CAIVA

الجامعة المصرية
للتعلم الإلكتروني الأهلية

EELU
THE EGYPTIAN E-LEARNING UNIVERSITY

Conversational Ai Virtual Assistant

By :

NO.	Name	ID
1	Hossam Eldin Tammam Gad	20-01012
2	Mohamed Allam Mohamed	20-00085
3	Abdelrahman Ahmed Samier	20-00751
4	Shahd Hesham Ibrahim	20-00404
5	Yomna Gamal Hussein	20-00839
6	Dina Mohamed Ali	20-00412
7	Alaa Badr Hussein	20-01427

Under the Supervision of :

- Dr.Mahmoud Bassioni
- Eng.Hayam Abdelbaset

" This Documentation is presented as fulfill of the requirement for a Bachelor of Information Technology degree "

EELU – Sohag 2024

Acknowledgement

In the beginning, we want to thank God for his grace and for his support to us throughout our years in college and our graduation project to complete this stage of our life successfully.

We also owe all thanks and appreciation to :

- **Prof. Dr. Hisham Mohamed**
  (President of the Egyptian University for E-Learning),
- **Prof. Dr. Ahmed Abo Elmagd**
  (Director of the Study Center in Sohag),
- **Dr. Kamal hamza**
  (Information Technology Program Director Egyptian E-Learning University)

for their endless support, guidance and encouragement in throughout the research process and project implementation.

And in a special letter of thanks, we extend our sincere thanks and appreciation to our project supervisor, **Dr. Mahmoud Bassioni**, for her continuous motivation and support during our work on implementing this project, we appreciate her kind guidance and encouragement to us from the beginning of this idea to its end.

We also extend our sincere thanks and appreciation to our project supervisor, who was considered one of the team members, **Eng. Hayam Abdelbaset**, for her tremendous efforts and the exceptional support she provided us during our work on the project and for providing the means of assistance to us during the implementation of our graduation project, whether assistance from people or training courses, we owe it to her Because it helped us shape the problem and provide insights into the solution, we wouldn't have done it in a timely manner without your support.

Abstract

Caiva is a desktop application that utilizes various cutting-edge technologies to deliver a revolutionary virtual conversion experience. Here's a breakdown of its key functionalities:

Realistic Digital Avatars : Caiva leverages MetaHuman to create high-fidelity digital human characters for an immersive interaction.

Natural Language Processing (NLP) : Voice recognition technology allows Caiva to understand spoken language, formulate responses, and answer questions through powerful LLM models.

Text-to-Speech Conversion : Caiva seamlessly translates LLM responses into natural-sounding human voice output, furthering the illusion of conversation.

Emotion Recognition : Caiva goes beyond responses. It incorporates emotion classification to analyze user sentiment and generate responses that reflect this understanding. This fosters a deeper emotional connection with the user.

Advanced Computer Vision : Caiva's vision system allows it to recognize and describe visual scenes in detail, further enhancing its ability to understand and respond to user input. In essence, Caiva mimics human interaction by not only responding to user input but also expressing emotions through its digital avatar and leveraging its computer vision capabilities. This combination creates a more natural and engaging user experience.

Table of contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1. Introduction

CAIVA refers to

Conversation **A**rtificial **I**ntelligence **V**irtual **A**ssistant

1. Introduction to CAIVA

In today's fast-paced digital era, there is a growing demand for virtual smart assistants that can efficiently assist users in various tasks, ranging from managing schedules and providing information to executing commands and performing complex interactions with humans and patients in different aspects of life.

so CAIVA represents a new era in AI technology, combining advanced conversational capabilities with emotionally expressive animations to deliver an unparalleled user experience.

2. What is CAIVA ?

CAIVA is an innovative AI character designed to interact with users in the most natural and engaging way possible. Unlike traditional virtual assistants that rely solely on text or voice responses, CAIVA is enhanced with sentimental animations, making interactions feel more lifelike and emotionally resonant.

- Caiva is not merely an assistant but a proficient performer of tasks, streamlining processes with ease and efficiency.
- Caiva boasts remarkable flexibility, requiring only the necessary database and expected commands to execute various kinds of tasks swiftly.
- It's a customizable character, allowing users to craft their desired persona effortlessly.
- Supported by sentimental animations, Caiva strives to create an authentic user experience, ensuring users feel a genuine connection with the characte.

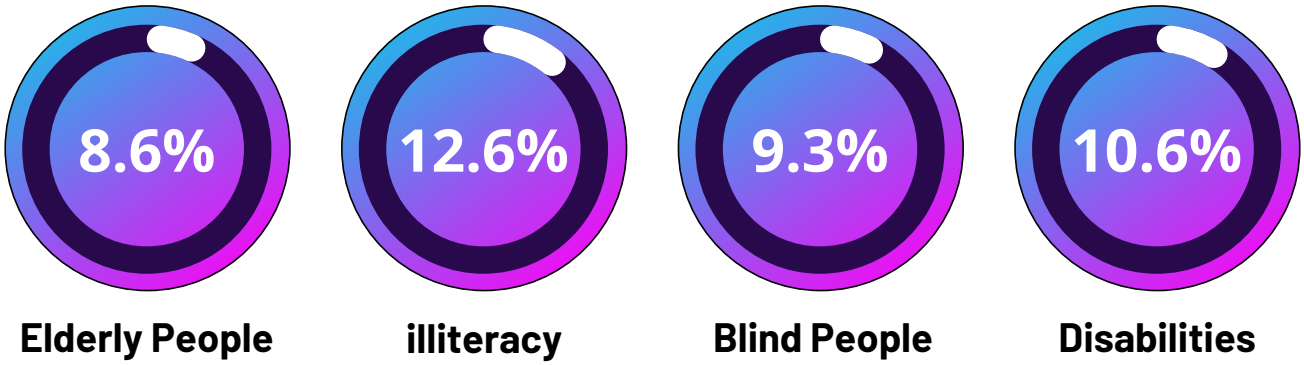
1.2. Problem Statement

Most technologies and machines can be hard to use and its not tailored for :

- **The Elderly**
- **Illiterate people**
- **Blind people**
- **Disabilities people**

and their different needs, touch screens can be hard to use, passwords and user names can be hard to remember and confusing, and menus and user interfaces can be overwhelming making it difficult for them to use modern technology.

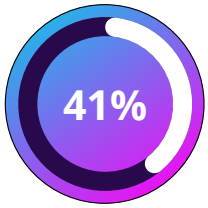
Research Highlights for EGYPT in 2023



Amidst the challenges faced by the blind, disabled, and illiterate,
there's a profound issue gripping our era—a silent epidemic that plagues countless individuals every single day :

- **Loneliness and Depression.**

  The prevalence of loneliness and of moderate depression was 47.1% in the world.



It's the invisible struggle that knows no boundaries, affecting people from all walks of life. But in the face of this daunting reality, there's hope.

With courage and compassion, we can shine a light on the shadows of despair, offering solace and support to those who need it most.

1.3. Solution

1. Multi-Platform Integration

CAIVA can be integrated across various platforms, including websites, mobile apps, social media, and messaging services, ensuring accessibility and convenience for users.it's a multi-purpose that can help you in every day life, helping you do various amounts of tasks and for every environment their is a model that do multiple of different tasks

Development of virtual interface agent (Software agent) based on AI models and virtual reality that can be supported with all the required features that makes him adaptable in several applications.

