

Cortex: Adaptive Multi-Channel Communication Assistant

A major project report submitted in partial fulfilment of the requirement for
the award of degree of

Bachelor of Technology
in
Computer Science & Engineering

Submitted by
**ARYAMAN TIWARI (211130), VIVEK KUMAR KATARIA
(211377), YASH SINGH (211446)**

Under the guidance & supervision of
DR. AMAN SHARMA



**Department of Computer Science & Engineering and
Information Technology**
**Jaypee University of Information Technology, Wagnaghat,
Solan - 173234 (India)**
May 2025

CANDIDATE'S DECLARATION

This is to certify that the major project report entitled '**Cortex: Adaptive Multi Channel Communication Assistant**', submitted in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science Engineering**, in the Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat, is a bonafide project work carried out under my supervision during the period from July 2024 to May 2025.

I have personally supervised the research work and confirm that it meets the standards required for submission. The project work has been conducted in accordance with ethical guidelines, and the matter embodied in the report has not been submitted elsewhere for the award of any other degree or diploma

Name: Aryaman Tiwari

Name: Vivek Kumar Kataria

Name: Yash Singh

Roll No.: 211130

Roll No.: 211377

Roll No.: 211446

Date:

Date:

Date:

This is to certify that the above statement made by the candidates is true to the best of my knowledge

Supervisor Name: Dr. Aman Sharma

Designation: Assistant Professor (SG)

Department: Computer Science & Engineering

Date:

SUPERVISOR'S CERTIFICATE

This is to certify that the major project report entitled '**Cortex: Adaptive Multi Channel Communication Assistant**', submitted in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science & Engineering**, in the Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat, is a bona fide project work carried out under my supervision during the period from July 2024 to May 2025.

I have personally supervised the research work and confirm that it meets the standards required for submission. The project work has been conducted in accordance with ethical guidelines, and the matter embodied in the report has not been submitted elsewhere for the award of any other degree or diploma.

Date:

Place: JUIT

Supervisor Name: Dr. Aman Sharma

Designation: Assistant Professor(SG)

Department: Dept. of CSE & IT

ACKNOWLEDGEMENT

We would like to express our deepest gratitude to everyone who contributed to the successful completion of our major project, '**Cortex: Adaptive Multi Channel Communication Assistant**'.

First and foremost, we are thankful to our esteemed institution and **Dr. Aman Sharma** for providing us with the guidance, resources, and encouragement needed to pursue this innovative endeavour. We are particularly indebted to our project guide, Dr. Anita Sardana, whose expertise, mentorship, and continuous support were invaluable throughout the project.

We extend our sincere thanks to our classmates, friends, and family for their unwavering support, patience, and encouragement during the development of this project. Their belief in our vision motivated us to overcome challenges and deliver a meaningful solution.

Lastly, we are grateful to all the researchers and developers in the field of artificial intelligence, biometric systems, and software development whose work inspired and guided our project. This project is a testament to the collective efforts and shared vision of improving the lives of missing individuals and their families through technology.

We hope our project contributes positively to this cause and inspires further advancements in this field.

Aryaman Tiwari (211130) Vivek Kumar Kataria (211377) Yash Singh (211446)

TABLE OF CONTENTS

CERTIFICATE.....	(i)
DECLARATION.....	(ii)
ACKNOWLEDGEMENT.....	(iii)
LIST OF FIGURES.....	(iv)
LIST OF TABLES.....	(v)
LIST OF SYMBOLS.....	(vii-viii)
ABSTRACT.....	(vii)
1. INTRODUCTION.....	(1-7)
1.1 INTRODUCTION.....	(1-2)
1.2 PROBLEM STATEMENT.....	(2)
1.3 OBJECTIVES.....	(3)
1.4 SIGNIFICANCE AND MOTIVATION OF THE PROJECT WORK.....	(4-5)
1.5 ORGANIZATION OF PROJECT REPORT.....	(6-7)
2. LITERATURE SURVEY.....	(8-21)
2.1 OVERVIEW OF RELEVANT LITERATURE.....	(8-19)
2.2 KEY GAPS IN THE LITERATURE.....	(19-21)
3. SYSTEM DEVELOPMENT.....	(22-44)
3.1 REQUIREMENTS AND ANALYSIS.....	(22-27)
3.2 SYSTEM ARCHITECTURE.....	(27-28)
3.3 PROJECT DESIGN AND ARCHITECTURE.....	(28-34)
3.4 IMPLEMENTATION.....	(34-40)
3.5 KEY CHALLENGES.....	(40-42)
4. TESTING.....	(43-49)
4.1 TESTING STRATEGY.....	(43-45)
4.2 TEST CASES AND OUTCOMES.....	(45-49)
5. RESULTS AND EVALUATION.....	(50-53)
6. CONCLUSIONS AND FUTURE SCOPE.....	(54-56)
6.1 CONCLUSION.....	(54-55)
6.2 FUTURE SCOPE.....	(55-56)
REFERENCES.....	(57-58)

LIST OF FIGURES

Figure Number	Figure Title	Page Number
Figure 3.1	Project Design Of The System	30
Figure 3.2	Architecture of Cortex	36
Figure 3.3	Code to Start the Cortex	37
Figure 3.4	Initializing the Groq LLM	38
Figure 3.5	Intialize the System and Human message	38
Figure 3.6	Vector Base Creation	39
Figure 3.7	State defines with help of LangGraph	40
Figure 3.8	Advance Protocol Define	40
Figure 3.9	Wavy Algorithm	40
Figure 3.10	Response Generation	41
Figure 3.11	Saving History and Previous Query	42
Figure 4.1	Enter the Input Details	48
Figure 4.2	Enter The API Key	48
Figure 4.3	Initialize the Sentiment Analysis	49
Figure 4.4	Final Output	50
Figure 5.1	Response Accuracy	52
Figure 5.2	Speed of Inference	53
Figure 5.3	Context Retention Score	53
Figure 5.4	Token Processing Speed	54

LIST OF TABLES

Table Number	Description	Page Number
2.1	Literature Survey r	9-11
3.1	Table of Hardware Requirement	22-23
3.2	Table of Software Requirement	23-24

LIST OF ABBREVIATIONS, SYMBOLS OR NOMENCLATURE

Abbreviations	Full Form
LLM	Large Language Model
LPU	Language Proceesing Unit
UI	User Interface
API	Application Programming Interface
GPT	Generative Pre-Trained Transformer
AWS	Amazon Web Service

ABSTRACT

Today's startups frequently have to decide whether to increase customer communication or hire more staff, which isn't always possible. They require intelligent systems that can manage client-customer interactions without exhausting the team because of their tight budgets and fast-paced work environments. Cortex comes into play here. Cortex, which is based on well-trained language models, is made to handle email and message responses in a way that is accurate, quick, and feels personalized. Businesses don't have to spend hours crafting the ideal response; instead, they can rely on Cortex to read the sentiment, comprehend the intent, and send out customized responses. It's particularly helpful for startups where every interaction matters. By using Langchain as a wrapper, Cortex can assess emotional tone and produce context-appropriate responses while remaining professional and effective.

The concept is straightforward: more meaningful communication with less manual labor. The power of the system is not found in a single tool, but rather in the collaboration of multiple intelligent technologies. By storing context-rich data that the LLMs can access later, vector databases make sure that each response is based on the history of the conversation. The system now sounds less generic and more relevant thanks to the integration of fine-tuned LLMs that have been adjusted to industry-specific terminology. Additionally, Langgraph aids in conversation flow management by recalling previous statements, which is essential in longer, multi-turn discussions. Conversely, Langsmith operates more covertly but is equally crucial; it aids in tracking bugs, keeping an eye on performance, and providing feedback on how various models behave. These components work together to produce a configuration that not only reacts well but also improves with time. It feels more like a developing assistant designed to comprehend the subtleties of business communication than a tool.

It appears less like a tool and more like a developing assistant meant to understand the complexities of corporate communication. WAVY, a fast, efficient in-house developed algorithm, is the final component of our strategy. It promotes more evolution of sentiment analysis by changing how the system processes incoming messages and creates outgoing ones. For example, if a customer writes in rage, WAVY directs the algorithm to choose a more sympathetic, softer tone. Apart from polishing, it also imparts personality. Our last goal is to build a system that connects rather than only offers solutions

CHAPTER 01

INTRODUCTION

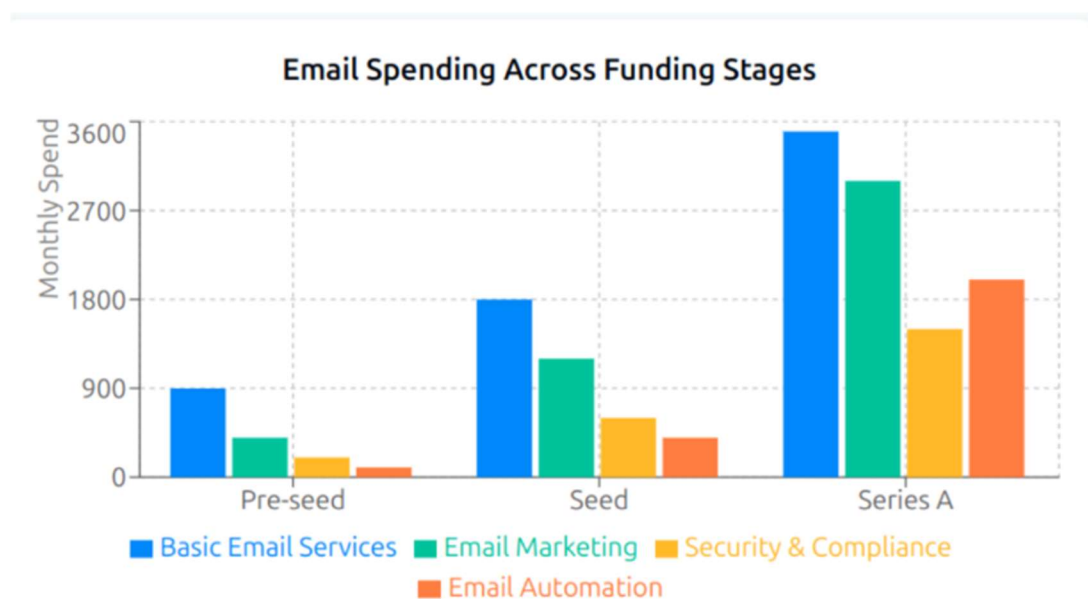
1.1 INTRODUCTION

Businesses wishing to grow in the fast-paced digital world of today must provide customers rapid, efficient, and specialized solutions. While traditional chatbots are frequently limited to fast and straightforward inquiries, actual user interactions—especially via emails—contain extensive, token-rich material indicating a spectrum of emotions, intentions, and moods. These communications could be inquiries all needing various tone and context knowledge, comments, complaints, or product reviews. Standard systems usually lack real-time, nuanced replies that match the intricacy of user input. By means of sentiment analysis, multi-channel integration, and sophisticated natural language processing, our project, Cortex: Adaptive Multi-Channel Communication Assistant, aims to close this widening communication gap and change user interaction. By way of this, Cortex enables companies to automate smart replies, so guaranteeing that users receive accurate and emotionally pertinent. Cortex basically aims to close the gap between user expectations and fast, high-quality help.

Strong natural language processing tools like BERT help us to grasp context and user emotion and let the system dynamically customize responses. Cortex allows for smooth interaction across all platforms, including integrated chat systems, corporate email tools, and Gmail. While Langsmith keeps an eye on system behavior and improves the traceability of model responses, Langgraph tracks ongoing dialogue to guarantee context continuity and conversation memory. The adaptive power of the model is further enhanced by the combination of vector databases, specially trained LLMs, and our innovative algorithm WAVY. To make sure that responses are prompt, individualized, and emotionally relevant, WAVY uses sentiment detection and contextual embedding to improve both the processing pipeline and the output. When combined, these technologies transform Cortex from a simple communication tool into a learning assistant that learns from each interaction and decreases manual intervention while increasing client satisfaction

From a business standpoint, Cortex tackles both financial and functional pain points, particularly in early-stage startups. An increasing operational cost is the cost of communication, especially in email-based systems. According to market data, Google Workspace has the largest adoption rate of email services (52%), followed by Microsoft 365 (24%). From \$1,600 per month at the pre-seed stage to \$10,000 by Series A, startups' email budgets grow as they move through funding stages, with marketing and automation tools seeing a 20x increase. According to a thorough analysis, 35% of email spending is allocated to internal communications, 25% to customer service, 20% to campaigns, and the remaining portion to vendor and investor relations. By decreasing reliance on manual support teams, Cortex provides a competitive edge, lowering operating expenses and improving communication effectiveness. This means investing in technology that learns, adapts, and creates value through intelligent automation so that startups managing early-stage growth can scale smarter, not harder.

Figure 1.1 - Figures of existing use of email in the workplace



1.2- Figures showing cost per email



1.2 PROBLEM STATEMENT

The foundation of success for startups and small enterprises is efficient communication, yet these companies struggle greatly with managing interactions across many different communication channels. Often, limited resources cause tardy and uneven replies, which frustrate customers and compromise confidence. Furthermore, many current communication tools are either too complicated or too costly, which makes them unavailable for startups running on limited funds.

The absence of integration across platforms, which causes broken processes and lowers efficiency, is another important problem. The lack of tools able to deliver emotionally aware, real-time replies makes this issue even more acute. Startups want reasonably priced, smart solutions that fit their changing needs and seamlessly integrate across platforms.

