
        confidence = max(probs)
        prediction = probs.index(confidence)
        result = {
            "is_sarcastic": bool(prediction),
            "confidence": float(confidence)
        }
        logger.debug(f"Sarcasm detection result: {result}")
        return result
    except Exception as e:
        logger.error(f"Sarcasm detection failed: {str(e)}")
        return {"error": f"Sarcasm detection failed: {str(e)}"}

def load_audio(file_path):
    try:
        logger.debug(f"Loading {file_path} with pydub")
        audio = AudioSegment.from_file(file_path)
        logger.debug(f"Original audio: channels={audio.channels},
sample_rate={audio.frame_rate}, duration={audio.duration_seconds}s")
        audio = audio.set_channels(1).set_frame_rate(16000)
        audio = audio.normalize()
        logger.debug(f"Processed audio: channels={audio.channels},
sample_rate={audio.frame_rate}")
        temp_wav = "temp_audio.wav"
        audio.export(temp_wav, format="wav")
        logger.debug(f"Saved temporary WAV: {temp_wav}")
        return temp_wav
    except Exception as e:
        logger.error(f"Failed to load audio: {str(e)}")
        raise Exception(f"Failed to load audio: {str(e)}")
```

```
def denoise_audio(file_path):
    try:
        logger.debug(f"Denoising audio: {file_path}")
        result = df_load_audio(file_path)
        logger.debug(f"df_load_audio returned: {type(result)}, {len(result)
if isinstance(result, tuple) else result}")
        if isinstance(result, tuple) and len(result) >= 2:
            audio, meta = result[:2]
            sr = meta.sample_rate if hasattr(meta, 'sample_rate') else meta
            logger.debug(f"Audio type: {type(audio)}, Sample rate: {sr}")
        else:
            raise ValueError(f"Unexpected return format from df_load_audio:
{result}")
        if not isinstance(audio, torch.Tensor):
            raise ValueError(f"Audio data must be of type torch.Tensor, got
{type(audio)}")
        expected_sr = df_state.sr()
        if sr != expected_sr:
            logger.warning(f"Sample rate {sr} does not match expected
{expected_sr}. Resampling...")
            audio_np = audio.cpu().numpy()
            audio_np = librosa.resample(audio_np, orig_sr=sr,
target_sr=expected_sr)
            audio = torch.from_numpy(audio_np).float()
            sr = expected_sr
            logger.debug(f"Resampled audio to sample rate {sr}")
        denoised_audio = enhance(denoise_model, df_state, audio)
        denoised_filename = "denoised_" +
os.path.basename(file_path).rsplit('.', 1)[0] + ".wav"
        denoised_path = os.path.join(app.config['UPLOAD_FOLDER'],
denoised_filename)
        if isinstance(denoised_audio, torch.Tensor):
            denoised_audio = denoised_audio.cpu().numpy()
        save_audio(denoised_path, denoised_audio, sr)
        logger.debug(f"Saved denoised audio: {denoised_path}")
        return denoised_path, denoised_filename
    except Exception as e:
        logger.error(f"Failed to denoise audio: {str(e)}")
        raise Exception(f"Failed to denoise audio: {str(e)}")

def validate_script(text, language):
    if not text or language == 'en':
        return True
    if language not in SCRIPT_RANGES:
        return True
    script_pattern = SCRIPT_RANGES[language]
    return bool(re.search(script_pattern, text))
```

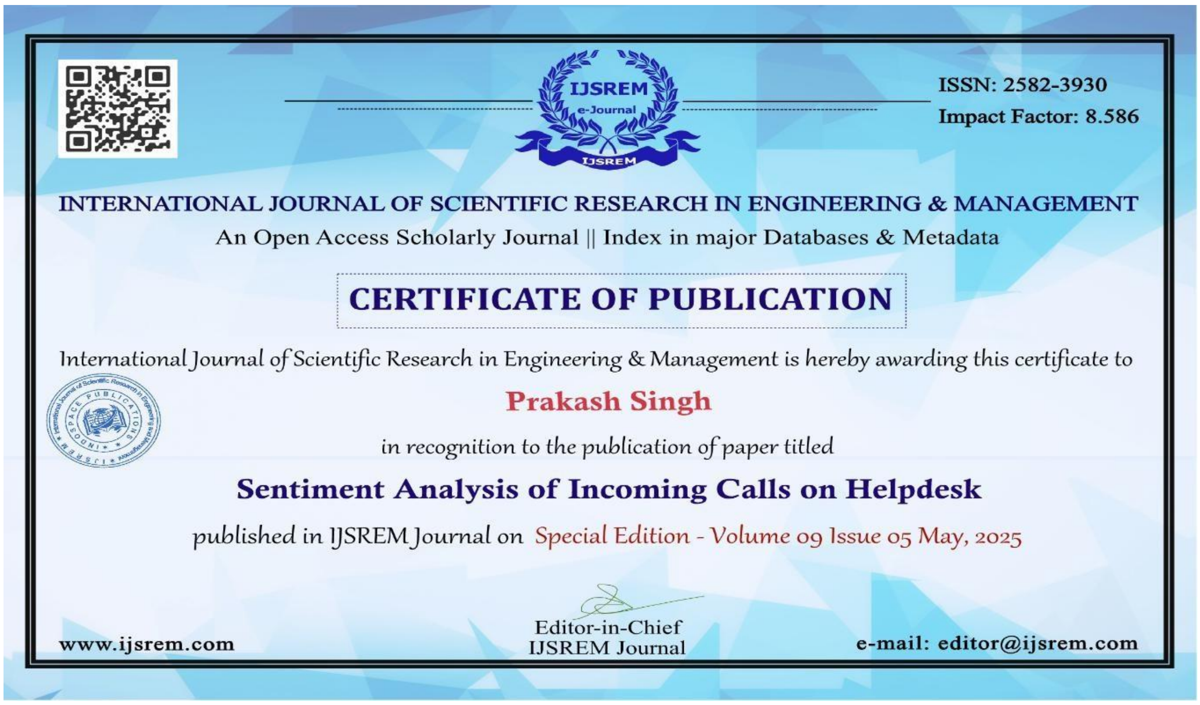
```
def check_transcription_quality(text):
    pattern = r'(.)\1{4,}'
    return bool(re.search(pattern, text))