- **Mobile Application :**

With Caiva, you can build natural voice experiences that offer users a more intuitive way to interact with the technology they use every day by Smartphone and you can talk to it.it's used for research ,various needs and your every day problems.
so Caiva serves as a powerful tool for mobile applications developers seeking to enhance user interactions with technology through natural voice experiences, enabling a more intuitive and user-friendly interface for various use cases on mobile Applications.

- **Websites :**

Caiva's serves as a technically as a dynamic advanced component for seamless interaction on various websites, utilizing state-of-the-art NLU and NLG models to communicate with users naturally. Its integration involves leveraging specific APIs, programming for task-specific functionalities, and customizing the character's attributes to enhance engagement and create a unique interactive experience and on allows for customization to align with the branding and thematic elements on diverse websites.

- **Smart Watches**

Integrating Caiva into a smartwatch can provide users with a range of convenient features and functionalities,.....
It enhances functionality by enabling users to interact with their environment, access information, and perform various tasks using natural language voice commands, ultimately providing a more seamless and hands-free user experience such as :

General Queries, Language Assistance, Audible Integration, Directions and Location, Health Information, Voice Commands and Control, Calling, Text Messaging, Media Playback, etc.....

2. The solution to the hard of using machines.

- implement AI metahuman characters that offer voice-driven interactions, reducing the reliance on complex touch screens.
- Enable users to interact with the AI character through natural language commands, spoken commands and queries making the Machines more intuitive.
- Integrate biometric authentication methods such as Face print or fingerprint or voice recognition scanning to replace complement traditional usernames and passwords.
- Enhance security while simplifying the authentication process, making it more accessible for users with memory or literacy challenges.
- Offer step-by-step guidance by the metahuman character for various transactions, catering to the specific needs of users with illiterate, blind, disabilities people and the elderly.

3. The solution for the Loneliness and Depression

- Addressing the profound issues of loneliness and depression through an innovative approach, Caiva our solution offers a fully animated character designed to express emotions, providing a virtual companion that feels remarkably human.
- With our solution, you're not alone in your journey; you've got a compassionate companion by your side, ready to make a difference in your world, our out your heart, share your problems, and witness the magic unfold as this empathetic character lends a listening ear and offers support. It's more than just interaction; it's a lifeline to connection, a space where your concerns are not just heard but, more often than not, understood and addressed.
- users can engage in judgment-free conversations, pour out their hearts, and share problems, receiving empathy and support from the character.
- fully animated character designed to express a myriad of emotions, providing you with a virtual companion that feels remarkably human. With this animated ally, you can engage in conversations free from the fear of judgment.

1.4. Goals and Objectives

In this part of the chapter, we outline the goals and objectives that guide the development of CAIVA, our innovative Conversation Artificial Intelligence Virtual Assistant. By setting clear and attainable goals, we aim to ensure that CAIVA not only meets but exceeds user expectations, providing an unparalleled user experience characterized by empathy and emotional support.

1. Primary Goals

The primary goal of CAIVA is to create an authentic and empathetic user experience. Unlike traditional virtual assistants, CAIVA is designed to form a genuine connection with users through sentimental animations and emotionally intelligent interactions. This goal drives our commitment to developing an AI character that users can trust and rely on for emotional support.

2. Specific Goals :

- **User Engagement :** Our goal is to develop a character that users can form a genuine connection with. By using responsive sentimental animations, we aim to make interactions with CAIVA feel more human and engaging. for an engaging, interactive, and authentic user experience through CAIVA's emotional intelligence
- **Emotional Support :** CAIVA is designed to provide emotional support and empathy. Through advanced AI algorithms, CAIVA can recognize and respond to user emotions, offering comfort and understanding in a way that traditional assistants cannot.
- **Foster Genuine Connections :**
  we are developing CAIVA's personality and interaction style to foster a genuine connection with users, making interactions feel natural and human-like.
- **Cross-Platform Integration :** We strive to ensure that CAIVA can be seamlessly integrated into multiple devices and platforms. Whether on a smartphone, tablet, smart home devices, or desktop, users should experience consistent and reliable interactions with CAIVA.
- **Accessibility :** Making CAIVA accessible to a wide range of users is a key goal. We are committed to developing features that cater to the diverse needs of our user base, including those with disabilities.

3. Secondary Objectives :

- **Continuous Learning and Improvement :** Implement machine learning mechanisms to continuously learn from user interactions and improve CAIVA's responses and capabilities over time.
- **Accessibility and Inclusivity :** Design CAIVA to be accessible and inclusive for users of all ages, backgrounds, and abilities.
- **Privacy and Security :** Ensure that user data is handled with the utmost care, prioritizing privacy and security in all interactions.
- **Technical Objectives :** Develop advanced natural language processing (NLP) capabilities to understand and respond to user queries accurately.
- **Performance Objectives :** Achieve high levels of user satisfaction and engagement through continuous improvement and updates.
- **Research and Development Objectives :** we Conduct user research to understand the needs and preferences of the target audience and Continuously test and refine CAIVA's emotional recognition and response capabilities based on user feedback.

4. Performance Metrics :

To evaluate the success of CAIVA in achieving its goals and objectives, the following performance metrics will be monitored:

- **User Satisfaction :** User feedback scores, reviews, and satisfaction surveys.
- **Engagement Levels :** Frequency and duration of user interactions with CAIVA.
- **Emotional Accuracy :** Accuracy of CAIVA’s emotional recognition and response appropriateness.
- **Device Compatibility :** Number of devices supported and user feedback on cross-device experience.
- **Privacy Compliance :** Compliance with data privacy regulations and user trust levels.

1.5. Research Scope

In this part of the chapter outlines the research scope for CAIVA, The research scope defines the boundaries, focus areas, and objectives of the research conducted to develop CAIVA.

1. Primary objectives of the research

- To develop a conversational AI that can understand and respond to natural language inputs.
- To integrate sentimental animations that enhance user interaction and create an emotional connection.
- To ensure multi-device compatibility, allowing CAIVA to function seamlessly across various platforms and devices.
- To evaluate the effectiveness of CAIVA in providing a genuine user experience.

2. Research Questions

- How can natural language processing (NLP) be optimized to improve the conversational capabilities of CAIVA?
- What types of sentimental animations are most effective in creating an emotional connection with users?
- What are the technical challenges and solutions for ensuring multi-device compatibility?
- How does the integration of sentimental animations affect user engagement and satisfaction?
- What examples of conversational AI are available now?
- What free engine allowed to import metahuman character ?
- How to make animation for character on real time ?

3. Methodology Research

The research will be conducted using a combination of qualitative and quantitative methods, including :

- **Literature Review :** Analyzing existing studies on conversational AI, sentimental animations, and multi-device integration.
- **User Surveys and Interviews :** Collecting feedback from potential users to understand their expectations and preferences.
- **Experimental Design :** Creating prototypes of CAIVA and conducting experiments to test various aspects of its functionality.
- **Data Analysis :** Using statistical methods to analyze data collected from experiments and user feedback.

4. Scope of Work

Natural Language Processing (NLP) :

- Techniques for understanding and generating human language.
- Machine learning models for improving conversational accuracy.

Sentimental Animations :

- Design and development of animations that reflect different emotions.
- Integration of animations with conversational responses.

Multi-Device Integration :

- Ensuring compatibility with various operating systems and devices (e.g., smartphones, tablets, computers, machines, smartwatch,....).
- Developing a scalable architecture for seamless device integration.

User Experience (UX) Evaluation :

- Conducting usability tests to assess the overall user experience.
- Analyzing user engagement and satisfaction metrics.