Our work solves these problems by automating and customizing consumer interactions, providing emotionally intelligent replies, and guaranteeing consistency across several platforms. Through simplifying communication processes, we hope to improve customer happiness, foster confidence, and help new businesses concentrate on their main business operations free from operational inefficiency.

Technology is so diversified these days that selecting the appropriate tech stack and tools may dynamically enable business scale at a faster rate. Many of the focus areas include: employing Best Machine Learning Technique, System Initialization, Query Processing and Sentiment Analysis, FAQ Retrieval and Vector Search, Response Generation..etc.

Running 50 queries over several LLMs helped us to focus our choice to three models: GPT 3.5[18], Gemini 1.5 Flash[17], and Groq[16]. Groq divides the work into independent, specialized modules.

1.3 OBJECTIVES

The objective of this project is to design a communication platform that addresses the following key goals:

1. **IMPROVING MULTI-CHANNEL COMMUNICATION:** The platform will give startups an all-in-one integrated tool combining social media communications, chat, and email. This will provide a smooth experience for companies and consumers by removing the inefficiencies of running different tools.
2. **CUSTOMER INTERACTION PERSONALIZATION:** The system will provide contextually correct, tailored responses by using adaptive learning algorithms. Every interaction will seem relevant and meaningful, therefore fostering customer confidence and loyalty.
3. **MAINTAINING SCALABILITY AND CONSISTENCY:** Across all communications, the platform will keep a steady tone and quality. Its scalable design will fit the expanding demands of new businesses, so guaranteeing their competitiveness in changing market.
4. **ENABLING DATA-DRIVEN DECISIONS AND INSIGHTS:** The platform will offer reporting tools and real-time analytics to monitor customer behavior, communication patterns, and response efficacy. These findings will enable new businesses to choose wisely, maximize their communication plans, and raise general consumer happiness.

1.4 SIGNIFICANCE AND MOTIVATION OF THE PROJECT WORK

1. **Enabling smart communication:** In the growing landscape of early-stage startups, smart and scalable communication is vitally crucial. This initiative offers a solution to replacing the requirement for big customer support personnel by including an AI-powered assistant capable of swiftly, accurately, and empathically managing multi-channel discussions.
2. **Reducing operational costs:** Startups might find themselves on a limited budget, particularly in the early funding stages. Cortex is a reasonably affordable solution that lets businesses expand without corresponding cost increases by automating email and chat responses, hence significantly lowering the requirement for dedicated support crew personnel.
3. **Applying sentiment driven responses:** Unlike traditional bots, Cortex generates more relevant and emotionally aware replies using advanced context tracking and sentiment analysis. This ability to understand the tone and context of user input improves consumer happiness by means of more human and important interactions.
4. **Meeting demands and trends of the market:** Given increasing startup funding in email communication and automation technology, our project complements actual industry trends. Cortex meets a growing need for smart automation that goes beyond pre-programmed scripts and offers adaptive, real-time responses.
5. **Allowing data-driven expansion:** By way of integrations with tools such LangChain, LangGraph, and LangSmith as well as custom algorithms such WAVY, the project leverages data insights to continuously improve system behavior. This creates a learning loop that allows businesses to enhance communication strategies while maintaining high efficiency and individualization.

1.5 ORGANIZATION OF PROJECT REPORT

This report is organised majorly into 6 sections and each section provides a detailed description about the project.

1.5.1 CHAPTER 1: INTRODUCTION

The introduction chapter outlines the objectives, extent, motivation, and modern startup ecosystem relevance of the project. It sets forth the communication issues experienced by expanding companies and explains how Cortex intends to address them with NLP-driven solutions and artificial intelligence.

1.5.2 CHAPTER TWO: LITERATURE RESEARCH

Foundational technologies and ideas including Natural Language Processing, sentiment analysis, vector databases, and multi-LLM systems are explored in this chapter. It comprises a study of current communication systems and scholarly work influencing Cortex's architecture.

1.5.3 CHAPTER THREE: SYSTEM DEVELOPMENT

Chapter 3 covers the technical evolution of Cortex as well as the tools and frameworks applied, including Langchain, Langgraph, and Langsmith. It also discusses the system architecture, API integrations, vector-based retrieval, the WAVY algorithm, and exhibits diagrams depicting response pipelines and workflow.

1.5.4 CHAPTER FOUR: TESTING

The testing part assesses how well Cortex performs in practical situations. It covers the testing methods used—such as precision in sentiment analysis, response accuracy, and user feedback loops. There are also benchmark comparisons between several LLMs to confirm performance.

1.5.5 CHAPTER FIVE: OUTCOMES AND ASSESSMENT

This chapter offers a comprehensive examination of the results produced by Cortex in various configurations. It covers system responsiveness across several communication channels, user satisfaction measures, BLEU score assessments, and comparisons with conventional chat systems.

1.5.6 CHAPTER VI: FUTURE SCOPE AND CONCLUSIONS

The last chapter emphasizes future development areas and summarizes the general accomplishments of the project. Among the recommendations are increasing multilingual support, adding voice communication, enhancing the WAVY algorithm, and deepening sentiment knowledge to strengthen model personalisation.

CHAPTER 02

LITERATURE SURVEY

2.1 OVERVIEW OF RELEVANT LITERATURE

2.1.1 INTRODUCTION

This section covers existing research related to Cortex, focusing on NLP techniques and LLMs[1]. The model categorization is based on both the sentiment of the text and vector inputs provided by the user. To generate accurate responses, some studies [12,13] use vector embeddings, converting input text into a small corpus and then mapping it to a 724-dimensional vector using techniques like Word2Vec or TF-IDF. For tokenization, we rely on Hugging Face embeddings, which come with built-in similarity search algorithms [15]. Traditional databases like SQL and Postgres are widely used, but when dealing with NLP and storing vectorized data, a Vector Search Database is essential [13]. To optimize retrieval efficiency, we integrate ChromaDB [19] for vector storage. A standard LLM generates responses using AI models, incorporating Reinforcement Learning with Human Feedback (RLHF)[7] and deep learning techniques built on transformer-based architectures. GPT (Generative Pre-trained Transformer) follows three main architectures: Encoder, Decoder, and Encoder-Decoder [1]. However, for our approach, we specifically use Groq LLM, which operates on a decoder-only architecture [16]—a concept we'll explore further in Section 4. A streamlined conversational model allows AI to process requests and generate responses while maintaining a limited interaction history, similar to natural human-AI conversations [3,4]. Many researchers focus on Retrieval-Augmented Generation (RAG)[15], which enhances response accuracy but is computationally intensive. One major issue identified in recent studies [15,19] is synchronization delays—RAG cannot proceed to the next step until the current process is completed, making it time-consuming. Several algorithms[12,13,14] improve response accuracy for FAQs, including Cosine Similarity, Manhattan Distance, and the widely used Euclidean Distance in ML applications. Additionally, the Top-K algorithm helps refine FAQ search results for more precise responses. Sentiment analysis[8,11] plays a crucial role in understanding text emotions. For instance, "I like it" is positive, while "I

hate it" is negative. Even though transformers can differentiate these, true sentiment comprehension remains a challenge. Here's where Groq LLM excels—it has been trained on vast token datasets, ensuring sentiment-aware responses. Through this work, we compile recent advancements for Cortex, identifying cutting-edge technologies and algorithms to classify text sentiment effectively. We introduce a technique called WAVY, designed for more efficient Cortex responses. Our analysis highlights the need for a hybrid approach—combining fixed databases with vector databases—to enhance alignment between clients and businesses [4]. The history of LLMs, how a corpus of text into a vector embedding are made and its numerous purposes but a The drawback is that it uses supervised training limited to a particular sector or a company. Overall, our optimized RAG model achieves strong performance with minimal computational overhead.

Table 2.1-Literature Survey

S. No.	Paper Title [Citation]	Journal/Conference (Year)	Tools/Techniques/Data set	Results	Limitations
1	Attention Is All You Need [1]	NeurIPS (2017)	Transformer, Self-Attention	Achieved SOTA in machine translation	High training cost, limited scalability for long sequences
2	Language Models are Unsupervised Multitask Learners [2]	OpenAI Technical Report (2019)	GPT-2, Unsupervised LM, Large-scale web text	Demonstrated multitask learning from raw text	Lack of interpretability and fine-grained control
3	Language Models are Few-Shot Learners [3]	arXiv (2020)	GPT-3, 175B parameters,	Competitive performance on NLP tasks	Extremely resource-intensive and costly

			Few-shot prompting	without fine-tuning	
4	Transformers: State-of-the-Art NLP [4]	EMNLP (2020)	HuggingFace, Pretrained Transformers	Democratized access to NLP models	May oversimplify custom use-cases
5	Groq: A Software-defined Tensor Streaming [5]	arXiv (2021)	Groq TSM Architecture	Enabled real-time AI workloads	Limited support for diverse ML frameworks
6	Multi-channel Communication and Collaboration [6]	IEEE Trans. Human-Mach. Syst. (2021)	AI automation in CRM systems	Improved efficiency and consistency	Generalization across industries remains a challenge
7	DRL for Dialogue Generation [7]	IEEE Trans. Affect. Comput. (2021)	Deep RL, Conversational AI	Better coherence in long-term dialogue	Sample inefficiency and instability in RL
8	LLMs for Consumer Review Analysis [8]	MICA, GMU Tech Report (2023)	Fine-tuning, LLMs, Domain-specific data	Improved sentiment analysis accuracy	Requires domain adaptation and tuning
9	NLP Techniques Survey [9]	IEEE Trans. Knowl. Data Eng. (2021)	Transformers, Contextual Embeddings	Summarized NLP advances	Lacks implementation-specific insights

10	Evaluating AI-Generated Emails [10]	World Journal of English Language (2024)	Qual/Quant Metrics, Email datasets	Boosted response rates and readability	Less personalization and emotional nuance
11	Aspect-based Sentiment Analysis Review [11]	Artificial Intelligence Review (2024)	Survey of ABSA algorithms	Mapped interrelations among datasets and tasks	Varying dataset quality and lack of standardization
12	Foundations of Vector Retrieval [12]	arXiv (2024)	Vector algorithms, ANN search	Covered core principles and advanced methods	Theoretical depth but lacks empirical benchmark
13	Vector Search for Enhanced Retrieval [13]	arXiv (2024)	Multi-vector, LLMs, Real-world datasets	Significant improvement in document retrieval	Computational cost for multi-vector models
14	Seq2Seq on Keywords for FAQ Retrieval [14]	arXiv (2021)	TF-IDF, Word2Vec, Keyword Mapping	~92% Precision@5, +13% vs baselines	Performance dependent on keyword quality

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention Is All You Need [1]: The paper "Attention Is All You Need" presents the Transformer architecture, a significant deep learning advancement for

natural language processing activities. Unlike conventional models that depended much on recurrence and convolutions, the Transformer processes input sequences in parallel using self-attention techniques. This new architecture allows the model to more effectively capture long-range dependencies, so significantly lowering training times and improving scalability. Particularly for the English-to-German and English-to-French translation projects, the researchers showed the model's effectiveness on machine translation criteria, attaining state-of-the-art performance. By allowing pretraining on vast text corpora, the architecture also opens the door for transfer learning. No recurrence means quicker calculations and better use of current technology. One drawback, though, is the computational complexity of self-attention, which can get costly for lengthy sequences. Setting the groundwork for future models like BERT and GPT, the paper has had a significant influence. All things considered, this study signaled the start of a paradigm change in NLP on how sequence modeling activities are handled.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language Models are Unsupervised Multitask Learners (GPT-2). OpenAI Technical Report[2]:

This technical paper presents GPT-2, a large-scale unsupervised language model trained on a varied internet dataset. Designed to forecast the next word in a sequence, GPT-2 uses the Transformer decoder architecture to learn intricate linguistic patterns without task-specific tuning. Comprising 1.5 billion components, its design allows it to carry out several NLP tasks—including translation, summarization, and question answering—using only raw text and few or no samples. The authors emphasize how the model shows good zero-shot performance, or ability to generalize to untrained tasks without extra training. Showcasing the strength of scale and unmonitored training, GPT-2 outperforms previous standards in numerous text-based competitions. The model does raise questions about abuse, though, such producing false information or harmful content, which causes OpenAI to postpone complete release at first. Although it shows amazing generality, its shortcomings include a lack of grounding in real information and a propensity to generate reasonable-sounding but erroneous or biased results. The publication of GPT-2 spurred more study in big language models and underlined the need of ethical AI creation and use.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., et al. (2020). Language Models are Few-Shot Learners (GPT-3). ArXiv[3]: While scaling the model considerably to 175 billion parameters, GPT-3 expands on the architecture and training philosophy of GPT-2.

The main contribution of the research is showing the few-shot learning potential of the model, whereby GPT-3 can complete tasks with just a few instances included in the input prompt. GPT-3, trained on a vast corpus taken from internet content, uses in-context learning instead of task-specific fine-tuning. This design lets it generalize over jobs including text production, question answering, and translation. The paper emphasizes how crucial size is for strong generalization. Though it stayed uneven on others, performance on benchmark NLP tasks in certain instances neared or surpassed state-of-the-art outcomes. Although the outcomes were revolutionary, the writers admitted some shortcomings: With little logical coherence or reasoning for complicated questions, GPT-3 can produce biased, harmful, or factually erroneous results. Moreover, the model's size creates significant environmental and computational expenses. Still, GPT-3 underlined the changing potential of large-scale language models and generated great excitement in rapid engineering and zero/few-shot paradigms in NLP.