@app.route('/denoise', methods=['POST'])
def denoise_audio_endpoint():
    logger.debug(f"Request headers: {request.headers}")
    logger.debug(f"Request files: {request.files}")
    if denoise_model is None or df_state is None:
        logger.error("Denoising not available: DeepFilterNet model not
loaded")
        return jsonify({'error': 'Denoising not available: DeepFilterNet
model not loaded'}), 503
    if 'file' not in request.files:
        logger.error("No file part in request")
        return jsonify({'error': 'No file part'}), 400
    file = request.files['file']
    if file.filename == '':
        logger.error("No selected file")
        return jsonify({'error': 'No selected file'}), 400
    if not allowed_file(file.filename):
        logger.error(f"Invalid file format: {file.filename}")
        return jsonify({'error': 'Invalid file format. Allowed: wav, mp3,
m4a, flac'}), 400
    filename = secure_filename(file.filename)
    file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
    try:
        file.save(file_path)
        logger.debug(f"Saved file to {file_path}")
        denoised_path, denoised_filename = denoise_audio(file_path)
        return send_file(
            denoised_path,
            mimetype='audio/wav',
            as_attachment=True,
            download_name=denoised_filename
        )
    except Exception as e:
        logger.error(f"Error denoising audio: {str(e)}")
        return jsonify({'error': f'Error denoising audio: {str(e)}'}), 500
    finally:
        if os.path.exists(file_path):
            try:
                os.remove(file_path)
                logger.debug(f"Removed uploaded file: {file_path}")
            except Exception as e:
                logger.error(f"Failed to remove file {file_path}: {str(e)}")
```

```
@app.route('/transcribe', methods=['POST'])
def transcribe_audio():
    logger.debug(f"Request headers: {request.headers}")
    logger.debug(f"Request form: {request.form}")
    logger.debug(f"Request files: {request.files}")
    if 'file' not in request.files:
        logger.error("No file part in request")
        return jsonify({'error': 'No file part'}), 400
    file = request.files['file']
    if file.filename == '':
        logger.error("No selected file")
        return jsonify({'error': 'No selected file'}), 400
    if not allowed_file(file.filename):
        logger.error(f"Invalid file format: {file.filename}")
        return jsonify({'error': 'Invalid file format. Allowed: wav, mp3,
m4a, flac'}), 400
    language = request.form.get('language')
    logger.debug(f"Received language parameter: {language}")
    if language and language not in SUPPORTED_LANGUAGES:
        logger.error(f"Invalid language: {language}")
        return jsonify({'error': f'Invalid language. Supported: {",
".join(SUPPORTED_LANGUAGES)}'}), 400
    strict_mode = request.form.get('strict_mode', 'false').lower() == 'true'
    logger.debug(f"Strict language mode: {strict_mode}")
    denoise = request.form.get('denoise', 'false').lower() == 'true'
    logger.debug(f"Denoise enabled: {denoise}")
    filename = secure_filename(file.filename)
    file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
    temp_wav = None
    try:
        file.save(file_path)
        logger.debug(f"Saved file to {file_path}")
        with open(file_path, 'rb') as audio_file:
            audio_file_id = fs.put(audio_file, filename=filename)
        audio_path = file_path
        if denoise and denoise_model:
            audio_path, _ = denoise_audio(file_path)
            logger.debug(f"Using denoised audio: {audio_path}")
        temp_wav = load_audio(audio_path)
        initial_prompt = LANGUAGE_PROMPTS.get(language, "") if language else
""

        transcription_options =
            { "beam_size": 10,
            "temperature": 0.0,
            "best_of": 5, "task":
            "transcribe",
            "initial_prompt": initial_prompt
        }
        if language:
            transcription_options["language"] = language
            if strict_mode:
                transcription_options["suppress_language_detection"] = True
        segments, info = model.transcribe(temp_wav, **transcription_options)
        detected_language = info.language if language is None else language
        logger.debug(f"Detected language: {detected_language} (probability:
{info.language_probability:.2f})")
        transcription_segments = []
        full_text = ""
        for segment in segments:
            full_text += segment.text
            transcription_segments.append({ "
                text": segment.text, "start":
                segment.start, "end":
                segment.end
            })
        full_text = full_text.strip()
        logger.debug(f"Transcription: {full_text}")
        warnings = []
        script_valid = validate_script(full_text, detected_language)
        if not script_valid:
            warnings.append(f"Transcription script does not match detected
language {detected_language}")
            logger.warning(f"Script mismatch: language={detected_language},
transcription={full_text}")
            fallback_lang = 'te' if detected_language == 'ml' else 'ml'
            if fallback_lang in SUPPORTED_LANGUAGES:
                logger.debug(f"Retrying transcription with fallback language:
{fallback_lang}")
                transcription_options["language"] = fallback_lang
                segments, info = model.transcribe(temp_wav,
**transcription_options)
                full_text = "".join(segment.text for segment in
segments).strip()
                detected_language = fallback_lang
                transcription_segments = [{"text": segment.text, "start":
segment.start, "end": segment.end} for segment in segments]
                logger.debug(f"Fallback transcription: {full_text}")
                if validate_script(full_text, detected_language):
                    warnings.append(f"Fallback to {fallback_lang} succeeded")
                else:
                    warnings.append(f"Fallback to {fallback_lang} still
produced incorrect script")
        if check_transcription_quality(full_text):
            warnings.append("Transcription contains repetitive characters,
possibly due to audio noise")
        try:
            sentiment_result = analyze_sentiment(full_text, 
detected_language)
            logger.debug(f"Sentiment result: {sentiment_result}")
        except Exception as e:
            logger.error(f"Sentiment analysis failed: {str(e)}")
            sentiment_result = {"label": "unknown", "score": 0.0}
            warnings.append(f"Sentiment analysis failed: {str(e)}")
        transcription_file = f"transcription_{filename}.txt"
        transcription_path = os.path.join(app.config['UPLOAD_FOLDER'], 
transcription_file)
        with open(transcription_path, 'w', encoding='utf-8') as f:
            f.write(full_text)
        with open(transcription_path, 'rb') as trans_file:
            transcription_file_id = fs.put(trans_file, 
filename=transcription_file)
        response = {
            'transcription_file_id': str(transcription_file_id),
            'language': detected_language,
            'language_probability': float(info.language_probability),
            'segments': transcription_segments,
            'sentiment': sentiment_result,
            'audio_file_id': str(audio_file_id)
        }
        if warnings:
            response['warnings'] = warnings
        if denoise and denoise_model:
            response['denoised_file'] = os.path.basename(audio_path)
        document = {
            'uploaded_date': datetime.utcnow(),
            'language': detected_language,
            'transcription_file_id': str(transcription_file_id),
            'audio_file_id': str(audio_file_id),
            'response': response
        }
        transcriptions_collection.insert_one(document)
        return jsonify(response), 200
    except Exception as e:
        logger.error(f"Error processing audio: {str(e)}")
        return jsonify({'error': f'Error processing audio: {str(e)}'}), 500
    finally:
        if os.path.exists(file_path):
            try:
                os.remove(file_path)
                logger.debug(f"Removed uploaded file: {file_path}")
            except Exception as e:
                logger.error(f"Failed to remove file {file_path}: {str(e)}")
        if temp_wav and os.path.exists(temp_wav):
            try:
                os.remove(temp_wav)
                logger.debug(f"Removed temporary WAV: {temp_wav}")
            except Exception as e:
                logger.error(f"Failed to remove temporary WAV {temp_wav}:
{str(e)}")