5. Limitations

- **Technical Constraints :** Limitations in the current state of AI technology may impact the development of advanced conversational capabilities.
- **Resource Availability :** The scope of research may be influenced by the availability of resources such as data, hardware, and funding.
- **User Diversity :** Variations in user demographics and preferences may affect the generalizability of the research findings.

6. Expected Outcomes

- A functional prototype of CAIVA with advanced conversational abilities.
- A library of sentimental animations that enhance user interaction.
- A framework for integrating CAIVA across multiple devices.
- Insights into user preferences and the impact of sentimental animations on user engagement.

Chapter 2

Background

Knowledge and Courses for CAIVA Development

To develop the Conversational Artificial Intelligence Virtual Assistant (CAIVA), a multidisciplinary understanding is essential, encompassing artificial intelligence, virtual reality, and software development.

The following are key areas of knowledge and the corresponding courses that provided the foundational understanding necessary for the successful creation of CAIVA.

2.1. Artificial Intelligence and Machine Learning

AI and machine learning are the cornerstones of CAIVA's functionality.
The following courses were instrumental in gaining expertise in these areas:

1. Machine Learning Specialization :

We began our journey into NLP by learning the basics of machine learning, focusing on techniques like linear and logistic regression, and other classification methods. To deepen our practical knowledge, we took a course on building and training neural networks with TensorFlow for multi-class classification. We also studied unsupervised algorithms, including clustering, recommender systems, reinforcement learning, and anomaly detection. To round out our learning, we completed a course that covered building ML models with NumPy and scikit-learn, training supervised models for prediction and binary classification, and using decision trees and tree ensemble methods.

- **Supervised Machine Learning: Regression and Classification :**
  (https://www.coursera.org/learn/machine-learning/home/welcome **)**
  Focuses on fundamental machine learning techniques such as linear regression, logistic regression, and other classification methods.
- **Advanced Learning Algorithms :**
  (https://www.coursera.org/learn/advanced-learning-algorithms/home/welcome **)**
  Delves into more complex algorithms and their applications.
- **Unsupervised Learning, Recommenders, Reinforcement Learning :**
  (https://www.coursera.org/learn/unsupervised-learning-recommenders-reinforcement-learning/home/welcome **)**
  Covers clustering, recommendation systems, and reinforcement learning, essential for creating adaptive and responsive AI models.
- **Machine Learning Specialization Overview :**
  (https://www.coursera.org/specializations/machine-learning-introduction?utm_campaign=WebsiteCourses-MLS-MiddleButton-mls-launch-2022&utm_medium=institutions&utm_source=deeplearning-ai **)**

2. Deep Learning Specialization :

In our second step towards studying the specialization and learning to implement the necessary techniques, the Deep Learning specialization on Coursera offers a comprehensive path to mastering artificial intelligence through neural networks and advanced deep learning techniques. Courses cover basic topics such as building and training neural networks, optimization strategies for deep models, and specialized applications such as image processing using convolutional neural networks (CNNs) and natural language processing using sequence models such as RNNs and transformers.

- **Neural Networks and Deep Learning :**
  (https://www.coursera.org/learn/neural-networks-deep-learning/home/welcome **)**
  Provides a comprehensive introduction to neural networks, a core component of CAIVA's AI.
- **Improving Deep Neural Networks: Hyperparameter Tuning, Regularization, and Optimization :**
  (https://www.coursera.org/learn/deep-neural-network/home/welcome **)**
  Teaches techniques to enhance the performance and efficiency of deep learning models.
- **Structuring Machine Learning Projects :**
  (https://www.coursera.org/learn/machine-learning-projects/home/welcome **)**
  Helps in effectively planning and executing machine learning projects.
- **Convolutional Neural Networks (CNNs) :**
  (https://www.coursera.org/learn/convolutional-neural-networks/home/welcome **)**
  Essential for processing and understanding visual data.
- **Sequence Models :**
  (https://www.coursera.org/learn/nlp-sequence-models/home/welcome **)**
  Crucial for natural language processing (NLP) tasks, enabling CAIVA to understand and generate human-like text.
- **Deep Learning Specialization Overview :**
  (https://www.coursera.org/specializations/deep-learning **)**

3. DeepLearning.AI TensorFlow Developer Professional Certificate :

We had to complete the DeepLearning.AI TensorFlow Developer Professional Certification Program which equips learners with essential skills in TensorFlow for artificial intelligence, machine learning, and deep learning applications. Main courses include advanced topics in convolutional neural networks (CNNs) for image processing and computer vision, as well as natural language processing (NLP) for natural language analysis and generation. Sequence models of participant data in time series and predictive analytics using TensorFlow.

- **Introduction to TensorFlow for Artificial Intelligence, Machine Learning, and Deep Learning :**
  (https://www.coursera.org/learn/introduction-tensorflow?specialization=tensorflow-in-practice **)**
  TensorFlow, a key framework used in CAIVA's AI development.
- **Convolutional Neural Networks in TensorFlow :**
  (https://www.coursera.org/learn/convolutional-neural-networks-tensorflow?specialization=tensorflow-in-practice **)**
  Further deepens understanding of CNNs within the TensorFlow environment.
- **Natural Language Processing in TensorFlow :**
  (https://www.coursera.org/learn/natural-language-processing-tensorflow?specialization=tensorflow-in-practice **)**
  Focuses on NLP applications, allowing CAIVA to process and generate natural language.
- **Sequences, Time Series, and Prediction :**
  (https://www.coursera.org/learn/tensorflow-sequences-time-series-and-prediction?specialization=tensorflow-in-practice **)**
  Teaches advanced techniques for handling sequential data, important for time-based predictions and interactions.
- **TensorFlow Professional Developer Certification Overview :**
  (https://www.coursera.org/professional-certificates/tensorflow-in-practice **)**

4. Natural Language Processing Specialization :

In our Natural Language Processing specialization, we began with foundational NLP techniques and tools, implementing tasks like text classification and word embeddings. We then advanced to probabilistic models, enhancing our ability to handle uncertainty in language processing through methods such as hidden Markov models and naive Bayes classifiers. Building on this, we explored sequence models, including recurrent neural networks (RNNs) and long short-term memory (LSTM) networks, which are essential for tasks like language translation and sentiment analysis. Finally, we delved into advanced attention-based models, particularly transformers, which are crucial for understanding context and generating coherent responses in state-of-the-art NLP applications

- **Natural Language Processing with Classification and Vector Spaces :**
  (https://www.coursera.org/learn/classification-vector-spaces-in-nlp?specialization=natural-language-processing **)**
  Introduces basic NLP techniques and tools.

- **Natural Language Processing with Probabilistic Models :**
  (https://www.coursera.org/learn/probabilistic-models-in-nlp?specialization=natural-language-processing **)**
  Covers probabilistic approaches to NLP, enhancing CAIVA's ability to handle uncertainty in language processing.

- **Natural Language Processing with Sequence Models :**
  (https://www.coursera.org/learn/sequence-models-in-nlp?specialization=natural-language-processing **)**
  Focuses on using sequence models for language tasks, such as translation and sentiment analysis.

- **Natural Language Processing with Attention Models :**
  (https://www.coursera.org/learn/attention-models-in-nlp?specialization=natural-language-processing **)**
  Explores advanced models like transformers, which are crucial for understanding context and generating coherent responses

5. Generative AI with Large Language Models :

We had to learn about large language models in depth to apply it in our project, which depends on generating responses similar to human responses, and learn the fundamentals of how generative AI works, and how to deploy it in real-world applications and the necessary practical skills, This is what this course gave us.