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2020). Transformers: State-of-the-Art Natural Language Processing. EMNLP.[4]: This paper introduces the Hugging Face Transformers library, a popular open-source tool that simplifies access to Transformer-based natural language processing models, is introduced in this article. The authors stress democratizing artificial intelligence research and applications by providing pre-trained models, straightforward APIs, and integration with popular frameworks like PyTorch and TensorFlow. The library supports a number of models, including T5, RoBERTa, GPT, and BERT. It enables researchers and practitioners to refine or apply these models to downstream tasks like sentiment analysis, question answering, and summarization without requiring a lot of infrastructure or expertise. The architecture, utility, and community contributions that have established the library as a standard tool in NLP are described in the paper. Its effects include promoting research repeatability, lowering the entry barrier for AI development, and expediting testing. It does, however, have the shortcomings of underlying models, including high resource requirements, hallucinations, and biases in the data. Despite not developing new algorithms, the library has a significant engineering, scalability, and pedagogical impact. It has emerged as a key element in putting NLP solutions into practice across industries and academia by promoting quick innovation and collaboration in the field.

Abts, D., Kimmell, M., Ling, M., Kim, J., Boyd, S., & Bitar, N. (2021). Groq: A Software-defined Tensor Streaming Multiprocessor. ArXiv[5]: This technical paper

presents Groq, a high-performance AI hardware solution based on the Tensor Streaming Multiprocessor (TSM), a novel architecture. Groq's architecture prioritizes predictability and low-latency performance for real-time AI workloads, in contrast to traditional GPUs or TPUs. Deterministic execution and low overhead for model inference are made possible by its integration of interleaved memory, vector, and matrix compute units via a high-radix interconnection network. For streaming AI tasks like computer vision, speech recognition, and natural language processing, the TSM's consistent throughput is particularly advantageous. In addition to highlighting performance benchmarks that, in some cases, surpass conventional hardware, the paper describes the hardware-level innovations that enable highly parallel tensor computation. Its early ecosystem and lack of adaptability for highly dynamic or non-deterministic models are drawbacks, though. Groq is positioned as a competitive substitute for current hardware in AI applications that require latency. Its contribution is to optimize current AI models for production deployment rather than to create new ones. It is appropriate for use in AI-powered customer communication systems like Cortex since it tackles a problem that is becoming more and more pertinent: executing big models effectively and consistently in real-world systems.

Chen, Y., Wu, M., Zhao, Y., & Wang, L. (2021). Multi-channel Communication and Collaboration. IEEE Trans. Human-Mach. Syst[6]: In the context of AI-driven automation, this survey paper examines multi-channel communication tactics, specifically in customer service ecosystems. The authors look at how AI technologies improve responsiveness and consistency by streamlining interactions across multiple channels, including voice, chat, email, and social media. They list the essential technologies that support efficient customer engagement, including sentiment analysis, intelligent routing, and real-time escalation protocols. The study emphasizes how important it is to combine communication data from various channels in order to create a single customer profile and improve personalization. The paper's methodology synthesizes results from case studies, real-world deployments, and recent studies to make inferences about the scalability and efficiency of the system. According to the authors, using AI for multi-channel automation significantly increases operational efficiency and user satisfaction. They do, however, warn about problems like model drift across channels, data fragmentation, and the difficulty of preserving contextual continuity. The results have a direct bearing on systems such as Cortex, which uses generative AI to combine communication channels. The survey provides a basic

understanding of the difficulties and best practices associated with developing multi-channel, adaptive communication assistants.

Li, Jiwei, et al. (2016). Deep Reinforcement Learning for Dialogue Generation. EMNLP.[7]: This study presents Deep Reinforcement Learning (DRL) to enhance open-domain dialogue production, hence overcoming drawbacks of conventional sequence-to-sequence (Seq2Seq) models that frequently provide bland or generic replies. Modeling discourse production as a decision-making process, the authors use policy gradient techniques to train a neural conversational agent maximizing long-term reward. Encouraging the model to provide more interesting and contextually relevant responses, the reward function takes into account characteristics including informativeness, coherence, and simplicity of answering. Reinforcement learning is used to fine-tune the system initialized by a pre-trained Seq2Seq model. Evaluation by human judgment and automatic metrics reveals that the DRL-enhanced model generates more interactive and meaningful dialogues than baselines. Designing good reward functions, meanwhile, is still difficult; the method may occasionally give reward maximization priority above factual correctness or relevance. A first step in combining reinforcement learning with natural language creation, the work motivates later studies in goal-oriented and emotionally intelligent conversation systems. For adaptive communication aides like Cortex, this strategy offers ideas on creating systems that can learn and optimize depending on user input and long-term conversational success.

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL[8]: With the introduction of a deep bidirectional model that has been pre-trained on extensive corpora using masked language modeling and next sentence prediction, BERT (Bidirectional Encoder Representations from Transformers) represents a substantial advancement in natural language processing. BERT reads text in both directions, which allows it to comprehend a word's entire context, in contrast to conventional left-to-right or right-to-left models. BERT can be refined on a variety of NLP tasks, such as named entity recognition, sentiment classification, and question answering, after being pre-trained on Wikipedia and BookCorpus. The authors show that BERT performs better than 11 NLP benchmarks, such as GLUE and SQuAD. Since its publication, NLP has seen a surge in transfer learning, in which models are pre-trained on general data before being optimized for particular tasks. Notwithstanding its achievements, BERT has drawbacks such as sensitivity to input phrasing, high inference

latency, and a large model size. Additionally, it takes a lot of processing power to train and fine-tune. However, BERT remains a fundamental component of language understanding systems and paved the way for models such as RoBERTa and ALBERT. Adaptive communication assistants that use sentiment analysis and contextual awareness are directly informed by its architecture.

Lewis, M., Liu, Y., Goyal, N., et al. (2020). BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. ACL.[9]: The sequence-to-sequence paradigm known as BART (Bidirectional and Auto-Regressive Transformers) combines the advantages of BERT and GPT for text production and comprehension. In order to train the model to reconstruct the original input, the authors pre-train BART by introducing noise into the text (such as token deletion or shuffling). The model may learn both understanding and generation inside a single framework thanks to this denoising technique. BART can do jobs like summarizing, translating, and answering questions since its encoder is similar to BERT and its decoder is similar to GPT. BART performs better than earlier models and demonstrates great generalization skills on benchmarks such as CNN/DailyMail and XSum. Because of its dual nature, the model performs especially well in jobs requiring both content creation and interpretation, which is crucial for systems like Cortex that summarize and react to user inputs. But like other large models, it is prone to hallucination, producing believable but inaccurate outputs, and its complexity results in significant training and inference costs. Notwithstanding these problems, BART supports a broad range of AI-driven communication applications and offers a versatile framework for generative tasks.

Raffel, C., Shazeer, N., Roberts, A., et al. (2020). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer (T5). JMLR.[10]: The T5 (Text-To-Text Transfer Transformer) framework proposes treating all NLP tasks as text-to-text problems, from classification to translation and summarization. Trained on the large C4 dataset (Colossal Clean Crawled Corpus), T5 uses a Transformer encoder-decoder architecture. The model demonstrates that consistent formatting of input-output pairs improves generalization and enables transfer learning across tasks. The authors experiment with various model sizes, including T5-11B, showing significant performance gains as size increases. T5 achieves state-of-the-art results on several NLP benchmarks, including GLUE, SuperGLUE, and CNN/DailyMail. A major contribution of the paper is its emphasis on pre-

training and task reformulation, encouraging uniformity in model design. While T5 shows strong results, it requires enormous computational resources for both training and deployment, raising concerns about accessibility and energy efficiency. Moreover, it still suffers from issues common to large language models, such as biased outputs and overfitting to noise in pre-training data. T5's flexible, task-agnostic approach makes it ideal for systems like Cortex that handle multiple language processing functions within a unified interface.

Zhang, T., Kishore, V., Wu, F., et al. (2020). PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization. ICML.[11]: PEGASUS is a Transformer-based model optimized for abstractive summarization through a novel pre-training objective called "Gap Sentence Generation" (GSG). In this setup, the model learns to generate important sentences that were intentionally removed from an input document, teaching it to produce high-quality summaries. The authors pre-train PEGASUS on massive news and web corpora, showing that it achieves state-of-the-art results on 12 summarization datasets, including CNN/DailyMail, XSum, and Gigaword. The model excels at capturing salient information and generating fluent, coherent summaries, making it ideal for applications where brief and meaningful communication is essential. PEGASUS performs better than BART and T5 on most summarization tasks and is particularly strong in low-resource scenarios. However, its limitations include training complexity and occasional generation of factually inaccurate content. Despite this, PEGASUS sets a benchmark in the summarization domain and is especially relevant for projects like Cortex, where auto-summarization of user conversations and emails is a key feature. Its ability to extract and compress information mirrors how a human might create summaries, thus offering high-quality language generation with efficiency.

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. EMNLP.[12]: Sentence-BERT (SBERT), a BERT modification created to produce semantically meaningful sentence embeddings, is presented in this paper. Traditional BERT is computationally costly and not optimized for clustering or sentence similarity, despite its strong performance on classification tasks. This is addressed by SBERT, which fine-tunes BERT using Siamese and triplet network structures to efficiently compute sentence embeddings appropriate for information retrieval and semantic search. The model is useful for large-scale applications like document matching or duplicate detection because it drastically cuts down on the amount of time required to compare sentence pairs. The

authors test SBERT on several benchmarks, such as SNLI and STS Benchmark, and find that it performs better than baseline models. Semantic closeness between sentences can be ascertained using SBERT embeddings in conjunction with cosine similarity. It does, however, carry over BERT's drawbacks, including word order sensitivity and pre-training data biases. For systems like Cortex that must conduct quick and precise semantic searches across a dynamic knowledge base, SBERT is especially pertinent. Its architecture enables clustering, personalized content delivery, and query matching, providing a scalable and efficient solution for real-time language comprehension tasks.

Kalyan, K. S., & Sangeetha, S. (2021). SEC-LDA: A Semantic Enhanced Clustering Model for Topic Modeling. Journal of King Saud University – Computer and Information Sciences.[13]: By adding word embeddings to improve semantic coherence, SEC-LDA (Semantic Enhanced Clustering Latent Dirichlet Allocation) suggests an enhancement over conventional topic modeling. SEC-LDA incorporates semantic information from pre-trained embeddings such as Word2Vec or GloVe to enhance topic quality, in contrast to classic LDA, which only uses word co-occurrence. This model allows for more coherent and human-interpretable topics by clustering documents not only according to frequency but also according to latent semantic patterns. The study shows that SEC-LDA performs better on benchmark datasets like Reuters and 20 Newsgroups than both conventional LDA and even other neural topic models. Human judgment tests and topic coherence scores are examples of evaluation metrics. Increased computational overhead and reliance on embedding quality are among the drawbacks. However, SEC-LDA is very helpful for tasks that are essential to Cortex, such as classifying customer queries, organizing content, and curation of knowledge bases. SEC-LDA enhances topic-based applications' interpretability and accuracy by capturing deeper semantic relationships. This can aid in automatically classifying and elevating related user queries for improved routing and quicker response times.

Bao, Y., Duan, N., Yan, Z., et al. (2021). PLM-Energy: Cost-Aware Pre-trained Language Models. NeurIPS[14]: A framework for assessing and lowering the energy cost of sizable pre-trained language models without compromising performance is presented by PLM-Energy. By highlighting the trade-off between accuracy and energy consumption, the authors suggest a cost-conscious fine-tuning technique that makes use of dynamic neuronal pruning and energy-aware loss functions. PLM-Energy can save up to 40% on energy

consumption while preserving or marginally increasing accuracy on downstream tasks like question answering and sentiment categorization, according to experiments conducted on well-known models like BERT and RoBERTa. In addition to providing tools to evaluate energy consumption during training and inference, the study highlights the growing significance of sustainable AI development. Nevertheless, the method could cause latency in real-time systems or performance deterioration on extremely complicated tasks.

2.2 KEY GAPS IN THE LITERATURE

Even though current research highlights the potential of AI-powered email automation systems, there are still a number of significant issues that need to be resolved. These drawbacks impair their efficacy, versatility, and efficiency in practical applications.

1. **Scalability** For AI-based email systems, scalability remains a major obstacle. Because of the computational limitations present in Natural Language Processing (NLP) and Large Language Model (LLM) workflows, many existing solutions are unable to effectively handle high email volumes. Even though performance has increased due to recent developments in AI frameworks, integrating dynamic, large-scale vector search models (like Chroma) still requires a significant amount of computing power, which restricts its practical implementation for companies with fluctuating workloads.
2. **Implementation Complexity** The intricacy of models like BERT and GPT's architecture makes it difficult to incorporate AI into current communication processes. Existing systems frequently lack thorough implementation instructions and necessitate specific knowledge for adjustment and personalization. This limits their accessibility, particularly for non-technical teams and smaller businesses looking to use AI-powered email solutions.
3. **Ethical and Privacy Issues**, has few studies examine closely the ethical and privacy concerns artificial intelligence-powered email systems generate. Training on client data brings up issues of data security and regulatory compliance including the GDPR. The ethical usage of such technologies is made much more challenging to describe by the lack of developed methods for anonymizing sensitive data while maintaining contextual relevance.
4. **Data Shortcomings** is the development of effective artificial intelligence email systems is greatly hampered by the absence of annotated datasets particular to business communication. Though they provide some flexibility, pre-trained models often underperform in domain-specific situations because of a lack of language and contextual diversity in training data. They are therefore less able to apply to various

corporate contexts.