        if denoise and denoise_model and audio_path != file_path and
os.path.exists(audio_path):
            try:
                os.remove(audio_path)
                logger.debug(f"Removed denoised file: {audio_path}")
            except Exception as e:
                logger.error(f"Failed to remove denoised file {audio_path}:
{str(e)}")

        if 'transcription_path' in locals() and
os.path.exists(transcription_path):
            try:
                os.remove(transcription_path)
                logger.debug(f"Removed transcription file:
{transcription_path}")
            except Exception as e:
                logger.error(f"Failed to remove transcription file
{transcription_path}: {str(e)}")

@app.route('/sarcasm', methods=['POST'])
def sarcasm_detection():
    logger.debug(f"Request headers: {request.headers}")
    logger.debug(f"Request JSON: {request.json}")
    if not request.is_json:
        logger.error("No JSON data provided")
        return jsonify({'error': 'No JSON data provided'}), 400
    data = request.get_json()
    text = data.get('text')
    if not text or not isinstance(text, str):
        logger.error("Invalid or missing 'text' field")
        return jsonify({'error': "Invalid or missing 'text' field"}), 400
    try:
        result = detect_sarcasm(text)
        if 'error' in result:
            return jsonify({'error': result['error']}), 500
        return jsonify({'sarcasm': result}), 200
    except Exception as e:
        logger.error(f"Error processing sarcasm detection: {str(e)}")
        return jsonify({'error': f'Error processing sarcasm detection:
{str(e)}'}), 500

@app.route('/transcriptions', methods=['GET'])
def get_transcriptions():
```

```
try:
        transcriptions =
list(transcriptions_collection.find().sort('uploaded_date', -1))
        for transcription in transcriptions:
            transcription['_id'] = str(transcription['_id'])
            transcription['uploaded_date'] =
transcription['uploaded_date'].isoformat()
        return jsonify(transcriptions), 200
    except Exception as e:
        logger.error(f"Error retrieving transcriptions: {str(e)}")
        return jsonify({'error': f'Error retrieving transcriptions:
{str(e)}'}), 500

@app.route('/transcription/<file_id>', methods=['GET'])
def get_transcription_file(file_id):
    try:
        file_data = fs.get(file_id)
        return send_file(
            file_data,
            mimetype='text/plain',
            as_attachment=True,
            download_name=file_data.filename
        )
    except Exception as e:
        logger.error(f"Error retrieving transcription file {file_id}:
{str(e)}")
        return jsonify({'error': f'Error retrieving transcription file:
{str(e)}'}), 500

@app.route('/audio/<file_id>', methods=['GET'])
def get_audio_file(file_id):
    try:
        file_data = fs.get(file_id)
        return send_file(
            file_data,
            mimetype='audio/' + file_data.filename.rsplit('.', 1)[1].lower(),
            as_attachment=True,
            download_name=file_data.filename
        )
    except Exception as e:
        logger.error(f"Error retrieving audio file {file_id}: {str(e)}")
        return jsonify({'error': f'Error retrieving audio file: {str(e)}'}),
500

@app.route('/health', methods=['GET'])
def health_check():
    return
        jsonify({ 'status':
        'ok',
        'model': 'large-v3',
        'denoise_model': 'DeepFilterNet' if denoise_model else 'None'
    }), 200