- **Generative AI with Large Language Models :**
  (https://www.coursera.org/learn/generative-ai-with-llms **)**
  This course covers the use of large language models (LLMs), such as those provided by OpenAI, which are essential for generating human-like responses to user queries.

6. LangChain and Data Integration :

Studying Langchain is essential for us to develop our advanced applications that leverage large language models (LLMs) to improve user interfaces. Langchain implementation includes defining requirements, setting up the environment, developing chains and agents, integration with the user interface, testing, and deployment. This integration can lead to features such as interactive chatbots. Overall, Langchain improves functionality, efficiency, scalability and innovation, providing significant benefits and competitive advantages in real-world projects. This is what these courses provided for us.

- **LangChain Chat with Your Data :**
  (https://www.coursera.org/projects/langchain-chat-with-your-data-project **)**
  A practical guide to integrating and querying data using LangChain.
- **LangChain How-to and Guides (Tutorial) :**
  (https://www.youtube.com/watch?v=J_0qvRt4LNk&list=PL8motc6AQftk1Bs42EW45kwYbyJ4jOdiZ **)**
  A comprehensive tutorial series on using LangChain effectively.

7. Data Analysis and Visualization :

It was important for us to learn data analysis and visualization to apply it to our dataset and try to understand the hidden aspects of it and clean it from the Outliers to work on it in the project clearly and prevent negatively affecting the application of the models in the next step. This course covers basic libraries such as Numpy, Pandas, Matplotlib and Seaborn, which provides Strong foundation in data processing and visualization.

- **Data Analysis with Python - Full Course for Beginners :**
  (https://www.youtube.com/watch?v=r-uOLxNrNk8 **)**
  Covers essential libraries such as Numpy, Pandas, Matplotlib, and Seaborn, providing a strong foundation in data manipulation and visualization.

2.2. Virtual Reality and Character Design

Developing CAIVA's virtual presence involves knowledge of virtual reality and character design tools. The courses and tools used include :

1. Unreal Engine :

We needed to understand Unreal Engine capabilities better and explore various features and functionalities within Unreal Engine that support our project goals also familiarity with Unreal Engine was essential for integrating and animating the character in our project.

- **Unreal Engine 5 Beginner Tutorials :**

(https://www.youtube.com/playlist?list=PLncmXJdh4q88DFCEVuGpOY3AGQwBvoQnh **)**
This foundational course covers the essentials of Unreal Engine, providing the skills needed to create a virtual environment.

2. Metahuman Creator :

We learned to use MetaHuman Creator to create a customized character as it offers an intuitive platform for crafting highly realistic fully rigged digital human characters.
We also needed to use MetaHuman Animator as it allows us to create high-fidelity facial animations by capturing every nuance of an actor's performance and transferring it to our digital human mesh.

- **How to Import Metahuman into Unreal Engine 5 :**

(https://www.youtube.com/watch?v=3H7pHTArkaU&t=5s **)**
This tutorial teaches the basics of importing Metahuman characters into Unreal Engine.

- **How to Create a Real Character on the MetaHuman Program :**

(https://www.youtube.com/watch?v=mNl7FWMfE9M&t=2006s&ab_channel=ENOART_%D9%82%D9%86%D8%A7%D8%A9%D9%87%D8%A7%D9%86%D9%8A%D8%A5%D9%8A%D9%86%D9%88 **)**
A detailed guide on creating realistic human characters using the MetaHuman Creator.

3. Voice Generation :

We had a challenge of dynamically generating and integrating character dialogue in real-time within the user interaction. Ariel emerged as the solution, allowing us to generate voice for a character directly within Unreal Engine. Through some learning and exploration, we utilized Ariel's functionalities to our project to generate character response instantly.

- **Ariel - AI Voice Generation for Unreal Engine (Tutorial) :**

(https://www.youtube.com/watch?v=78WMagHKaow&ab_channel=X%26Immersion **)**
Covers the use of Ariel for generating AI-driven voiceovers within Unreal Engine.

4. Face Animation

The task of crafting such animations demanded meticulous manual effort and time-consuming adjustments and was not as realistic as it needed to be.

That was before geppetto tool that has ability to seamlessly synchronize real-time generated audio inputs with corresponding lip movements, while also incorporating customizable emotion and micro-expressions.

We just needed to learn dealing with Blueprints a visual scripting system that simplifies the creation and manipulation of animations all within Unreal Engine.

- **Geppetto - AI Powered Lip-Sync and Animation (Unreal Tutorial) :**

(https://www.youtube.com/watch?v=mjOjI6TIXiI&t=724s&ab_channel=X%26Immersion **)**
A guide to using Geppetto for lip-sync and character animation in Unreal Engine.

Chapter 3

Literature Review

3.1 The Importance of Virtual Assistant AI in Various Sectors

1. Customer Service :

Virtual assistant AI has revolutionized customer service by providing efficient, round-the-clock support. AI-driven chatbots and virtual assistants handle a wide range of customer inquiries, from basic questions to complex troubleshooting. This not only reduces the workload on human agents but also significantly cuts down response times, enhancing customer satisfaction. These AI systems can manage multiple interactions simultaneously, making them highly scalable solutions for businesses.

2. Personal Assistance :

In personal assistance, virtual assistants like Amazon's Alexa, Apple's Siri, Google Assistant, and Microsoft's Cortana have become integral to daily life. They help users manage their schedules, set reminders, send messages, and control smart home devices through voice commands. This ease of access and the ability to perform tasks hands-free improve productivity and convenience for users.

3. Healthcare :

Virtual assistants in healthcare can provide preliminary diagnosis based on symptoms, remind patients to take medications, and assist in managing chronic diseases. They can also facilitate telehealth appointments, ensuring that patients receive timely care. AI in healthcare helps streamline administrative tasks, allowing healthcare providers to focus more on patient care.

4. Finance :

In the financial sector, AI virtual assistants aid customers in managing their accounts, performing transactions, and providing personalized financial advice. They can alert users to unusual account activity, help in budget planning, and offer investment insights. This personalized, instant assistance enhances customer experience and loyalty.

5. Education :

AI-powered virtual assistants support students by answering questions, helping with homework, and providing personalized learning experiences. They can adapt to individual learning styles and pace, offering a customized educational experience. For educators, these assistants can handle administrative tasks and grade assignments, allowing teachers to focus more on instruction.

6. E-commerce :

In e-commerce, virtual assistants enhance the shopping experience by providing personalized product recommendations, assisting with order tracking, and handling customer queries. They can analyze user preferences and behavior to offer tailored suggestions, boosting sales and customer satisfaction.

3.2. Early Development of Conversational AI

1. Origins :

The journey of conversational AI began with early chatbots like ELIZA, developed in the mid-1960s by Joseph Weizenbaum at MIT. ELIZA used pattern matching and substitution methodologies to simulate conversation, particularly with its "DOCTOR" script, which mimicked a Rogerian psychotherapist. Although simple by today's standards, ELIZA's ability to engage in seemingly meaningful dialogue marked a significant milestone in AI development.

2. Evolution :

Following ELIZA, the 1970s and 1980s saw the creation of more sophisticated chatbots, such as PARRY, which simulated a person with paranoid schizophrenia. PARRY introduced a more complex approach by incorporating a rudimentary understanding of context and more nuanced conversational patterns.