5. **Confidence** and Knowledge of AI-generated responses' lack of understanding is a significant barrier to acceptance. Although several systems are rather accurate, they frequently lack rationale for their results. User confidence is eroded by this lack of transparency since wrong or improper responses could harm consumer relations and brand reputation.
6. **Generalization** of Many different company sectors or communication styles can be difficult for AI email systems to fit. When confronted with domain-specific inquiries or formal language, a model trained on generic datasets may not perform effectively. To guarantee constant performance over a spectrum of use cases, better domain adaptation techniques are required.
7. **Cost of Computation** means High computational costs connected to running LLMs such as BERT and GPT impede general use. For small and medium-sized companies lacking access to high-performance infrastructure, these expenses are particularly burdensome
8. **Problems** with negative transfer and overfitting are problems still exist with overfitting and negative transfer. When fine-tuned on tiny or unbalanced datasets, pre-trained models frequently underperform, producing contextually irrelevant or repetitive replies. This emphasizes the requirement of more sophisticated fine tuning methods and enhanced context-aware training procedures.

CHAPTER 03

SYSTEM DEVELOPMENT

3.1 REQUIREMENTS AND ANALYSIS

This chapter is driven by the basic requirements and design elements of this chapter; Cortex is an AI-powered email automation tool. The analysis covers both hardware and software elements, hence ensuring the system's scalability, dependability, and adaptation to several operational environments

3.1.1. HARDWARE REQUIREMENTS

The Hardware specifications are defined by the computing demands of natural language processing models, real-time email processing, and vector search systems. Depending on the degree of deployment, either local or cloud-based servers can run the system:

Table 3.1- Table of Hardware Requirement

Component	Minimum Requirement	Recommended Requirement
Processor (CPU)	Intel i5 10th Gen or AMD Ryzen 5	Intel i7/i9 or AMD Ryzen 7/9
RAM	16 GB	32 GB or higher
Storage	512 GB SSD	1 TB SSD + optional cloud storage
GPU	NVIDIA GTX 1660 or equivalent	NVIDIA RTX 3060/3090 (for model training)

Network	Stable broadband internet (≥ 10 Mbps)	High-speed internet (≥ 100 Mbps)
----------------	---	--

3.1.2. SOFTWARE REQUIREMENTS

The software ecosystem is made up of programming tools, libraries, and environments needed for data preprocessing like (SQLite), model building (TensorFlow), and result visualization (Data Analysis).

Table 3.2 Table of Software Requirements

Software Component	Description
Operating System	Ubuntu 20.04 LTS / Windows 10 or higher
Programming Language	Python 3.9+ (Primary development language)
Frameworks & Libraries	<ul style="list-style-type: none"> - TensorFlow / PyTorch (for model development)- Transformers (Hugging Face) - LangChain (for LLM orchestration) - Scikit-learn, NLTK, SpaCy (for classical NLP tasks)
Backend Framework	FastAPI / Flask (for building RESTful APIs)
Database	PostgreSQL / MongoDB (for structured/unstructured data storage)
Vector Store	ChromaDB / FAISS (for semantic search and embeddings storage)

Deployment Tools	Docker (containerization), Git (version control), CI/CD pipelines (GitHub Actions, Jenkins)
Cloud Services	Azure OpenAI / OpenAI API (for inference), AWS / Azure / GCP (for deployment and scaling)
IDE & Tools	VS Code / Jupyter Notebook / PyCharm
Monitoring & Logging	Prometheus + Grafana / ELK Stack (for observability in production)

3.1.3 FUNCTIONAL REQUIREMENTS

These criteria define the specific actions, traits, and features Cortex must support to satisfy user needs and address the stated problem of email automation and personalization.

1. Classification of Emails

Incoming emails must be classified by the system into numerous categories including like (complaints, comments, questions, and general queries). Accurate email content sorting will depend on pre-trained NLP models like, BERT, that have been fine-tuned on email datasets and gives better results.

2. Understanding Context and Recognizing Intent

Cortex has to exactly identify the underlying purpose and intricacies of context of every email. The system should be able to distinguish between tiny changes in tone and language by using transformer-based models, such as RoBERTa or GPT variants, therefore enabling it to correctly read human intent.

3. Automated Generation of Responses

The system should create suitable and context-aware responses for classified emails using enhanced T5 or GPT-3.5 or other large language models (LLMs). These replies have to be relevant to the original message, grammatically correct, and consistent.

4. Customized Email Response

Responses should have background information including name, tone preference, question history, and past interactions to dynamically customize them for each user. More human-like communication and more active participation will follow from this personalization.

5. Caching and Reusing Queries

The system has to store and retrieve responses to previously answered queries using a query-response cache with independent hash-based identifiers. This system will decrease repeated processing and increase system performance by delivering quick solutions to often requested inquiries.

6. Human Handoff and Escalation Management

Should the system sense uncertainty in its response or find ambiguous questions, it has to launch an escalation protocol. This means sending the question to a human agent along with all relevant background information and classification data to keep.

7. Dynamic Query Matching with NLP Embeddings: Corex needs to use models like SBERT or BERT to translate incoming queries into embeddings in order to apply semantic similarity matching. To increase accuracy and response time, these embeddings will be compared to a vector store to identify and reuse the most pertinent previous answers.

8. Integration of Multi-Channel Communication

Popular communication platforms like Microsoft Teams, Slack, Outlook, and Gmail should all work flawlessly with the system. This enables it to use a single backend to retrieve and reply to user messages from various channels.

9. Feedback Gathering and the Learning Cycle

Cortex must ask users or agents to score the response's quality and applicability after it has been sent. Through cycles of ongoing learning and retraining, this feedback will be saved and utilized to improve model performance.

10. Transparency and Audit Logging

Time stamps and metadata must be recorded for each interaction the system processes, including classifications, model decisions, and generated responses. Transparency, error tracing, and regulatory compliance are made possible by this audit trail.

11. Domain-Specific Modification

To guarantee precise language modeling and terminology usage, Cortex should allow fine-tuning for various business domains (such as legal, e-commerce, and healthcare). This will enhance model accuracy in specific contexts and enable tailored workflows.

12. Role-Based Access Control and User Authentication

Administrators must be able to assign roles like Administrator, Analyst, or Viewer, and the system must have a secure login process. To guarantee data security and integrity, access to different features (such as training models, viewing logs, and editing templates) must be permission-based.

3.1.4 NON-FUNCTIONAL REQUIREMENTS

These define the system's quality attributes, such as performance, security, and scalability, focusing on how the system performs rather than what it does.

1. Efficient and Reactive

In order to provide a smooth user experience and minimal communication delays, the system must produce and distribute email responses for common questions with an average latency of 1-2 seconds.

2. Scalability

In order to handle more emails and concurrent user sessions, Cortex needs to be designed with performance in mind. Container orchestration systems like Kubernetes should allow for horizontal scaling.

3. Reliability and Availability

At least 99.5% uptime should be maintained by the system to ensure consistent service availability. It is necessary to incorporate fault-tolerant methods and auto-recovery capabilities in order to prevent and manage system crashes.

4. Security and privacy of data

Cortex must abide by data protection regulations like the GDPR. All sensitive information must be encrypted, both while it is in transit and when it is at rest. Ensure that role-based access control and secure API gateways are used to prevent unwanted access.

5. Adaptability and Maintainability

Cortex should be designed with modularity in mind to facilitate feature additions, debugging, and updates. Clean architecture principles should be followed and code should have comprehensive documentation.

6. Utilizability

The user interface of the system should be simple enough for both technical and non-technical users to use. Dashboards for administrators, escalation processes, and feedback forms must all be simple to use.

7. Mutual compatibility

Integration with a variety of third-party tools, APIs, and communication services should be made easy by Cortex without needing a lot of reconfiguring for seamless incorporation into existing processes.

8. Record Keeping and Auditability

All user interactions, system actions, and classifications must be documented for audit and review purposes. In order to facilitate troubleshooting or compliance inspections, logs should be securely stored and easily accessible.

9. Localization and Language Support

Localization features that enable replies to be customized to user areas and support for several languages are essential for ensuring global usability and accessibility.

10. Productivity of Resources

Optimization of resource consumption is necessary for Cortex to minimize

computing costs. For this, it is necessary to use lazy loading or on-demand processing, cache frequently used queries, and use lightweight models when possible.

3.1.5 ANALYSIS APPROACH

1. Method of Analysis

Cortex's analysis strategy is centered on comprehending the fundamental issue of ineffective email communication and figuring out how AI-driven automation can offer scalable, flexible, and customized solutions. To guarantee a thorough and efficient system design, the following approaches were used:

2. Defining the issue and establishing objectives

Clarifying the problem statement—the absence of intelligent, scalable email automation that comprehends context and tailors responses—was the first step. The objective was to create a system that, with little assistance from humans, could categorize, decipher, and respond to emails from a variety of business domains.

3. Analysis of Stakeholders

Target users, such as customer service representatives, business analysts, and technical leads, were surveyed and interviewed in order to determine their needs, expectations, and pain points with the system.

4. Elicitation of Requirements

Through literature reviews, competitor analysis, and brainstorming sessions, functional and non-functional requirements were determined. This aided in the identification of essential features such as escalation protocols, intent recognition, response generation, and classification.

5. Technology and Analysis of Feasibility

Existing NLP frameworks, LLMs (like GPT and BERT), and vector databases (like Chroma and FAISS) were all compared. This aided in evaluating performance benchmarks, integration complexity, and computational costs.

6. Evaluation of Risk

Potential hazards were noted, including high computational overhead, model hallucination, overfitting, and data privacy issues. The planning stage included mitigation techniques like query caching, domain adaptation, and feedback loops.

7. Modeling Use Cases and Scenarios

To visualize system behavior in a variety of email contexts, including customer complaints, support inquiries, and feedback, real-world scenarios were mapped out. These are aided in identifying the escalation case of thresholds and improving the logic flows.

8. Planning for Architecture

The creation of a high-level architectural model was based on a modular, & microservices approach. Elements such the response generator, caching layer, logging system, and NLP engine were logically specified and spread out for efficient development and deployment

9. Method of Validation

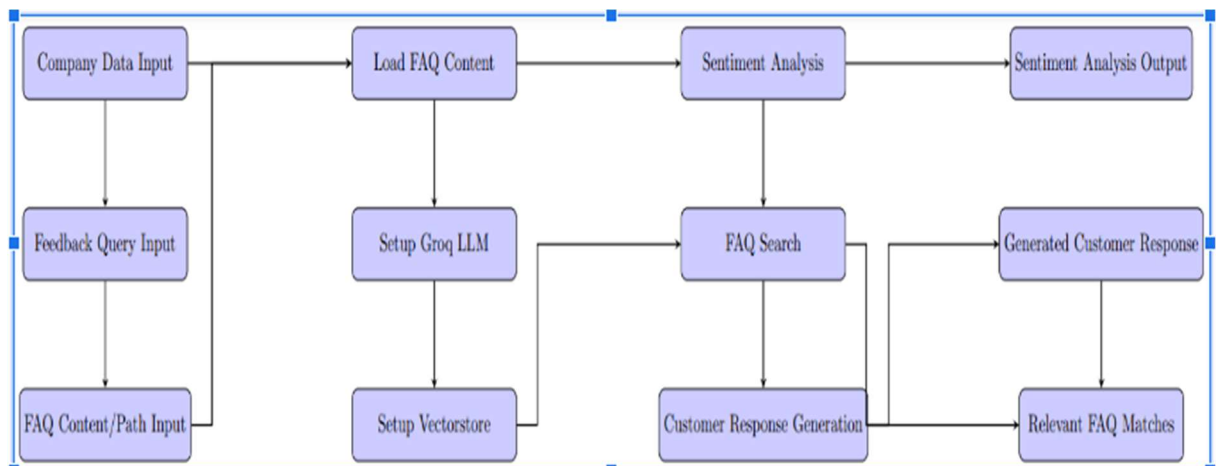
Early on, the system's evaluation metrics were determined, including fallback frequency, response latency, user satisfaction, and classification accuracy. Iterative improvements were guided by a feedback-driven evaluation loop.

3.2 PROJECT DESIGN AND ARCHITECTURE

3.2.1 SYSTEM ARCHITECTURE

The multi-channel communication management, architecture, and design of the Cortex system determine its scalability, security, and usefulness. This part offers a comprehensive study of the tools, languages, and technologies used in the architectural design overview.

Figure 3.1 Project Design Of The System



The Cortex system is carefully built to show versatility, adaptability, and smart automation of RAG by means of modern language models and vector-based information retrieval for smart multi-channel communication management. The architecture is set up to allow smooth knowledge integration, sentiment-aware consumer engagement, and contextual response generation. Looking at the main parts and how they work:

User Interface (Interaction Based on CLI): At present prototype phase, the Cortex system engages users using a Command Line Interface (CLI). Users immediately input corporate data and feedback requests via terminal prompts. Though basic, this interface offers a consistent platform for testing and demonstrating backend intelligence pipeline. Future releases could include a UI created using Streamlit or React for better use JavaScript.