if __name___== '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
```

APPENDIX-C

ENCLOSURES



ISSN: 2582-3930
Impact Factor: 8.586

INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING & MANAGEMENT
An Open Access Scholarly Journal || Index in major Databases & Metadata

CERTIFICATE OF PUBLICATION

International Journal of Scientific Research in Engineering & Management is hereby awarding this certificate to

Prakash Singh

in recognition to the publication of paper titled

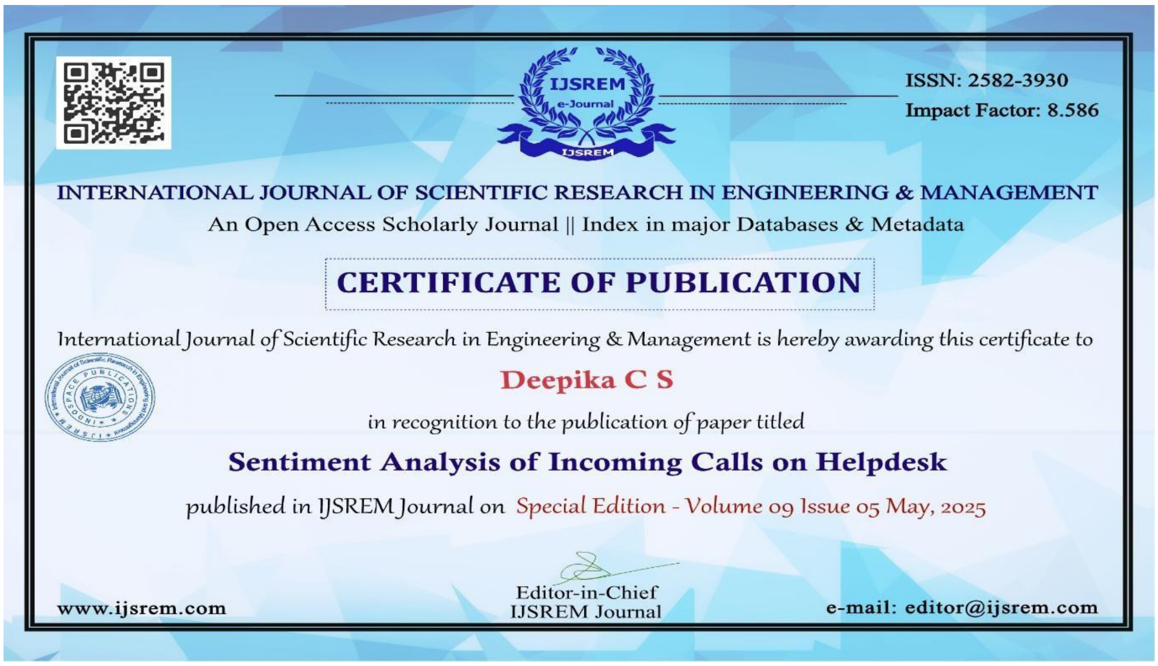
Sentiment Analysis of Incoming Calls on Helpdesk

published in IJSREM Journal on Special Edition - Volume 09 Issue 05 May, 2025

www.ijsrem.com

Editor-in-Chief
IJSREM Journal

e-mail: editor@ijsrem.com



ISSN: 2582-3930
Impact Factor: 8.586

INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING & MANAGEMENT
An Open Access Scholarly Journal || Index in major Databases & Metadata

CERTIFICATE OF PUBLICATION

International Journal of Scientific Research in Engineering & Management is hereby awarding this certificate to

Deepika C S

in recognition to the publication of paper titled

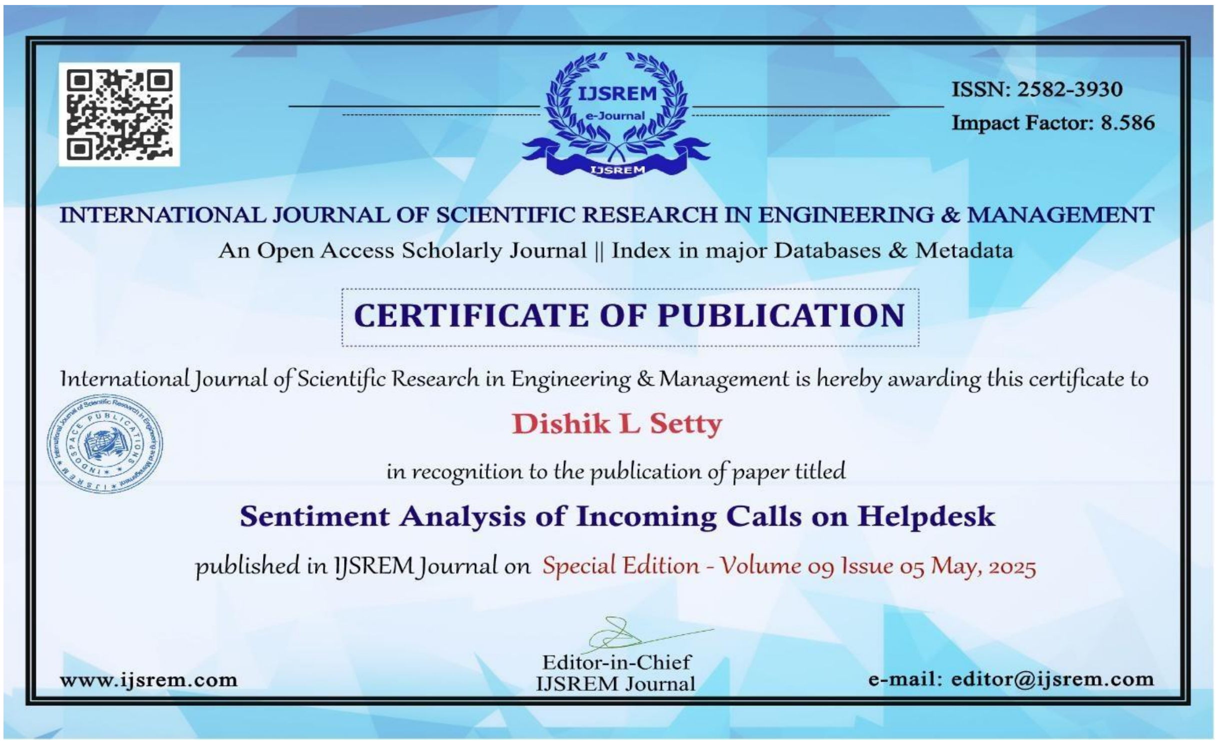
Sentiment Analysis of Incoming Calls on Helpdesk

published in IJSREM Journal on Special Edition - Volume 09 Issue 05 May, 2025

www.ijsrem.com

Editor-in-Chief
IJSREM Journal

e-mail: editor@ijsrem.com



ISSN: 2582-3930
Impact Factor: 8.586

INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING & MANAGEMENT
An Open Access Scholarly Journal || Index in major Databases & Metadata

CERTIFICATE OF PUBLICATION

International Journal of Scientific Research in Engineering & Management is hereby awarding this certificate to

Dishik L Setty

in recognition to the publication of paper titled

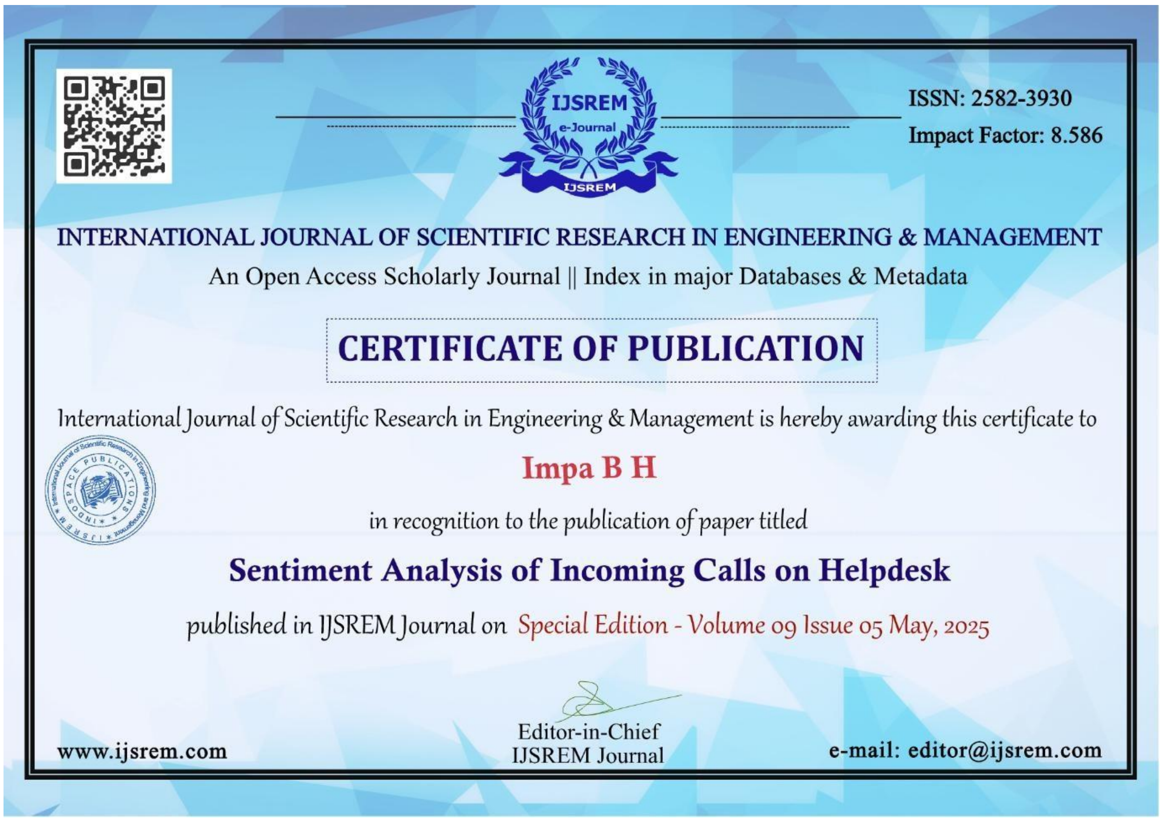
Sentiment Analysis of Incoming Calls on Helpdesk

published in IJSREM Journal on Special Edition - Volume 09 Issue 05 May, 2025

Editor-in-Chief
IJSREM Journal

www.ijsrem.com

e-mail: editor@ijsrem.com



ISSN: 2582-3930
Impact Factor: 8.586

INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING & MANAGEMENT
An Open Access Scholarly Journal || Index in major Databases & Metadata

CERTIFICATE OF PUBLICATION

International Journal of Scientific Research in Engineering & Management is hereby awarding this certificate to

Impa B H

in recognition to the publication of paper titled

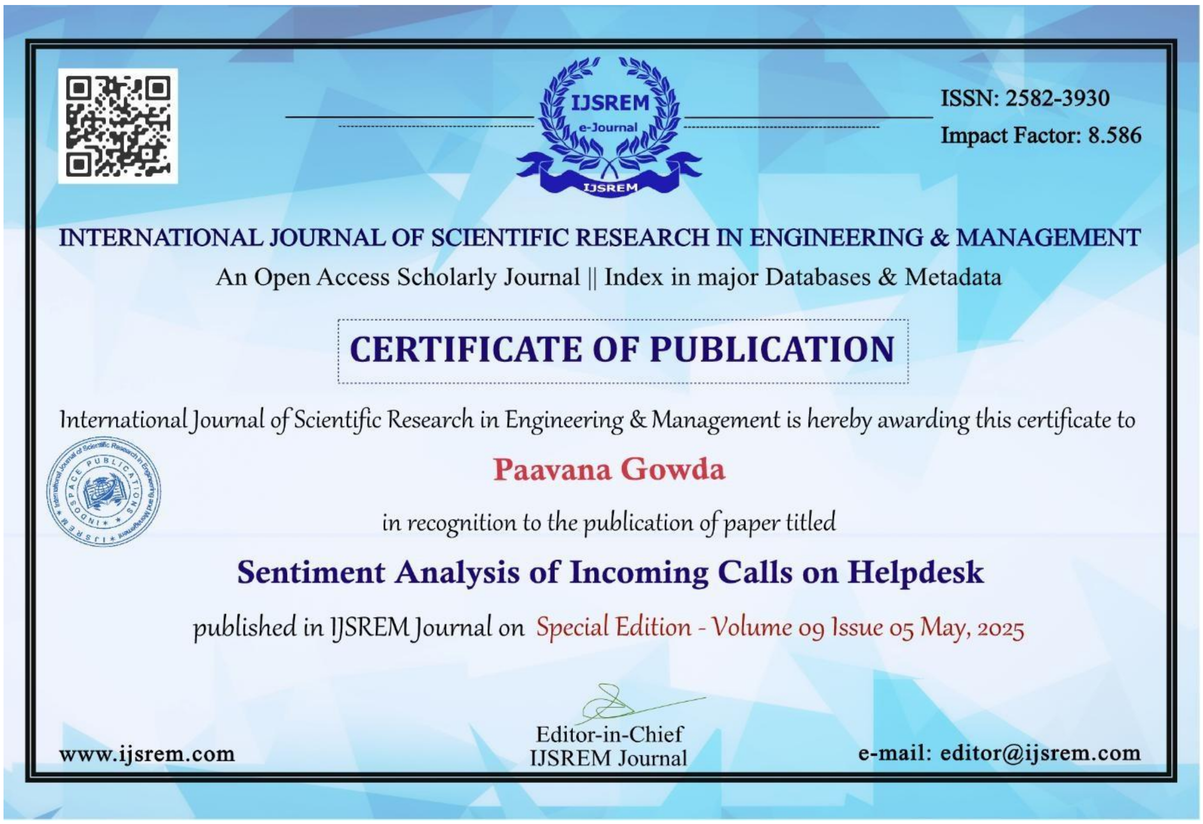
Sentiment Analysis of Incoming Calls on Helpdesk

published in IJSREM Journal on Special Edition - Volume 09 Issue 05 May, 2025

Editor-in-Chief
IJSREM Journal

www.ijsrem.com

e-mail: editor@ijsrem.com



IJSREM e-Journal IJSREM

ISSN: 2582-3930
Impact Factor: 8.586

INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING & MANAGEMENT

An Open Access Scholarly Journal || Index in major Databases & Metadata

CERTIFICATE OF PUBLICATION