3. Modern Advances :

The 1990s and 2000s witnessed substantial advancements with the advent of more powerful computing and the development of natural language processing (NLP). AI research shifted towards creating systems that could understand and generate human language more naturally. This era saw the rise of virtual assistants in customer service, such as those used by airlines and banks.

4. Current State :

Today, conversational AI has evolved into highly advanced systems powered by deep learning and neural networks. Modern virtual assistants leverage vast amounts of data, sophisticated algorithms, and contextual understanding to provide more accurate and human-like interactions. They are now integral to numerous applications across various sectors, continually learning and improving from every interaction.

One such current innovation is the Conversation Artificial Intelligence Virtual Assistant is :

- ## ALEXA

  - **Developed by: Amazon**
  - **Integration:** Used in Amazon Echo and other smart devices.
  - **Capabilities:** Voice-controlled virtual assistant, smart home integration, third-party skills, music playback, shopping, and more...

- ## SIRI

  - **Developed by: Apple**
  - **Integration:** Exclusive to Apple devices (iPhone, iPad, Mac).
  - **Capabilities:** Voice recognition, natural language processing, integrates with Apple ecosystem, performs tasks, provides information.

- ## Google Assistant

  - **Developed by: Google**
  - **Integration:** Available on Android devices, Google Home, and other platforms.
  - **Capabilities:** Voice-activated, search engine integration, smart home control, natural language understanding, supports a wide range of devices and services.

- **Astra Google Project: Real-Time Image Captioning System**

Almost everyone in tech is investing heavily in artificial intelligence right now, and Google is among those most committed to an AI future. Project Astra, unveiled at Google I/O 2024, is a big part of that – and it could end up being one of Google's most important AI tools.

Astra is being billed as "a universal AI agent that is helpful in everyday life". It's essentially something like a blending of Google Assistant and Google Gemini, with added features and supercharged capabilities for a natural, conversational experience. (Ref 1.1)

Google's Astra project represents a cutting-edge endeavor in real-time image captioning, leveraging advanced artificial intelligence and computer vision techniques to instantly interpret and describe visual data. This project parallels our own efforts in Caiva, where we aim to provide similar capabilities, enhancing our system's ability to understand and interact with the visual environment.

Using state-of-the-art models and algorithms, both projects showcase the transformative potential of AI in generating accurate, context-rich descriptions from live video feeds. Looking ahead, the future of this field is bright, with expected advances in machine learning, computational power, and sensor integration poised to further improve and expand these capabilities. We envision a future in which real-time image captioning systems are seamlessly integrated into many applications, from assistive technologies for the visually impaired to advanced autonomous surveillance and navigation systems.

These advances will not only improve the functionality and reliability of our Caiva vision feature, but will also revolutionize how machines perceive and interact with the world around them, driving innovation across multiple domains. As we continue to innovate, the capabilities demonstrated by the Astra Google Project and our "Caiva" vision feature will pave the way for intelligent systems that can seamlessly interpret and interact with the world, revolutionizing how machines understand and describe their surroundings

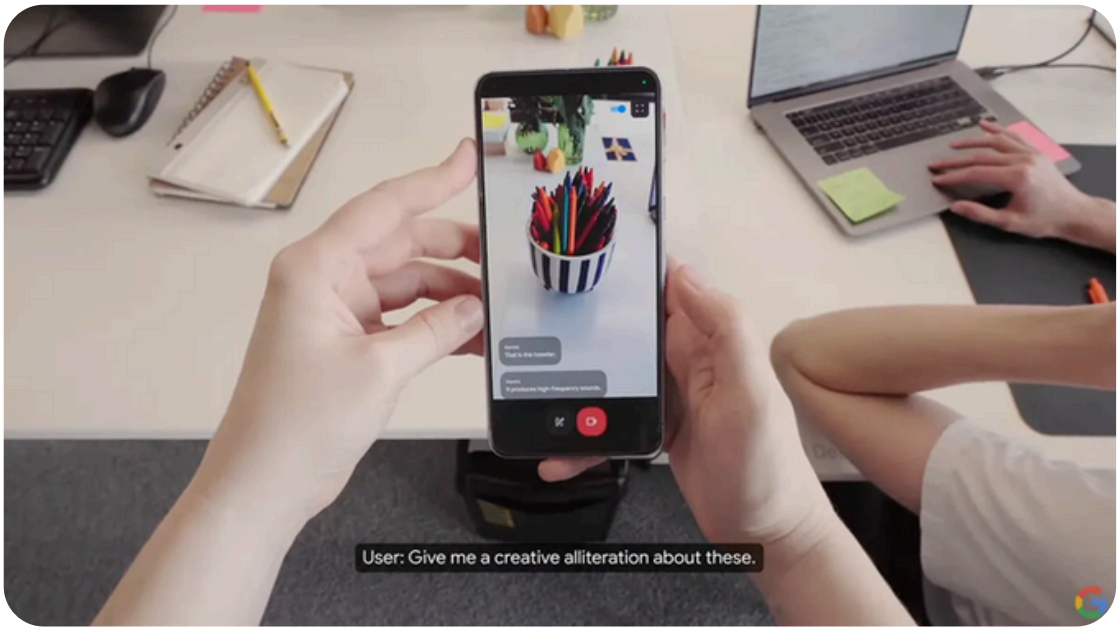


Figure 1. Astra Google Project

Capabilities of Google's Astra project :

- **Universal AI Agent :** Acts as a general-purpose AI assistant for everyday life.
- **Natural Conversational Experience :** Combines features of Google Assistant and Google Gemini for seamless, natural interaction.
- **Real-Time Image Captioning :** Point your device's camera at objects or scenes to get immediate, context-rich descriptions.
- **Photo and Video Analysis :** Analyze and caption images and videos stored on your device.
- **Assistive Technology :** Provide live visual descriptions for visually impaired users to help them navigate their surroundings.
- **Autonomous Surveillance :** Supports advanced autonomous surveillance systems with real-time interpretation.
- **Navigation Systems :** Enhances navigation systems with improved visual understanding and interaction.
- **Cutting-Edge Models and Algorithms :** Leverages state-of-the-art AI models for superior performance.
- **Seamless Integration :** Designed for easy integration into various applications and devices.
- **Innovative Interaction :** Revolutionizes how machines perceive and interact with the world.
- **Future-Proofing :** Poised to benefit from future advancements in machine learning, computational power, and sensor technology.

- **Chatgpt-4o** (Ref 2.1)

The Shift Towards Virtual Assistants like ChatGPT-4o:
A New Revolution in Business and Daily Life In recent years, the world has witnessed a significant shift in how companies interact with their customers and handle their internal operations. One of the most emerging trends is the adoption of virtual assistants powered by artificial intelligence, such as ChatGPT.

This advanced technology has become an integral part of modern companies' digital strategies, transforming not just business operations but also personal daily activities.
But what drives companies toward this direction, and how can this shift fundamentally change their way of functioning :

1. Speed and Efficiency : GPT-4o offers GPT-4-level intelligence but operates much faster, improving its capabilities across text, voice, and vision.

2. Image Understanding : It has a better understanding of images, allowing users to share pictures for translation, learning about food's history and significance, and getting recommendations.

3. Voice and Video Interaction : Future updates will enable more natural, real-time voice conversations and the ability to converse with ChatGPT via real-time video.

4. Language Support : GPT-4o's language capabilities have been improved in terms of quality and speed, supporting more than 50 languages.