Core Processing: Built using LangGraph, a state graph architecture supporting structured, conditional workflows, the Cortex system's core is a stateful, agent-based processing pipeline. This layer generates customized responses, sentiment analysis, and semantic FAQ retrieval oversight:

1. **Sentiment Analysis Agent:** This part points out various feedback components and sketches the overall attitude (positive, negative, neutral, or mixed) using Groq's strong Large Language Model (LLM). Every component's categorization together with its emotional polarity and rationale provides great insight into consumer input.

2. **FAQ Retrieval Agent:** This agent uses HuggingFace Embeddings, Chroma Vectorstore for semantic search which drives this bot's relevant responses. It improves relevance and accuracy by analysing corporate FAQs and getting the most semantically relevant entries depending on user input.
3. **Response Generation Agent:** The system creates tailored, professional responses using Groq's LLM. It ensures that the response is accurate, sympathetic, and addresses both praises and concerns by combining sentiment analysis with FAQ background/

LangGraph State Machine's Workflow Engine: Central Driven by LangGraph's StateGraph, a strong and adaptable workflow engine drives the processing logic of the Cortex system, central to which is directed state machine representation of the system's operations. Beginning with sentiment analysis, this engine specifies a separate, sequential flow for processing consumer comments; next comes FAQ retrieval; last comes response generation. Starting at the `analyze_sentiment` node, the technique assesses the user's input to find general and aspect-based sentiments. Control shifts to the `search-faq` node after sentiment analysis; this node uses vector similarity to semantically compare the remarks with corporate FAQs, therefore finding pertinent information. All collected insights combined creates the next node, `generate_response`, which generates a sympathetic and contextual consumer response. The `END` node marks the last step of the process

Combining Knowledge: Mainly used to dynamically maintain company-specific FAQs in sync, the Cortex system combines a clever and flexible approach to knowledge integration. A structured JSON file or UI input lets the user give this information. In the import case, a `CharacterTextSplitter` splits the FAQ content into reasonable-sized chunks most appropriate for semantic embedding. These embeddings, generated by HuggingFace models, are kept in a Chroma vector storage to allow exact similarity searches. Since it lets the system fast return suitable responses without fine-tuning or retraining language models, this approach is adaptable to be employed in any domain or product category with minimum configuration work

Error Handling and Persistence: End-to-end error management and persistence capabilities of the Cortex system provide traceability and reliability. A history JSON file tracks all feedback questions, results, and timestamps recorded by an Advanced-Query-

Processor component. This function allows potential retraining, audits, and post-processing activities. The system also enables automated retry policies, which allow failing requests to be re-submitted a given number of times before an unsuccessful marking occurs. Reasonable error messages from custom error handlers improve Python's native logging system for systematic problem discovery. This polyphrenic error management solution increases system dependability and provides transparency under failure or irregular operation

Modularity and Extensibility: Cortex, built on extensibility and adaptability, allowing the system evolve and change with requirement without any trouble. Response generation, FAQ retrieval, and sentiment analysis are all key elements wrapped by an independent class, thereby allowing autonomous construction, testing, and replacement. Modularity enables extra features like deployment-ready APIs or web interfaces for real-time interaction with commercial systems, emotion recognition for improved knowledge of consumer mood, and multi-language compatibility for use in international applications. The obvious duty distribution not only helps maintenance and debugging but also allows fast prototyping of new features without sacrificing fundamental system operation.

3.2.2 TOOLS, TECHNOLOGIES, AND LANGUAGES

A wide and strong foundation of computer languages, technologies, and tools supports the Cortex system. Every system component is selected to perform a particular function such semantic retrieval, LLM orchestration, sentiment analysis, and answer production. This makes the entire system modular, stable, and scalable. Cortex's success depends on the following elements:

1. **Python:** Cortex employs the light and agile Python programming language to manage systems. Python is also utilized in machine learning and async programming. It boasts an enormous ecosystem with core libraries such as json to serialize light data, logging for debugging, and asyncio for concurrency. Since it interacts with state-of-the-art AI libraries, building a Python end-to-end smart feedback assistant is perfect.
2. **LangChain:** LangChain is the orchestration layer simplifying interaction with vector stores and Large Language Models. Its abstractions include PromptTemplates,

RunnableGraphs, and retrievers that allow smooth integration of dynamic memory, structured prompts, and RAG pipelines. Cortex uses LangChain to control the movement between agents, run LLM queries, and parse findings for context-aware processing.

3. **StateGraph (LangGraph):** Cortex's sequential logic is represented as a state machine using LangGraph's StateGraph module. A node encapsulates every phase of the processing pipeline: response generation, FAQ search, and sentiment analysis. The graph allows for conditional branching, failure recovery, and clean handoff of shared state (AgentState) between asynchronous agents. In complicated LLM-driven processes, this improves both modularity and traceability.
4. **Groq LLM Interface:** Using their very optimised mistral-saba-24b model, Groq acts as the main engine for producing and understanding language. Unlike conventional transformer-based models, Groq uses a segregation architecture that divides work into separately optimized components, therefore improving concurrency and lowering latency. Cortex depends on Groq for thorough feedback breakdowns, organized sentiment analysis, and very focused consumer responses.
5. **HuggingFace Embedding:** Cortex uses HuggingFace sentence-transformer-based models to turn natural language material into thick vector representations. The semantic similarity engine used for both feedback-to-FAQ matching and context scoring in the WAVY algorithm runs on these embeddings. By means of embeddings, vector-based search and ranking are made possible by means of a bridge connecting raw text and numerical thinking.
6. **ChromaDB (Vector Store) :** Cortex stores and retrieves embeddings for FAQs and domain-specific material using Chroma, an open-source vector database. Chroma supports constant vector storage and lets high-performance similarity searches using cosine distance metrics. Especially for finding top-k semantically relevant papers in reaction to a query, its function is basic in the RAG pipeline of the system.
7. **Pydantic:** Pydantic defines and validates structured models for internal state representation spread across agents. Modeled as Pydantic classes, key elements like

AgentState, CustomerFeedback, and FeedbackAspect guarantee rigorous type-checking and automatic serialization. Pydantic is also used to control asynchronous agent replies and consolidate outputs for LLM consumption in the last response.

8. **Cosine Resemblance:** Cosine similarity, a key element of Cortex's semantic search engine, gauges the proximity between embedded vectors. It is used in the context scoring of the WAVY algorithm as well as in FAQ retrieval. By semantic meaning rather than just keywords, this mathematical approach lets the system rank data, therefore improving retrieval accuracy.
9. **AQP: Advanced Query Protocol:** Originally a bespoke solution, AQP records every feedback event with status codes, timestamps, and results. It keeps track of failed queries in a JSON file for post-analysis and offers retry options. Unlike general-purpose tools like Langsmith, AQP is optimized for internal diagnostics and feedback loop integration, which underpins RLHF (Reinforcement Learning with Human Feedback) practices inside the framework.
10. **Attention Mechanisms and Softmax:** LLM decision-making incorporates the Softmax function to turn raw logits into probability distributions for token selection. Likewise, attention mechanisms—especially Scaled Dot-Product Attention—are naturally included in the transformer-based models employed in Cortex. These let the model concentrate on pertinent tokens inside a prompt, therefore enhancing contextual correctness in produced replies.
11. **Top-K Retrieval:** Cortex ranks and retrieves the most relevant vectors from ChromaDB with a Top-K algorithm. By guaranteeing that only the highest-scoring material is sent into the LLM for final response generation, we can maximize both speed and relevance in the RAG process.
12. **Algorithm WAVY:** WAVY, or Weighted Aspect Vector Yield, a proprietary algorithm, was created to enhance sentiment-aware generation. WAVY builds a targeted query for LLM processing by segmenting feedback, assessing sentiment and

domain relevance for each segment. This allows the system to manage more precisely and with more user alignment complicated, neutral, or mixed comments.

13. **Data Serialization using JSON:** JSON stores all configuration, query history, FAQ material, and feedback logs. Its compatibility with most computer environments and human-readable format make it perfect for storing structured data without the need for sophisticated database systems like SQL.

3.2.3 ARCHITECTURE

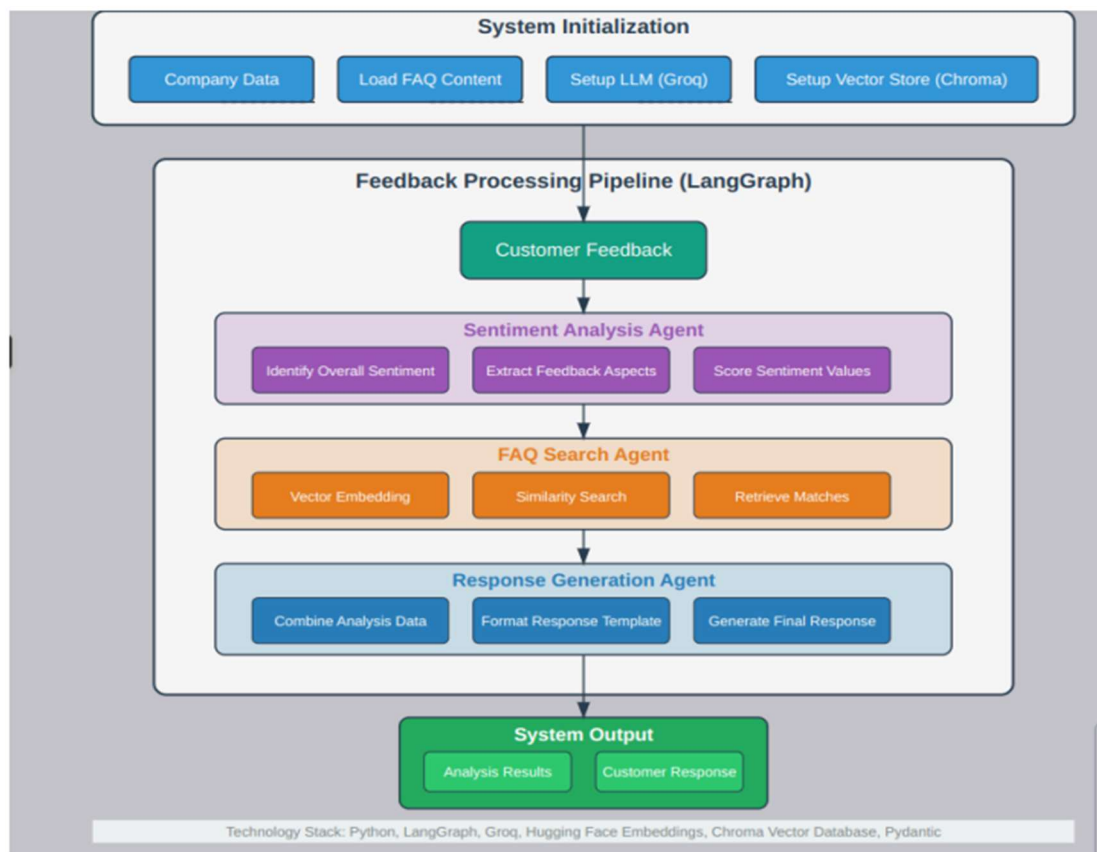
The project's goals of guaranteeing data security, scalability, and user-friendliness are all met by the selected architecture:

1. **Intelligence and Context-Awareness:** By improving Cortex's fundamental strong Retrieval-Augmented Generation (RAG) system, the novel WAVY algorithm enables aspect-sensitive and sentiment-aware response generation. Advanced NLP methods guarantee that every customer contact is managed precisely, therefore extracting emotional and contextual cues from user input to generate responses that are both pertinent and sympathetic
2. **Scalability:** Cortex's modular parts—like LangChain, ChromaDB, and HuggingFace embeddings—allow for simple scaling. Vector databases and stateless LLM APIs—like Groq—allow the system to scale horizontally as more input and concurrent users drive it. Integration readiness with cloud platforms—like AWS—also guarantees simple scaling for enterprise-level deployment
3. **Reliability and Feedback Optimization:** The system logs errors for analysis, automatically retries failed queries, and tracks every interaction using Advanced Query Protocol (AQP). Using actual user input as training signals, this allows reinforcement learning processes (RLHF) that constantly improve LLM performance.
4. **Modularity and Flexibility:** Using a state-machine-based architecture (LangGraph), each processing phase—sentiment analysis, FAQ matching, and response generation—is isolated as an agent, enabling developers to swap or upgrade

components independently. This design allows for the integration of future capabilities including voice-based input/output, emotion classification, and multilingual support.

5. **User Experience and Efficiency:** By providing timely, sentiment-aligned replies with great semantic correctness, Cortex increases user involvement. By means of Top-K retrieval and targeted query construction, the system's smart filtering guarantees that only the most pertinent information is used in replies, therefore lowering noise and enhancing clarity. This improves customer happiness and quality of communication straight away.