International Journal of Scientific Research in Engineering & Management is hereby awarding this certificate to

Paavana Gowda

in recognition to the publication of paper titled

Sentiment Analysis of Incoming Calls on Helpdesk

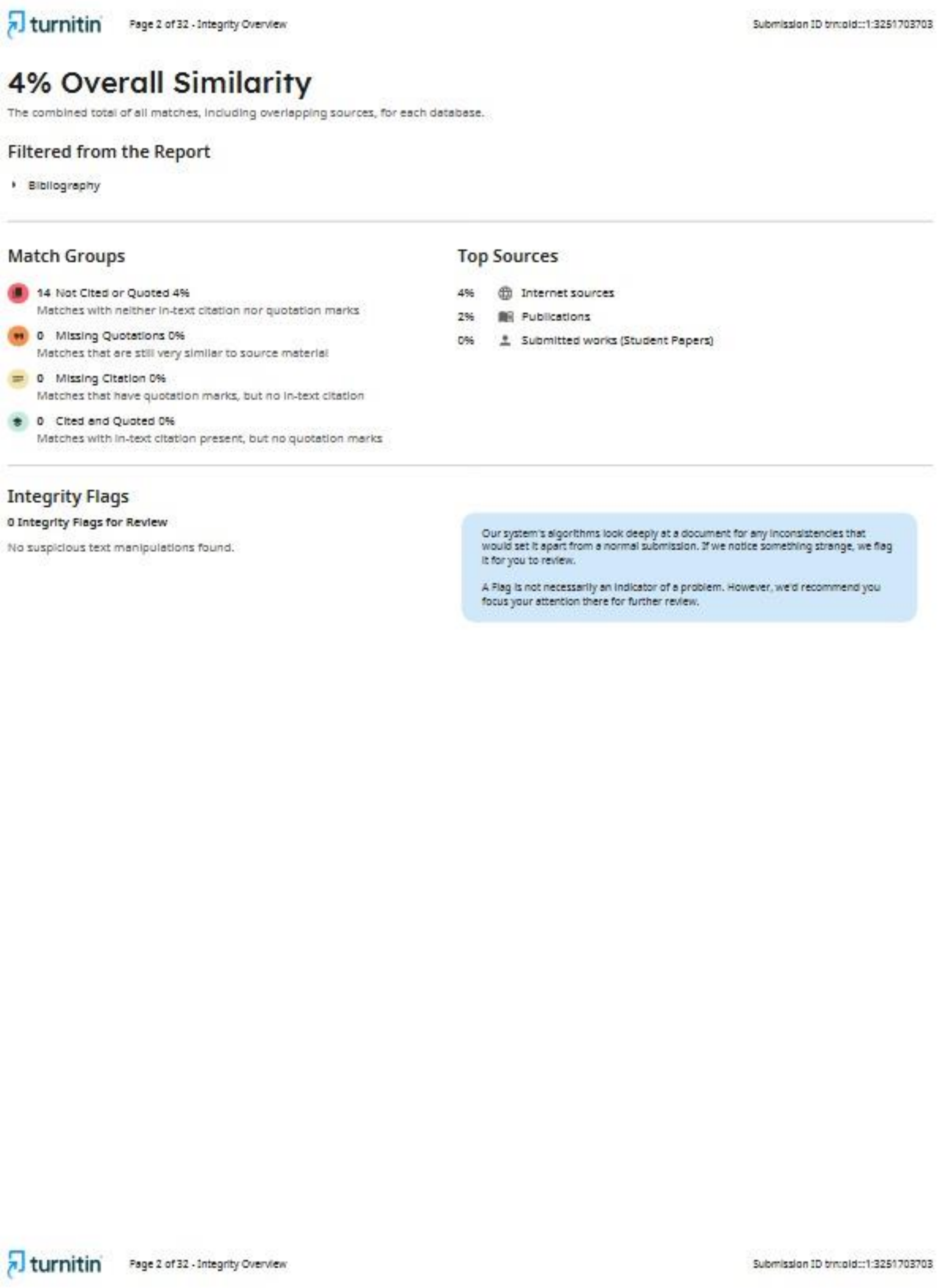
published in IJSREM Journal on Special Edition - Volume 09 Issue 05 May, 2025

Editor-in-Chief
IJSREM Journal

www.ijsrem.com

e-mail: editor@ijsrem.com

PLAGIARISM REPORT



turnitin Page 2 of 32 - Integrity Overview Submission ID trn:oid:::1:3251703703

4% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography

Match Groups

🟥 14 Not Cited or Quoted 4%
Matches with neither in-text citation nor quotation marks

● 0 Missing Quotations 0%
Matches that are still very similar to source material

● 0 Missing Citation 0%
Matches that have quotation marks, but no in-text citation

● 0 Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

4%	Internet sources
2%	Publications
0%	Submitted works (Student Papers)

Integrity Flags

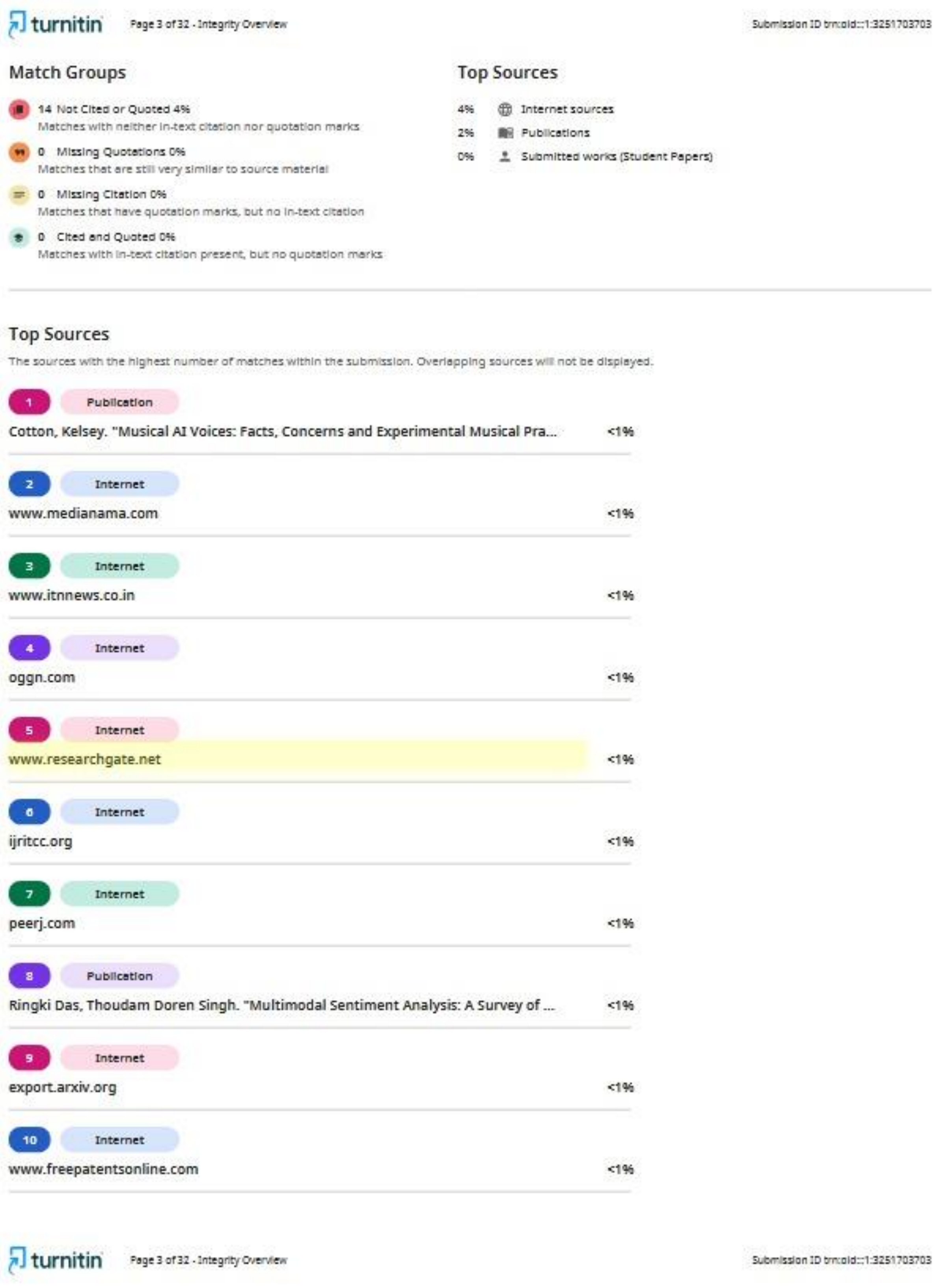
0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

turnitin Page 2 of 32 - Integrity Overview Submission ID trn:oid:::1:3251703703



turnitin Page 3 of 32 - Integrity Overview Submission ID trn:oid:::1:3251703703

Match Groups

- 14 Not Cited or Quoted 4%
  Matches with neither in-text citation nor quotation marks
- 0 Missing Quotations 0%
  Matches that are still very similar to source material