5. Advanced Tools for Free Users : ChatGPT Free users now have access to advanced tools such as data analysis, chart creation, chatting about photos, file uploads for summarizing, writing, or analyzing, and discovering and using GPTs and the GPT Store.

6. Memory Feature : A new feature to build a more helpful experience with memory, enhancing the continuity of conversations.

7. Desktop App : A new ChatGPT desktop app for macOS is being launched for both free and paid users, designed to integrate seamlessly into your workflow.

Personal Use : The Friend and Assistant in Daily Tasks In addition to commercial uses, companies and individuals are beginning to adopt virtual assistants as friends and helpers in daily tasks. This trend enhances the value of AI in our daily lives in multiple ways :

1. Organization and Time Management :
Virtual assistants can help individuals organize their daily schedules, remind them of important appointments, and offer recommendations to improve time management. This boosts personal productivity and helps achieve a better work-life balance.

2. Emotional Support and Conversation :
In a world full of stress and challenges, virtual assistants like ChatGPT can offer emotional support and engaging conversations. Individuals can talk to these tools about various topics, receive advice, or simply enjoy entertainment.

3. Learning and Personal Development :
Virtual assistants can provide educational resources and useful information to enhance skills and knowledge. They can offer lessons on different subjects, assist students with homework, or even recommend books and articles to read.

4. Assistance with Household Tasks :
These tools can offer guidance on how to perform household tasks, such as cooking or simple repairs. They can also provide customized shopping lists based on individual needs and preferences.

Project Caiva :

An Integrated Platform Combining Vision, PDF Search, and Voice and Video Interaction :

In this era of advanced technology, it is essential for companies and individuals to rely on smart solutions that enhance their efficiency and productivity. Project Caiva is one such integrated solution that brings together a wide range of advanced services to meet the diverse needs of users. This project combines features like computer vision, search in uploaded PDFs, and voice and video interaction to offer a comprehensive and seamless experience. But what makes Caiva stand out, and how can these integrated services make a difference in our daily lives and the business world?

1. Computer Vision

One of the leading features of Project Caiva is its ability to process and understand images and videos using advanced computer vision techniques. Caiva can analyze images to recognize objects, texts, and even faces, which opens up many applications such as identifying products in images, analyzing videos to determine activities, or even enhancing security through facial recognition.

2. Search in Uploaded PDFs

Many individuals and companies face significant challenges when searching for specific information within large and complex PDF documents. Project Caiva offers an effective solution to this problem with its advanced search feature in uploaded files. Users can upload PDF files and use Caiva's intelligent search engine to quickly and accurately find the required information, saving time and effort, and improving document management.

3. Voice Interaction

Voice interaction is another feature that distinguishes Caiva, allowing users to interact with the system using voice commands. Whether for making inquiries, executing specific commands, or even controlling smart devices, Caiva makes interacting with technology easier and more seamless. This feature improves the user experience, making it more natural and flexible.

4. Real Time conversation

Caiva supports real-time conversation, enabling users to interact directly with the platform for daily assistance and technical support. This feature offers more effective and personal communication, enhancing the user's interaction with Caiva and improving their overall experience.

Benefits of Project Caiva

1. Enhanced Productivity

By integrating these diverse services into a single platform, Caiva provides a comprehensive solution that increases user productivity. Individuals and companies can complete multiple tasks more efficiently, saving time and effort, and allowing focus on more critical activities.

2. Superior User Experience

The integrated features of Caiva offer a seamless and unified user experience. By providing solutions that rely on the latest technologies in computer vision, voice search, and video interaction, users can benefit from technology in the simplest and most effective ways.

3. Improved Communication and Collaboration

The voice and video interaction features in Caiva enhance communication and collaboration among different teams, whether they are working remotely or on-site. Individuals can share information and interact in real-time, increasing the effectiveness of teamwork.

Chapter 4

System Analysis

Caiva is an advanced desktop application designed to embody the capabilities of a conversational AI virtual assistant. It leverages state-of-the-art AI models and technologies to facilitate interactive communication and perform various tasks efficiently.

Purpose of the chapter :

This chapter aims to provide a detailed exploration of Caiva's functionalities, outlining both its functional and non-functional requirements. It serves as a reference guide for developers, stakeholders, and users, offering insights into the design, implementation, and potential applications of Caiva.

4.1. Functional Requirements :

1. Emotion Recognition :

- **Description :** Caiva should accurately recognize and display emotions such as happiness, sadness, excitement, or confusion based on user speech.
- **Example Application :** When a user asks a question, Caiva's facial expressions change dynamically to reflect the detected emotion. For instance, if a user expresses excitement about a topic, Caiva's avatar smiles and shows animated gestures to convey enthusiasm.
- **Application Idea :** Implement emotion recognition to enhance user engagement in virtual meetings or educational environments, where Caiva's expressions provide non-verbal cues during interactions.

2. Sentiment Analysis :

- **Description :** Utilize fine-tuned sentiment analysis models like Roberta to classify user inputs and generated responses into positive, negative, or neutral sentiments.
- **Example Application :** If a user expresses dissatisfaction with a response, Caiva adjusts its tone or provides a more empathetic follow-up based on the sentiment detected.
- **Application Idea :** Integrate sentiment analysis to enhance customer support interactions, where Caiva can adapt its responses based on the customer's mood and sentiment.

3. Chatbot Functionality :

- **Description :** Enable Caiva to engage in natural language conversations, responding to general inquiries and providing relevant information.
- **Example Application :** Users can ask Caiva about famous character, methodology to learn something new, specific topics like literature, or book-related questions, and Caiva generates informative responses using AI models.
- **Application Idea :** Implement Caiva as a virtual assistant in educational settings, where students can interact with Caiva to ask questions about course material or study tips.

4. Facial Expression Generation :

- **Description :** Use advanced facial animation technology to synchronize Caiva's facial expressions with its detected emotions and responses.
- **Example Application :** Caiva's avatar displays subtle expressions like raised eyebrows or narrowed eyes to convey surprise or skepticism during conversations.
- **Application Idea :** Deploy Caiva in virtual therapy sessions, where its expressive avatar helps convey empathy and understanding based on the patient's emotional state.

5. PDF Querying :

- **Description :** Allow users to upload PDF documents for querying within Caiva, extracting relevant text and providing contextual answers.
- **Example Application :** Caiva can assist researchers by extracting information from academic papers or textbooks, answering specific questions related to the content.
- **Application Idea :** Integrate Caiva into a digital library system, where users can upload documents and interact with Caiva to retrieve information or summaries.

6. Speech to Text:

- **Description :** Enable users to interact with Caiva using voice commands and speech input, converting spoken language into text for processing.
- **Example Application :** Users can dictate questions or commands to Caiva, and the system transcribes the speech accurately to initiate responses.
- **Application Idea :** Implement Caiva in automotive interfaces, allowing drivers to interact hands-free by speaking commands or inquiries while on the road.

7. Multi-device Compatibility:

- Ensure Caiva can seamlessly integrate into various devices and platforms including websites, mobile applications, smartwatches, and car touchscreens.
- Implement responsive design principles to adapt Caiva's interface to different screen sizes and resolutions.

4.2. Non-Functional Requirements :

1. Accuracy and Reliability :

- **Description :** Ensure Caiva consistently provides accurate responses and reliably interprets user interactions.
- **Example Application :** Caiva's AI models undergo continuous training and testing to maintain high accuracy levels in understanding user intents and sentiments.
- **Application Idea :** Deploy Caiva in professional settings such as virtual conferences, where accurate responses and reliability are crucial for effective communication.