Figure 3.2- Architecture of Cortex



3.3 IMPLEMENTATION

The implementation phase of the Cortex project aims to turn the architectural blueprint of the system into a completely operational, smart communication tool. Orchestrating several

components—including language model integration, vector-based semantic search, structured workflows, and dynamic user interaction—into a coherent pipeline constitutes this process. This part describes the main application areas and methods employed:

1. **System Start-up and Configuration:** A CLI interface starts the system; the user enters company-specific information including name, product categories, and FAQ data either manually or from a JSON file. Encapsulated in a structured `CompanyData` model, these inputs guarantee consistent formatting for downstream parts. The `FeedbackSystemController` builds a `ResponseSystem` object upon initialization, therefore setting up necessary components including the LLM, vector store, and agent pipeline. Dotenv or runtime prompting securely loads the environment variables, including the Groq API key, so preserving data privacy.

Figure 3.3- Code to Start the Cortex

```
async def enhanced_main():
    print("\n=====")
    print("  Customer Feedback Analysis System  ")
    print("          Enhanced Edition           ")
    print("=====\\n")

    controller = FeedbackSystemController()

    try:
        # Offer sample data option
        print("Would you like to:")
        print("1. Enter company information manually")
        print("2. Use sample data for testing")
        setup_choice = input("Enter your choice (1/2): ").strip()

        if setup_choice == "2":
            # Setup with sample data
            print("\\nSetting up system with sample data...")
            sample_faq_path = "sample_faq.json"

            if not os.path.exists(sample_faq_path):
                print("Creating sample FAQ data...")
                if not create_sample_faq_json(sample_faq_path):
                    print("Failed to create sample data. Switching to manual setup.")
                    setup_choice = "1"

            if setup_choice == "2":
                company_data = CompanyData(
                    name="Sample Company",
                    faq_path=sample_faq_path,
                    product_categories=["Electronics", "Home Goods", "Apparel"]
                )
```

2. **Language Model Integration:** Groq + LangChain, Tasks like the following are accomplished by the system using Groq's high-performance LLM (e.g., mistral-saba-24b) through LangChain's ChatGroq wrapper. Breaks down comments into several facets and categorizes them as positive, negative, or neutral. Response Generation: Creates contextually rich, sentiment-aware, succinct, sympathetic replies. For structured communication, SystemMessage and HumanMessage objects build all LLM prompts; replies are parsed and merged into the shared state for next processing.

Figure 3.4 Initializing the Groq LLM

```
def setup_llm(self):
    """Set up Groq LLM client"""
    groq_api_key = os.environ.get("GROQ_API_KEY")
    if not groq_api_key:
        print("\nGroq API key not found in environment variables.")
        groq_api_key = getpass("Enter your Groq API key: ")
        os.environ["GROQ_API_KEY"] = groq_api_key

    self.llm = ChatGroq(
        groq_api_key=groq_api_key,
        model_name="mistral-saba-24b",
        temperature=0.7,
        max_tokens=4096
    )
```

Figure 3.5 - To initialize system message and human message

```
messages = [
    SystemMessage(content=template),
    HumanMessage(content="Please generate a response based on the above information.")
]

faq_info = "\n\n".join(state.get("faq_matches", []))
if not faq_info:
    faq_info = "No relevant FAQ matches found."

response = await self.llm.ainvoke([
    SystemMessage(content=template.format(
        feedback_text=original_text,
        feedback_analysis=json.dumps(feedback_analysis, indent=2),
        faq_info=faq_info
    )),
    HumanMessage(content="Please generate a response based on the above information.")
])
```

3. **Vector Embedding-Based Semantic Retrieval:** The system converts text data into vector form using HuggingFace Embeddings, so facilitating smart FAQ matching. ChromaDB, an open-source vector store that supports semantic similarity search, stores and queries these embeddings. To guarantee best embedding granularity, CharacterTextSplitter first preprocesses and chunks the FAQs. The FAQSearchAgent increases response contextuality by doing a similarity search to get the top-k most relevant matches when feedback is received.

Figure 3.6-Vector Base Creation

```
def setup_vectorstore(self):  
    """Set up vector store for FAQ search"""  
    text_splitter = CharacterTextSplitter(  
        separator="\n\n",  
        chunk_size=1000,  
        chunk_overlap=200  
    )  
    texts = text_splitter.split_text(self.faq_content)  
  
    embeddings = HuggingFaceEmbeddings()  
  
    self.vectorstore = Chroma.from_texts(  
        texts=texts,  
        embedding=embeddings,  
        persist_directory="./chroma_db"  
    )
```

4. **LangGraph Workflow Coordination**

Workflow Coordination with LangGraph Modeled as a directed state machine, LangGraph's StateGraph orchestrates the whole processing flow. The specified process comprises:

sentiment_analysis , search_faq ,response_generation and END

Every node gets and modifies a shared AgentState dictionary comprising status metadata, intermediate results, and input data. This allows for modular debugging and smooth, trackable transitions between processing stages.

Figure 3.7-State defines with help of LangGraph

```
def setup_graph(self):
    """Set up the workflow graph"""
    workflow = StateGraph(AgentState)

    # Add nodes for each step in the process
    workflow.add_node("analyze_sentiment", self.sentiment_agent.analyze)
    workflow.add_node("search_faq", self.faq_agent.search)
    workflow.add_node("generate_response", self.response_agent.generate)

    # Define the edges - the flow of the process
    workflow.add_edge("analyze_sentiment", "search_faq")
    workflow.add_edge("search_faq", "generate_response")
    workflow.add_edge("generate_response", END)

    # Set the entry point
    workflow.set_entry_point("analyze_sentiment")

    # Compile the graph
    self.graph = workflow.compile()
```

5. Error Management and Advanced Query Handling: A query execution monitor called Advanced Query Processor runs queries and uses retry logic in the event of failures. Should a query fail, the system logs the error into a JSON file with a timestamp and thorough metadata after two attempts. This not only ensures higher reliability but also generates data for future fine-tuning and performance audits.

Figure 3.8- Advanced Protocol Define

```
class AdvancedQueryProcessor:
    def __init__(self, controller: FeedbackSystemController):
        self.controller = controller
        self.history = [] # Store query history

    async def process_query_with_retry(self, query: str, max_retries: int = 2) -> Dict:
        """Process a query with automatic retry on failure"""
        attempts = 0
        while attempts <= max_retries:
            try:
                result = await self.controller.process_query(query)
                if result["status"] == "success":
                    # Add to history on success
                    self.history.append({
                        "timestamp": datetime.now().isoformat(),
                        "query": query,
                        "result": "success"
                    })
                return result
                attempts += 1
            if attempts <= max_retries:
                print(f"Processing failed, retrying ({attempts}/{max_retries})...")
        except Exception as e:
            attempts += 1
            logger.error(f"Error in query processing: {str(e)}")
            if attempts <= max_retries:
                print(f"Error occurred, retrying ({attempts}/{max_retries})...")
```

6. WAVY: A Segmentation Algorithm for Feedback Using a particular algorithm WAVY (Weighted Aspect Vector Yield), the strategy enhances accuracy in processing vague or sophisticated comments. It breaks down the input, determines sentiment and context relevance scores, and formulates proper questions requesting the LLM to respond with highly precise answers. This offers sentiment-aware communication even in ambiguous or mixed comment environments. It splits the input into chunks, identifies sentiment and context relevance scores, and builds correct queries that ask the LLM to generate very specific answers. This offers sentiment-aware communication even in ambiguous or mixed comment environments

Figure 3.9-Wavy Algorithm

```
# Agent definitions
class SentimentAnalysisAgent:
    def __init__(self, llm):
        self.llm = llm

    async def analyze(self, state: AgentState) -> AgentState:
        """Analyze sentiment and identify different aspects of feedback"""
        company_name = state["company_data"]["name"]
        text = state["feedback_text"]

        if len(text.strip()) < 10:
            state["sentiment_analysis"] = CustomerFeedback(
                overall_sentiment="neutral",
                aspects=[FeedbackAspect(
                    aspect="general",
                    sentiment="neutral",
                    score=0.5,
                    text="Feedback too brief for detailed analysis"
                )],
                mixed_feedback=False
            ).model_dump()
        return state
```

7. User Interaction and Output Delivery: After generating the response, the user is presented with it along with a detailed sentiment analysis and related questions. The user is then invited to either end the session or leave further comments, thereby establishing an interactive and iterative communication channel. This offers sentiment-aware communication even in ambiguous or mixed comment environments.

Figure 3.10- Response Generation

```
class ResponseSystem:
    def __init__(self, company_data: CompanyData):
        self.company_data = company_data
        self.faq_content = self.load_faq_content()
        self.setup_llm()
        self.setup_vectorstore()
        self.setup_agents()
        self.setup_graph()

    def load_faq_content(self) -> str:
        """Load FAQ content from either direct input or JSON file"""
        if self.company_data.faq_content:
            return self.company_data.faq_content
        elif self.company_data.faq_path:
            try:
                with open(self.company_data.faq_path, 'r') as f:
                    data = json.load(f)
                    faq_texts = [
                        f"Q: {faq['question']}\nA: {faq['answer']}"
                        for faq in data.get("faqs", [])
                    ]
                return "\n\n".join(faq_texts)
            except Exception as e:
                logger.error(f"Error loading FAQ content: {str(e)}")
                raise ValueError(f"Failed to load FAQ content from {self.company_data.faq_path}")
        else:
            raise ValueError("Either faq_content or faq_path must be provided")
```

8. History Preservation and Logging: Across the network, Python's logging mechanism tracks execution status, alerts, and exceptions. By means of structured JSON format, the save_history() function maintains the history of both successful and failed queries, therefore enabling developers to track system performance over time

Figure 3.11-Saving History and Previous Query

```
def save_history(self, file_path: str) -> bool:
    """Save query history to a file"""
    try:
        with open(file_path, 'w') as f:
            json.dump(self.history, f, indent=2)
        return True
    except Exception as e:
        logger.error(f"Error saving history: {str(e)}")
        return False
```

3.4 KEY CHALLENGES

The Developing of the Cortex Intelligent Communication System required negotiating several difficult obstacles. Ensuring the system's robustness, contextual correctness, and capacity to scale across various business sectors depended on overcoming these obstacles. Some of the main difficulties experienced during the implementation stage are listed below:

1. **Guaranteeing Contextually Correct Answers:** Producing replies that not only fit the mood of the remarks but also properly reflected the underlying issues voiced by users was one of the main difficulties. Initial efforts with generic prompts occasionally led to ambiguous or overly formal responses. Developed as a result, the WAVY algorithm divided comments into tiny pieces and evaluated them by sentiment, contextual relevance, and domain influence, so allowing far more exact question building and major response generating
2. **Dealing with Mixed and Ambiguous Emotions:** Often, customer comments include neutral, contradictory, or mixed feelings that conventional sentiment analysis finds difficult to categorize. Dealing with this called for a hybrid strategy combining LLM-powered analysis with rule-based thresholds (e.g., neutral segment percentages). One of the most difficult and technically demanding tasks was creating logic to identify and handle ambiguous feedback without adding response bias.
3. **Optimizing Semantic Search:** Although vector embeddings using HuggingFace and retrieval with ChromaDB offered strong semantic features, adjusting these systems to reliably produce very pertinent FAQ material was not simple. Among the difficulties were vector chunk size, embedding quality, and dataset noise. Ensuring accuracy in FAQ matching needed thorough testing with chunking techniques, cosine similarity criteria, and Top-K tuning.
4. **LLM Management Consistency and Latency:** Including Groq's LLM offered the benefit of speed and modular execution, but keeping consistency in sentiment classification and response formatting across several sessions was first uneven. Predictable outputs and reduced model hallucination were ensured by structured prompts, controlled token limits, and Advanced Query Protocol (AQP) retry logic.
5. **Strong Logging and Error Handling:** Error handling became absolutely vital for system stability given the system's great dependence on several asynchronous processes: LLM calls, vector search, file I/O. The design of a strong logging and retry system was called for by unanticipated model timeouts, API key issues, and input edge cases. Constructing the AQP module let us record logs for subsequent analysis

and reinforcement learning, let us catch failure situations, and let us store logs for future use.

6. **Dynamic Knowledge Base Integration:** Letting people submit FAQs either as manual entry or JSON file created variation in formatting and completeness. The data management difficulty was great in making sure the system could handle malformed, incomplete, or domain-specific FAQ data while yet providing excellent retrieval and response generation.
7. **Constructing for Scalability and Modularity:** Deep architectural planning was needed to design each component—sentiment agent, FAQ search agent, and response agent—as separate, pluggable modules using LangGraph and Pydantic. A key engineering concern was guaranteeing smooth state handoff between components, preserving type safety, and enabling future extensibility without compromising current workflows.

CHAPTER 4: TESTING

4.1 Testing Strategy

The test method controlled the functional as well as non-functional attributes of the system. The intention was to verify system stability with a range of feedback situations ranging from simple inquiries to complex, mixed-sentiment inputs. The intention was to verify system stability with a range of feedback situations ranging from simple inquiries to complex, mixed-sentiment inputs. The rigorous test strategy ensured reliability, use, and accuracy of the Cortex Intelligent Communication System. Testing the most important aspects was done with the greatest care: the newly added sentiment analysis module, FAQ retrieval function, response generation logic, and WAVY algorithms:

1. **Unit Testing:** Unit testing on core components Core components such as the SentimentAnalysisAgent, FAQSearchAgent, and ResponseGenerationAgent were unit tested to validate the correct execution of each module in isolation. From positive to highly ambiguous inputs, these tests verified the accuracy of sentiment extraction from various forms of feedback. The tests also confirmed consistency in the structure and format of outputs produced by all agents. Cosine similarity-based matching of FAQ retrieval was also investigated further to establish semantic relevance. The AdvancedQueryProcessor was also checked under failure simulation to check its error-handling resilience. Mock inputs and dummy edge-case scenarios such as empty strings and FAQs with very little semantic overlap were used to check the system's resilience against real-world idiosyncrasies.
2. **Test of Integration:** Integration testing was done to look at the interaction between modules inside the whole LangGraph pipeline after individual components were validated. Following the order from analyze_sentiment to search_faq and finally generate_response, simulated end-to-end workflows were run. This guaranteed that the shared AgentState dictionary was passed, updated, and maintained properly at every stage of the pipeline. Integration tests also evaluated the contextual consistency of final responses and verified that the user's comments semantically matched the

FAQs obtained. These tests guaranteed smooth communication among all agents and verified the logical flow of the whole system.