- 0 Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

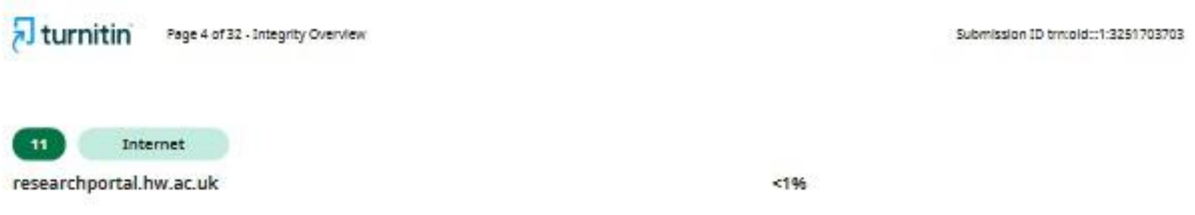
4%	Internet sources
2%	Publications
0%	Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Publication	Cotton, Kelsey. "Musical AI Voices: Facts, Concerns and Experimental Musical Pra...	<1%
2	Internet	www.medianama.com	<1%
3	Internet	www.itnnews.co.in	<1%
4	Internet	oggn.com	<1%
5	Internet	www.researchgate.net	<1%
6	Internet	ijritcc.org	<1%
7	Internet	peerj.com	<1%
8	Publication	Ringki Das, Thoudam Doren Singh. "Multimodal Sentiment Analysis: A Survey of ...	<1%
9	Internet	export.arxiv.org	<1%
10	Internet	www.freepatentsonline.com	<1%

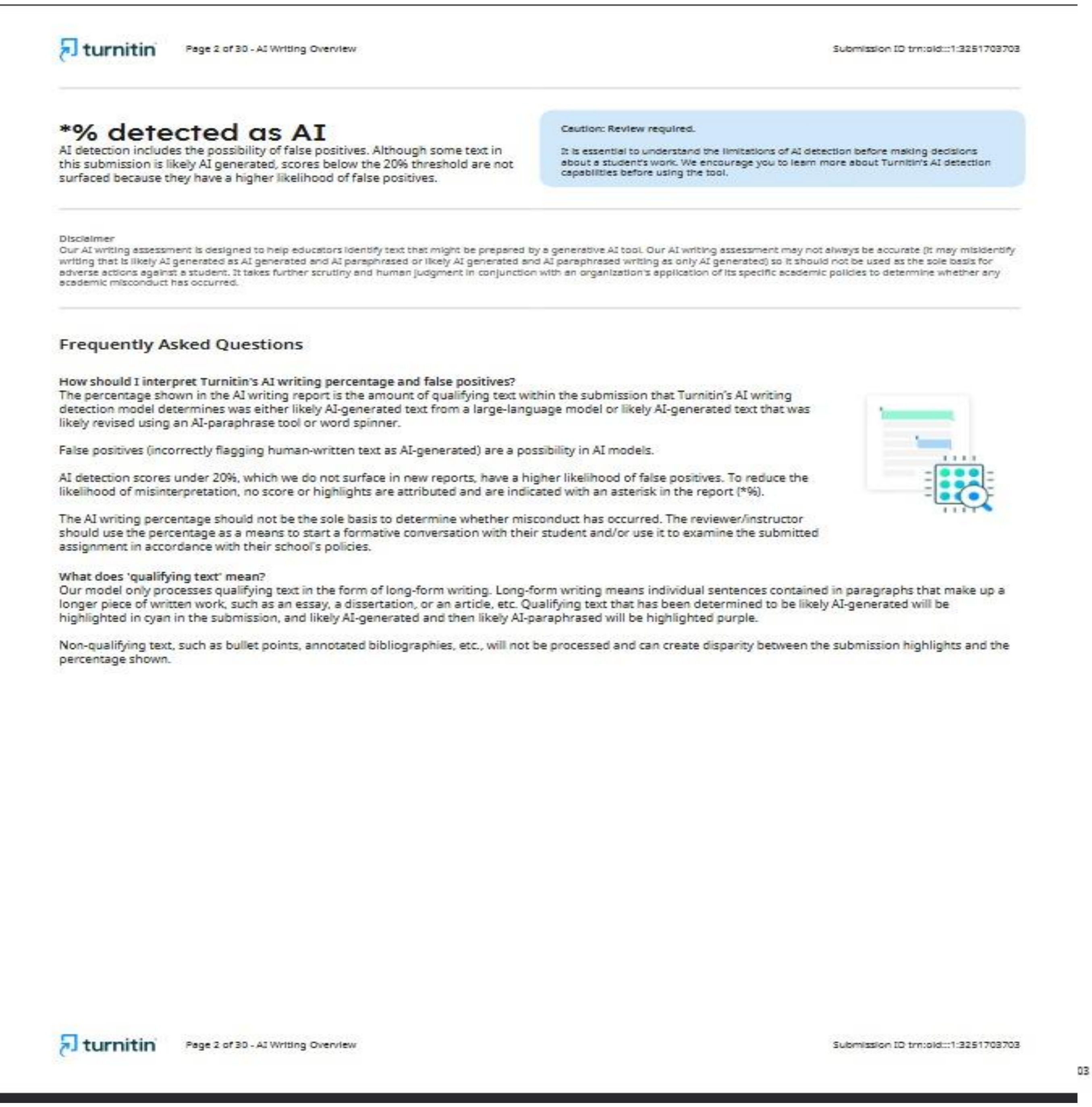
turnitin Page 3 of 32 - Integrity Overview Submission ID trn:oid:::1:3251703703



turnitin Page 4 of 32 - Integrity Overview Submission ID trn:oid:::1:3251703703

11 Internet

researchportal.hw.ac.uk <1%



turnitin Page 2 of 30 - AI Writing Overview Submission ID trn:oid:::1:3251703703

***% detected as AI**

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Disclaimer
Our AI writing assessment is designed to help educators identify text that might be prepared by a generative AI tool. Our AI writing assessment may not always be accurate (it may misidentify writing that is likely AI generated as AI generated and AI paraphrased or likely AI generated and AI paraphrased writing as only AI generated) so it should not be used as the