2. Responsiveness and Latency :

- **Description :** Optimize Caiva's performance to minimize response times and provide a seamless conversational experience.
- **Example Application :** Caiva responds to user inputs in real-time, minimizing delays between user queries and generated responses.
- **Application Idea :** Integrate Caiva into customer service platforms, where rapid response times contribute to improved customer satisfaction and engagement.

3. Usability :

- **Description :** Design Caiva's interface and interactions to be intuitive and user-friendly for a wide range of audiences.
- **Example Application :** Caiva employs simple conversational prompts and visual cues to guide users through interactions effectively.
- **Application Idea :** Implement Caiva as a personal productivity assistant, helping users manage tasks and schedules through natural language commands and reminders.

4. Security and Privacy :

- **Description :** Implement robust security measures to protect user data and ensure privacy throughout interactions.
- **Example Application :** Caiva encrypts user data and adheres to strict privacy policies, safeguarding sensitive information from unauthorized access.
- **Application Idea :** Deploy Caiva in healthcare environments, where strict security protocols are essential to protect patient data and maintain confidentiality.

5. Interactive Animations :

- **Description :** Develop interactive and engaging animations to enhance Caiva's visual appeal and user experience.
- **Example Application :** Caiva's animations respond dynamically to user inputs, creating a more immersive and personalized interaction.
- **Application Idea :** Integrate Caiva into gaming environments, where animated characters like Caiva enhance storytelling and player engagement through interactive dialogue.

6. Continuous Improvement :

- **Description :** Establish feedback mechanisms and iterative development processes to continuously enhance Caiva's capabilities and performance.
- **Example Application :** User feedback and usage data drive ongoing improvements to Caiva's AI models and user interface.
- **Application Idea :** Deploy Caiva in educational settings, where continuous improvement ensures Caiva remains adaptive to evolving learning needs and preferences.

By incorporating these detailed examples and application ideas, the functional and non-functional requirements for Caiva can be further refined and applied to real-world scenarios, ensuring the development of a versatile and effective conversational AI virtual assistant.

Right now Let's explore some of these practical scenarios for each functional requirement of Caiva :

4.3. Functional Requirements Scenarios :

1. Emotion Recognition :

Scenario : In a virtual classroom setting, a student interacts with Caiva to ask a question about a complex topic. Caiva detects the student's confusion based on the query's complexity and displays a supportive expression, encouraging the student to ask further clarifying questions.

2. Sentiment Analysis :

Scenario : A customer engages with Caiva in an e-commerce chatbot to inquire about product availability. Caiva detects the customer's positive sentiment based on their enthusiastic inquiry and responds with product details and personalized recommendations, enhancing the customer's shopping experience.

3. Chatbot Functionality :

Scenario : A user interacts with Caiva on a website to ask about upcoming events in their area. Caiva utilizes natural language processing to understand the query, retrieves event information from a database, and provides a list of relevant events with dates and locations.

4. Facial Expression Generation :

Scenario : During a virtual therapy session, a patient shares personal challenges with Caiva. As the patient speaks, Caiva's avatar displays empathetic facial expressions, such as a nod of understanding or a comforting smile, to convey support and encouragement.

5. PDF Querying :

Scenario : A researcher uploads a scientific paper to Caiva's system and asks specific questions about experimental methods mentioned in the document. Caiva processes the PDF text, extracts relevant information, and provides concise answers based on the content of the paper.

6. Speech to Text :

Scenario : A driver interacts with Caiva through voice commands in a smart car interface. The driver asks Caiva to find nearby gas stations, and Caiva converts the spoken request into text, retrieves location-based data, and displays a list of nearby gas stations on the car's dashboard display.

4.4. Non-Functional Requirements Scenarios:

1. Accuracy and Reliability :

Scenario : A professional uses Caiva during a virtual conference to present complex concepts. Caiva accurately transcribes and interprets the speaker's technical language, providing accurate real-time translations and summaries to the audience, ensuring reliable communication.

2. Responsiveness and Latency :

Scenario : A user interacts with Caiva in a customer support chat, seeking assistance with a product issue. Caiva responds promptly to the user's inquiries, reducing wait times and maintaining a smooth conversation flow, leading to efficient issue resolution.

3. Usability :

Scenario : An elderly user interacts with Caiva on a tablet to schedule medication reminders. Caiva's interface uses intuitive voice commands and visual prompts, making it easy for the user to set up and manage daily reminders without confusion.

4. Security and Privacy :

Scenario : A patient uses Caiva in a healthcare setting to inquire about appointment availability. Caiva ensures strict adherence to privacy regulations, encrypting all sensitive patient data and securely transmitting appointment details to healthcare providers.

5. Interactive Animations :

Scenario : A child interacts with Caiva in an educational app, learning about dinosaurs. Caiva's animated character engages the child with playful gestures and educational content, fostering a fun and interactive learning experience through engaging animations.

6. Continuous Improvement :

Scenario : A language learner interacts with Caiva in a language tutoring app. Caiva collects feedback on user interactions and usage patterns, continuously updating its language models and content to provide personalized and effective language learning support over time.

These scenarios illustrate how Caiva's functional and non-functional capabilities can be applied across various use cases, from education and healthcare to customer service and personal productivity.

By envisioning these practical applications, developers can design and optimize Caiva to deliver impactful experiences that meet diverse user needs and expectations.

Chapter 5

Tools and Technologies

5.1. Tools and Technologies for Unreal

5.1.1. Unreal Engine (Ref 13.1)

1. What is Unreal Engine ?

Unreal Engine (UE) is a powerful and widely-used game engine developed by Epic Games. It provides developers with a complete suite of tools for creating high-quality games and real-time 3D applications.