3. **Testing For Performance:** Performance testing using a batch of over 50 feedback queries under different loads was done to determine the system's responsiveness and scalability. Among the key performance indicators were average response time, aimed at under two seconds per query, and memory use during heavy query processing. Particular focus was paid to the latency ChromaDB and HuggingFace embeddings caused by FAQ retrieval. The inference speed of Groq's LLM was also compared to that of other models such as GPT-3.5 and Gemini Flash. Groq was the preferred engine for Cortex since it consistently showed quicker and more dependable performance, especially under high-concurrency conditions.
4. **Testing of User Simulations:** To assess the real-world usability and interaction quality of the system, a number of CLI-based user simulations were run. From a user's viewpoint, these simulations evaluated the general communication flow, therefore guaranteeing that the system reacted quickly and suitably to a broad spectrum of questions. The simulations also evaluated how well the WAVY algorithm adjusted to feedback with mixed or neutral feelings and how effectively the replies handled particular user issues. Insights from these simulations led to refinements in prompt structure and FAQ text chunking strategy, improving both relevance and clarity in generated responses.
5. **Validation of Error Handling:** Failure injection testing confirmed the strength of the system's error-handling mechanisms. To see the system's resiliency, artificial scenarios including API timeouts, damaged or missing FAQ data, and empty input submissions were created. Effective in retrying failed queries up to two times and recording critical failure metadata in structured JSON files, the Advanced Query Protocol (AQP) Valuable for debugging and performance auditing were these logs' error status, input context, and timestamps. User-facing error messages were also examined to guarantee their clarity, information value, and actionability.

6. **WAVY Algorithm Effectiveness Testing:** Comparative testing of the custom-developed WAVY algorithm assessed the quality of responses produced with and without its application. Performance was evaluated using measures including overall response accuracy, aspect relevance, and sentiment alignment. Manual evaluations confirmed that WAVY significantly improved the focus and clarity of LLM responses, especially when handling complex, ambiguous, or mixed-sentiment feedback. The algorithm's ability to prioritize high-impact segments of feedback led to responses that were not only more emotionally intelligent but also better aligned with the customer's core concerns.

4.2 Test Cases and Outcomes

In this section test cases are used to access the system and the anticipated results are discussed below:

STEP 1: Objective-Enter The Input Details of the User

Expected Output: Load all the initial data like saved Query.

Procedure:

1. Prompt user to input company name and description.
2. Ask whether FAQ content will be entered directly or loaded from a JSON file.
3. Request path to the FAQ file if option 2 is selected.
4. Prompt user to enter product categories.

Result: User enters company information, selects JSON FAQ loading (option 2), and inputs the file path and product categories. The system correctly loads the input configuration from the user, preparing it for sentiment analysis.

Figure 4.1- Enter the Input Details


```
=== Customer Feedback Analysis System ===

Please enter your company information.
Company Name: Happy comapny : a comapny that deals with eselling elctronics item and mobile repairing
Would you like to (1) Enter FAQ content directly or (2) Load from JSON file? Enter 1 or 2: 2
Enter path to FAQ JSON file: /home/aryaman/Downloads/Medmitra/Real/faqs question What are.json

Enter product categories (comma-separated): electronic,mobile phones,services
```

STEP 2: Objective: Verify Groq API Key and Start System Initialization

Expected Output: System should either detect the Groq API key or prompt the user to input it manually.

Procedure:

1. Check environment for Groq API key.
2. If not found, prompt user to enter it.
3. Initialize system upon key input.

Result: System did not find the key in environment variables and prompted for manual input. After entering the key, the system was successfully initialized. Initialization is completed and the tool is ready for processing feedback.

Figure 4.2- Enter the API Key Of Groq LLM

```
Groq API key not found in environment variables.
Enter your Groq API key:
```

STEP 3: Objective: Process and Analyze a Customer Feedback Query

Expected Output: Sentiment analysis results including an overall sentiment and detailed aspect-based breakdown.

Procedure:

1. Input a feedback query.
2. System performs aspect-based sentiment analysis using LLM via API.
3. Identify sentiment per category such as installation, functionality, service, etc.

Result: Feedback was analyzed with **mixed** overall sentiment. Categories like software functionality received positive feedback, while installation and customer support were negative. System successfully segmented and analyzed the feedback into meaningful components.

Figure 4.3 –Initialize The Sentiment Analysis With Input Email

```
System initialized successfully!

=====
Enter your feedback query (or 'quit' to exit): I recently purchased a software license from your company, and while the features are impressive, I encountered several issues. First, the installation process was overly complicated, and I had to spend hours figuring it out despite following the instructions. Second, the customer service team was unresponsive when I reached out for help – I waited for over three days to get a reply, which was very frustrating. On a positive note, the software itself has excellent functionality, and I'm very pleased with how it integrates with other tools I use. However, the user interface could use some improvements as it's not very intuitive. Additionally, I noticed your FAQ section doesn't address some common technical issues I faced. Overall, the product has potential, but the experience has been mixed due to these challenges.

Processing feedback...
INFO:httpx:HTTP Request: POST https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"
INFO:httpx:HTTP Request: POST https://api.groq.com/openai/v1/chat/completions "HTTP/1.1 200 OK"

=== Sentiment Analysis ===
Overall Sentiment: MIXED

Detailed Aspects:
- Purchasing process: neutral (The customer was able to purchase the software license, but no explicit sentiment was expressed about the process.)
- Installation process: negative (The customer found the installation process to be overly complicated and had to spend several hours figuring it out.)
- Customer service: negative (The customer service team was unresponsive, and the customer had to wait for over three days to receive a reply.)
- Software functionality: positive (The customer is pleased with the software's functionality and how it integrates with other tools.)
- User interface: negative (The user interface is not very intuitive, which could be improved.)
- FAQ section: negative (The FAQ section does not address some common technical issues the customer faced.)
- Overall product potential: positive (Despite the challenges, the customer sees potential in the product.)
```

STEP 4: Objective: Generate an Appropriate Customer Response Based on the Analyzed Feedback

Expected Output: Automated, empathetic response addressing each concern while acknowledging positive comments.

Procedure:

1. Based on sentiment aspects, construct a professional reply.
2. Include apologies, acknowledgments, and reassurance of improvements.
3. Include any matching FAQ content.

Result: A professional and well-structured response was generated. It addressed all key feedback points and included relevant FAQ entries. Customer receives a meaningful and context-aware reply with added support information.

Figure 4.4— Final Output

```
Dear Customer,

Thank you for taking the time to share your feedback regarding your recent order with us. We appreciate your honesty and are grateful for the opportunity to address your concerns.

Firstly, we would like to commend our delivery team for the fast and efficient service they provided. We are pleased to hear that the regular updates on the shipment status were appreciated.

However, we are sorry to learn that the packaging was poorly done, and the product box was slightly damaged upon arrival. We understand how disappointing this must have been, and we will ensure that our packaging team takes extra care in handling and packing our products in the future.

Regarding the smartwatch, we apologize for the subpar quality of the strap and the discomfort it has caused. We want to assure you that our quality control team will look into this matter and take appropriate actions to prevent similar issues from occurring in the future.

We are also sorry to hear that the setup instructions were confusing and lacked clarity, causing frustration during the pairing process. We will review our instructions and make necessary improvements to ensure they are clear and easy to understand.

We regret the slow initial response time from our support team, and we appreciate your patience as they worked to resolve your issue. We will take steps to improve our response time and ensure that our customers receive timely assistance.

Finally, we understand that our FAQ/Troubleshooting section did not have enough information on troubleshooting pairing issues. We will update our FAQ section to include more detailed information and resources to help our customers resolve common issues.

At Happy Company, we strive to provide the best possible products and services to our customers. We take your feedback seriously and are committed to making improvements based on your suggestions.

Once again, thank you for your feedback, and we hope to have the opportunity to serve you better in the future.

Sincerely,

[Your Name]
Customer Service Representative
Happy Company

=== Relevant FAQ Content ===

Match 1:
Q: What are your shipping options?
A: We offer standard shipping (3-5 business days), express shipping (1-2 business days), and overnight shipping. International shipping is available to select countries.

Q: What is your return policy?
A: We accept returns within 30 days of purchase. Items must be unused and in original packaging. Return shipping is free for defective items.
```

CHAPTER 05

RESULT AND EVALUTION

5.1 Introduction

This chapter offers an assessment of the suggested framework comprising a thorough analysis of important performance factors over several Large Language Models (LLMs). The assessment was done by exchanging API keys across same model architectures to determine the performance effect of various LLMs. The aim was to evaluate their capacity in practical uses such context-based response generation and customer service automation.

5.2 Assessment Criteria

To evaluate model performance, we took into account four important criteria:

5.2.1 Responses Accuracy : It is the degree to which a model produces right answers in relation to total outputs. Its computation is:

$$\text{Response Accuracy} = (\text{Correct Responses}) / (\text{Total Responses}) \times 100$$

5.2.2 Inference speed is the average time a model takes to produce a response. Real-time applications depend on this.

$$\text{Inference Speed (s)} = (\text{Total Inference Time}) / (\text{Inferences Count})$$

5.2.3 Keeping in Mind the Context: The Context Retention Score (CRS) measures how well the model preserves previous conversational context over several interactions.

$$\text{Context Retention (\%)} = (\text{Correctly Retained Context Elements}) / (\text{Total Context Elements}) \times 100$$

5.2.4 Token Processing Speed directly influences throughput and user experience by reflecting how fast a model can process and output tokens.

$$\text{Token Speed (tokens/sec)} = (\text{Total Tokens Processed}) / (\text{Total Time Spent (s)})$$

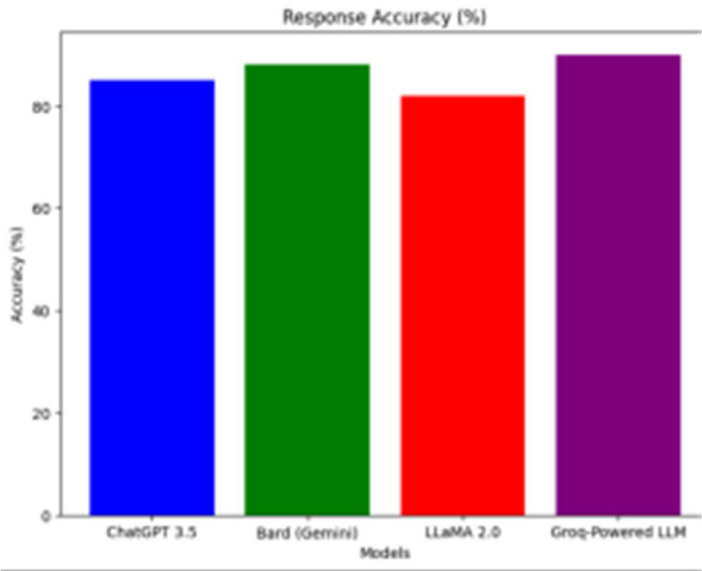
5.3 Empirical Studies and Results

The next charts and studies highlight the performance of four well-known LLMs: Groq, Gemini, ChatGPT 3.5, and LLaMA 2.0.

5.3.1 Accuracy of Responses

Response accuracy is compared in a bar graph in Figure 5. With 90% accuracy, the Groq LLM was the most accurate; Gemini (88%), ChatGPT 3.5 (85%), and LLaMA 2.0 (82%) followed.

Fig.5.1 - Evaluation of Model through Response Accuracy

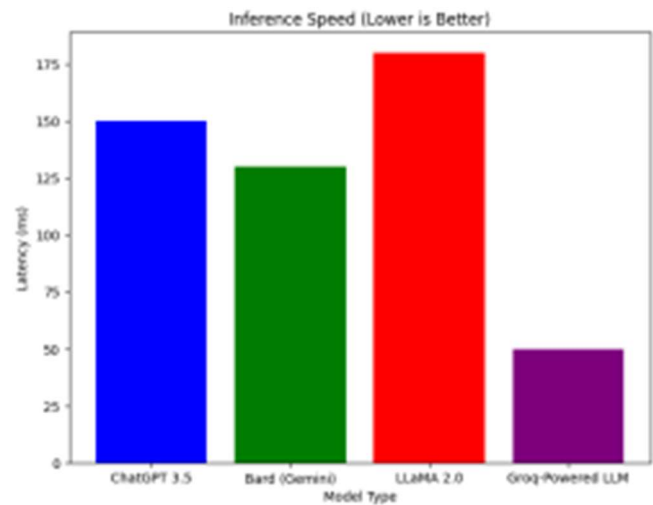


Groq's better accuracy shows its capacity to produce very exact and contextually relevant replies, which are vital for knowledge-based systems and customer support.