sole basis for adverse actions against a student. It takes further scrutiny and human judgment in conjunction with an organization's application of its specific academic policies to determine whether any academic misconduct has occurred.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.

turnitin Page 2 of 30 - AI Writing Overview Submission ID trn:oid:::1:3251703703

03

SUSTAINABLE DEVELOPMENT GOALS

SUSTAINABLE DEVELOPMENT GOALS
17 GOALS TO TRANSFORM OUR WORLD
1 NO POVERTY
2 ZERO HUNGER
3 GOOD HEALTH AND WELL-BEING
4 QUALITY EDUCATION
5 GENDER EQUALITY
6 CLEAN WATER AND SANITATION
7 AFFORDABLE AND CLEAN ENERGY
8 DECENT WORK AND ECONOMIC GROWTH
9 INDUSTRY, INNOVATION AND INFRASTRUCTURE
10 REDUCED INEQUALITIES
11 SUSTAINABLE CITIES AND COMMUNITIES
12 RESPONSIBLE CONSUMPTION AND PRODUCTION
13 CLIMATE ACTION
14 LIFE BELOW WATER
15 LIFE ON LAND
16 PEACE, JUSTICE AND STRONG INSTITUTIONS
17 PARTNERSHIPS FOR THE GOALS
SUSTAINABLE DEVELOPMENT GOALS

The Project Work Carried out here is mapped to SDG-04 Quality Education. The provided Arduino project illustrates a simple yet effective system that monitors voltage from a solar panel and light intensity using an LDR (light-dependent resistor) to control a relay connected to a light source. This design, paired with a LiquidCrystal_I2C display for real-time data feedback, has potential applications that align with themes of inclusive growth, accessibility, and sustainability.