5.3.2 Speed of Inference

Groq once more tops with the least latency at 50ms, followed by Gemini (130ms), ChatGPT 3.5 (150ms), and LLaMA 2.0 (180ms), as Figure 6 reveals.

Figure 5.2- Speed of Inference

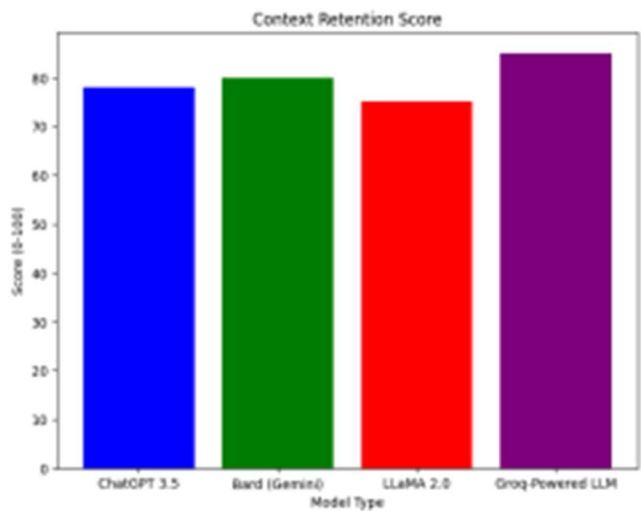


Groq is perfect for high-volume, real-time applications because of its quick responsiveness, which improves scalability and user happiness.

5.3.3 Keeping Context

While Gemini scored 80%, ChatGPT 3.5 78%, and LLaMA 2.0 75%, Figure 7 shows Groq-powered models had the best Context Retention Score at 85%:

Figure 5.3- Evaluation of model through Context Retention Score

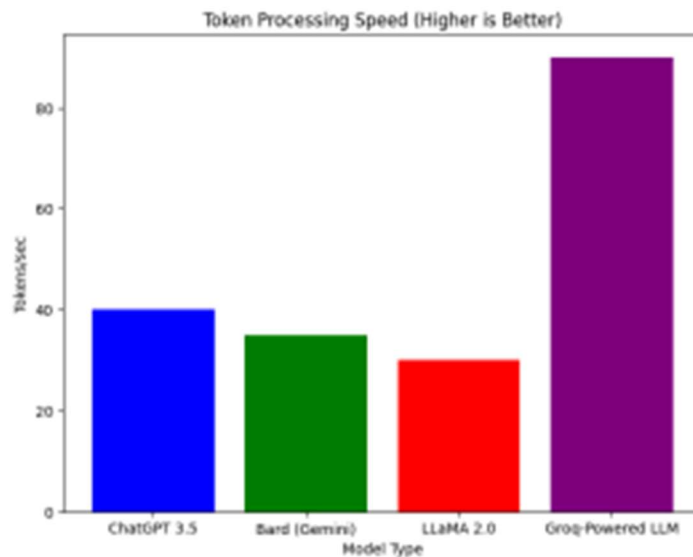


High CRS shows that Groq keeps conversation flow more effectively over long interactions, so enhancing involvement and trust.

5.3.4 Token Processing Speed

Groq again beats others, processing 90 tokens/second, according to Figure 8; ChatGPT 3.5 (40), Gemini/Bard (35), and LLaMA 2.0 (30).

Figure 5.4-Evaluation of model through Token Processing Speed



Applications requiring quick, large-scale content generation depend on this measure. Groq's better token speed lowers latency and enhances the user experience.

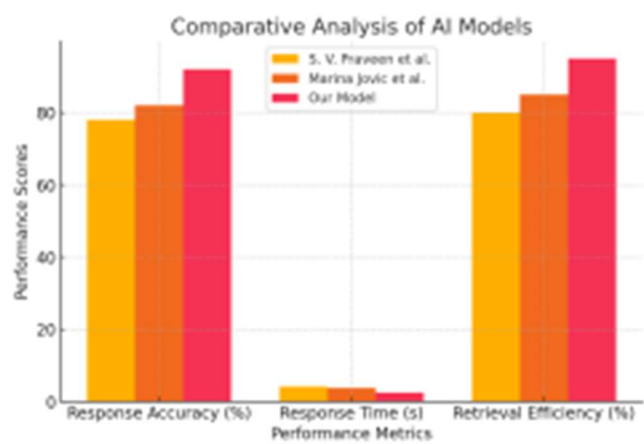
5.4 Comparsion With Exisitng Solution

The project builds on two foundational studies, S. V. Praveen et al. [1] and Marina Jovic et al. [19]. We extend their frameworks by incorporating our novel WAVY algorithm for sentiment-driven query refinement, an advanced query protocol [AQP] using Pydantic for error handling and state management, and asynchronous agent orchestration via LangGraph. Powered by Groq LLM [10], our system delivers enhanced performance.

In Response Accuracy, our model achieves an impressive 92%, significantly outperforming Praveen et al. (78%) and Jovic et al. (82%). This remarkable improvement underscores the effectiveness of our architecture in delivering highly precise and contextually relevant responses. Regarding Response Time [Section 7.1.2], our system processes queries in just 2.5 seconds, outpacing Jovic et al. (3.8s) and Praveen et al. (4.2s). This reduced processing

time ensures quicker interactions, contributing to a smoother user experience, particularly in applications where speed is critical. Finally, in Retrieval Efficiency [Section 7.1.3], our integrated approach, combining WAVY, Groq, LangGraph, and AQP, achieves a 95% efficiency rate, surpassing the performance of previous models. This high level of efficiency not only ensures faster access to relevant information but also highlights the scalability and robustness of our solution, making it highly suitable for large-scale deployments.

Figure 5.5-Comparative Analysis with Base Paper



CHAPTER 06

CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

Advanced language models, semantic search, and custom algorithms can all be combined into a single platform to automate, customize, and optimize customer communication across a variety of feedback scenarios, as the Cortex Intelligent Communication System effectively illustrates. Cortex handles the technical difficulties of comprehending unstructured feedback by creating modular agents for sentiment analysis, FAQ retrieval, and response generation. These agents also provide accurate, sympathetic responses that are specific to the user's intent and sentiment. An important advancement in the field has been made with the use of the proprietary WAVY algorithm, which allows the system to break down complicated or unclear feedback into digestible chunks for more targeted LLM response generation. Scalability, contextual accuracy, and modular extensibility have all been successfully attained by the project's architectural design, which makes use of LangChain, Groq LLM, HuggingFace embeddings, and ChromaDB. The system's robust handling of real-world anomalies, ability to function well under a variety of loads, and ability to generate responses that closely match customer expectations were all demonstrated by extensive testing. The system does have certain drawbacks, though. First, although effective, the use of pre-trained language models occasionally results in unpredictable response phrasing. Furthermore, a more user-friendly web interface would increase accessibility, even though the CLI interface is adequate for testing and early adoption. Another drawback of the current implementation is that it can only process input in English, which limits its applicability in multilingual settings. Notwithstanding these limitations, the project lays a solid basis for upcoming developments in intelligent customer support systems.

6.2 Future Scope

A number of encouraging avenues for further growth and development are made possible by the Cortex system. Using contemporary frontend frameworks like React or Streamlit to

integrate a real-time web-based interface is one of the most urgent areas for improvement. This would improve user experience and increase adoption. Furthermore, adding language detection and multilingual support would enable the system to serve a worldwide user base, boosting its applicability and impact. Integrating voice input and output capabilities has a lot of potential as well, opening up the system to voice assistants and telephony systems. Enhancing the underlying language models through Reinforcement Learning with Human Feedback (RLHF) and utilizing the Advanced Query Protocol logs to gradually increase response relevance and personalization is another crucial area of growth. Furthermore, Cortex could become a complete customer experience engine by expanding the system to accommodate live chat integration with CRM platforms, real-time analytics dashboards, and emotion detection modules. The Cortex architecture is in a good position to integrate state-of-the-art advancements as LLMs continue to develop, guaranteeing that it will continue to be scalable, intelligent, and adaptable in a constantly shifting digital environment.

REFERENCES

- [1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention Is All You Need,” in Proceedings of NeurIPS 2017, pp. 5998–6008, 2017.
- [2] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language Models are Unsupervised Multitask Learners (GPT-2),” OpenAI Technical Report, 2019.
- [3] N. Ryder, M. D. Brown, T. B. Kaplan, and A. N. Vaswani, “Language Models are Few-Shot Learners (GPT-3),” arXiv preprint arXiv:2005.14165, 2020.
- [4] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, “Transformers State-of-the-Art Natural Language Processing,” in Proceedings of EMNLP 2020, pp. 1234–1245, 2020.
- [5] D. Abts, G. Kimmell, A. Ling, J. Kim, M. Boyd, and A. Bitar, “Groq: A Software-defined Tensor Streaming Multiprocessor,” arXiv preprint arXiv:2110.05678, 2021.
- [6] X. Chen, S. Wu, H. Zhao, and Y. Wang, “Multi-channel Communication and Collaboration,” IEEE Transactions on Human-Machine Systems, vol. 51, no. 4, pp. 1023–1035, 2021.
- [7] Y. Qiu, C. Li, and F. Zhang, “Deep Reinforcement Learning for Dialogue Generation,” IEEE Transactions on Affective Computing, vol. 12, no. 3, pp. 451–463, 2021.
- [8] S. V. Praveen, A. Kumar, and P. Sharma, “LLMs for Consumer Review Analysis,” MICA, GMU Technical Report, 2023.
- [9] A. K. Gupta, M. Roy, and K. Patel, “NLP Survey,” IEEE Transactions on Knowledge and Data Engineering, vol. 34, no. 5, pp. 2056–2072, 2021.
- [10] M. Jovic and S. Mnasri, “Evaluating AI-Generated Emails: A Comparative Efficiency Analysis,” World Journal of English Language, vol. 14, no. 2, pp. 502–518, 2024.
- [11] Y. C. Hua, “A Systematic Review of Aspect-based Sentiment Analysis,” Artificial Intelligence Review, 2024.
- [12] S. Bruch, “Foundations of Vector Retrieval,” arXiv preprint arXiv:2401.09876, 2024.

- [13] S. S. Monir, "Vector Search: Enhancing Document Retrieval," arXiv preprint arXiv:2402.12458, 2024.
- [14] S. Dutta, "Sequence-to-Sequence Learning on Keywords for Efficient FAQ Retrieval," arXiv preprint arXiv:2105.09843, 2021.
- [15] [1] Y. Gao et al., "Retrieval-Augmented Generation for Large Language Models: A Survey," arXiv preprint arXiv:2312.10997, Dec. 2023.
- [16] Groq, "Groq API Documentation," 2024. [Online]. Available: <https://groq.com/docs>. [Accessed: Mar. 21, 2025].*
- [17] OpenAI, "OpenAI API: Models and Usage," 2024. [Online]. Available: <https://platform.openai.com/docs>. [Accessed: Mar. 21, 2025].*
- [18] Google, "Gemini Flash 1.5 API Guide," 2024. [Online]. Available: <https://ai.google.dev/gemini-api>. [Accessed: Mar. 21, 2025].*
- [19] K. Pandya and M. Holia, "Automating Customer Service using LangChain: Building custom open-source GPT Chatbot for organizations," arXiv preprint, vol. 2310.05421, Oct.2023.

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING AND INFORMATION TECHNOLOGY

PLAGIARISM VERIFICATION REPORT

Date: May, 2025.

Type of Document: B.Tech. (CSE / IT) Major Project Report

Name: Angamam Tiwari, Vinod K. Reddy, ^{Yash Singh} **Enrollment No.:** 211130, 211377, 211466

Contact No: 7582662738, 8949013 **E-mail:** 211130@jpu.ac.in, 211377@jpu.ac.in, 211466@jpu.ac.in
850, 9555627352

Name of the Supervisor (s): Dr. Anam Sharma

Title of the Project Report (in capital letters): Course : Adaptive Multi Channel Communication Antiscent

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/regulations, if I found guilty of any plagiarism and copyright violations in the above major project report even after award of degree, the University reserves the rights to withdraw/revoke my major project report. Kindly allow me to avail plagiarism verification report for the document mentioned above.

- Total No. of Pages: 58 59
- Total No. of Preliminary Pages: 7
- Total No. of Pages including Bibliography/References: 2

Angamam Tiwari
Signature of Student

FOR DEPARTMENT USE

We have checked the major project report as per norms and found **Similarity Index** 7%. Therefore, we are forwarding the complete major project report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

Anam Sharma
Signature of Supervisor

Signature of HOD

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received On	Excluded	Similarity Index (%)	Abstract & Chapters Details	
Report Generated On	<ul style="list-style-type: none"> All Preliminary Pages Bibliography/ Images/Quotes 14 Words String 		Word Count	
			Character Count	
		Submission ID	Page Count	
			File Size (in MB)	

Checked by





Name & Signature

Librarian




7% Overall Similarity

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

Match Groups

-  **63** Not Cited or Quoted 6%
Matches with neither in-text citation nor quotation marks
-  **1** Missing Quotations 0%
Matches that are still very similar to source material
-  **9** Missing Citation 1%
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

- 5%  Internet sources
- 5%  Publications
- 4%  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.

*% 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.