2. Why Unreal Engine ?

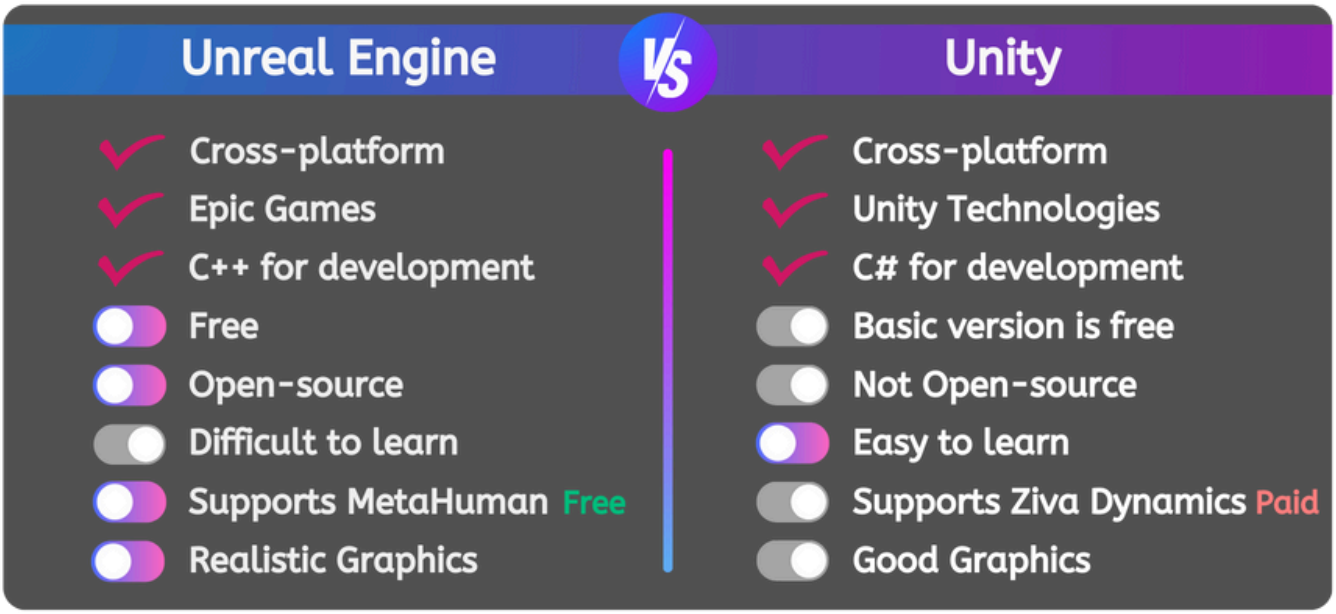


Table 1. comparing between Unreal Engine & Unity

3. Key Features of Unreal Engine.

- **High-Quality Graphics :** Advanced rendering capabilities and visual effects.
- **Blueprints :** Visual scripting system for rapid development.
- **C++ Integration :** High-performance, low-level programming capabilities.
- **Cross-Platform :** Develop for PC, consoles, mobile, VR, and AR.
- **Asset Marketplace :** Access to a vast library of ready-made assets.
- **Real-Time Ray Tracing :** Support for realistic lighting and reflections.

4. System Requirements

Recommended :

- **OS :** Windows 10 64-bit or macOS 10.14.6 or later.
- **Processor :** Hexa-core Intel or AMD processor, 3.0 GHz or faster.
- **Memory :** 32 GB RAM.
- **Graphics :** NVIDIA GeForce GTX 1080 or AMD Radeon RX Vega 64.

5. Basic Concepts

- **Project:** A container for all your game assets and code.
- **Level:** A map or a game environment.
- **Actors:** Objects that can be placed in a level (e.g., characters, lights, cameras).
- **Components:** Sub-objects that define the Actor's behavior and properties.

6. Unreal Engine Editor

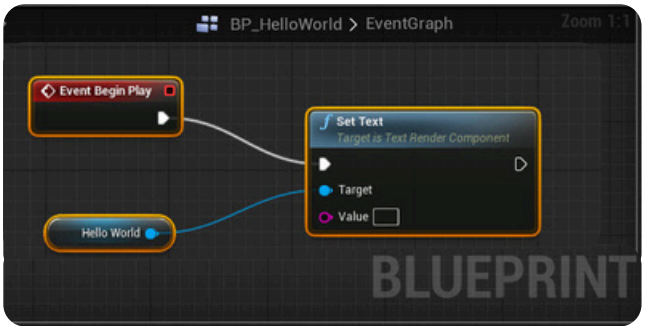
- **Viewport:** Main area for interacting with your levels.
- **Content Browser:** Manage your assets like textures, models, and sounds.
- **Details Panel:** View and edit properties of selected objects.
- **Blueprint Editor:** Create and edit visual scripts

7. Scripting and Programming

- **Blueprints:** Drag-and-drop interface for scripting gameplay without code.
- **C++ :** Comprehensive API for advanced programming.

Example Blueprint:
Creating a simple "Hello World" blueprint :

Create a new Actor blueprint.
Add a Text Render component.
In the Event Graph, add a Begin Play event.
Connect the event to a Set Text node.
Set the text to "Hello World".



8. Realistic Human Characters

- **MetaHuman Creator** enables developers to create highly detailed and realistic human characters more efficiently. This tool leverages cloud computing to handle complex tasks, allowing developers to customize characters' appearances, movements, and facial expressions with ease.

9. Asset Management

- **Importing Assets:** Import models, textures, audio, and animations.
- **Asset Organization:** Use folders and naming conventions for better management.
- **Material Editor:** Create and edit materials for your 3D models.

10. Rendering and Graphics

- **Lighting:** Static, dynamic, and real-time ray tracing options.
- **Post-Processing:** Effects like bloom, depth of field, and color grading.
- **Shaders:** Customizable through the Material Editor.
- **Dynamic Lighting:** Lumen provides dynamic global illumination, allowing for real-time lighting changes that reflect accurately across scenes. This means environments react to light sources in a realistic manner, enhancing immersion and visual fidelity, particularly in dynamic scenes where lighting conditions change frequently.

11. Nanite Virtualized Geometry

- **Detail and Performance:** Nanite allows for unprecedented levels of detail by enabling artists to import film-quality source art comprised of millions of polygons directly into the engine. This eliminates the need for manual optimization and LODs (Levels of Detail), making it possible to create highly detailed environments without sacrificing performance.

12. Physics and Collision

- UE5 introduces advanced animation tools that integrate physics-based animations, enabling more realistic and fluid character movements. This includes Control Rig, Full-Body IK solver, and Motion Warping, which allow for more dynamic and responsive character interactions.

13. Animation

- **Skeletal Meshes:** 3D models with bones for animation.
- **Animation Blueprints:** Control animations through Blueprints.
- **Sequencer:** Create cinematic sequences and cutscenes.

14. Packaging and Deployment

- **Build Configurations:** Choose between Development, Debug, and Shipping builds.
- **Platform Targets:** Package your game for different platforms (PC, mobile, consoles).
- **Distribution:** Use platforms like Steam, Epic Games Store, or direct distribution.

5.1.2. Metahuman Creator (Ref 16.1)

1. What is Metahuman Creator ?

MetaHuman Creator is a cloud-based application developed by Epic Games that allows users to create highly realistic digital humans for use in Unreal Engine 5. It provides a streamlined, easy-to-use interface for generating lifelike characters with customizable features, including skin, hair, clothing, and facial expressions.

2. Key Features

High-Quality Models : Create photorealistic human characters.

Customization : Users can customize various aspects of their MetaHumans, including facial features, hairstyles, body proportions, and clothing. The tool provides a range of preset characters to start from or allows users to create characters from scratch.

Animation Ready : Fully rigged models compatible with Unreal Engine's animation tools.
Hair and Clothing : Realistic hair and clothing options with physical properties.
Blendshapes : Detailed control over facial expressions and emotions.

3. System Requirements

MetaHuman Creator is a cloud-based application, so it primarily relies on internet access and the specifications of the Unreal Engine 5 installation on your local machine. and for optimal performance, higher-end specifications are recommended (Unreal Engine).

Recommended :

- **OS :** Windows 10 64-bit
- **Processor :** Octa-core Intel or AMD, 3.0 GHz or faster
- **Memory :** 32 GB RAM
- **Graphics :** NVIDIA RTX 2080 / AMD RX 5700XT or better

4. Integration with Unreal Engine

MetaHumans are designed to be used within Unreal Engine, ensuring seamless integration. Characters created with MetaHuman Creator can be downloaded using Quixel Bridge and imported directly into Unreal Engine for further use and animation.

5. Creating a MetaHuman

Start a New Project : Open MetaHuman Creator and choose to start a new project.
Base Model : Select a base model to start with.
Customizing : Use the intuitive interface to adjust features such as:

- **Face :** Modify facial features like eyes, nose, mouth, and ears.
- **Body :** Adjust height, build, and proportions.
- **Skin :** Choose skin tone and apply realistic textures.
- **Hair :** Select and customize hairstyles.
- **Clothing :** Pick from a variety of outfits and accessories.

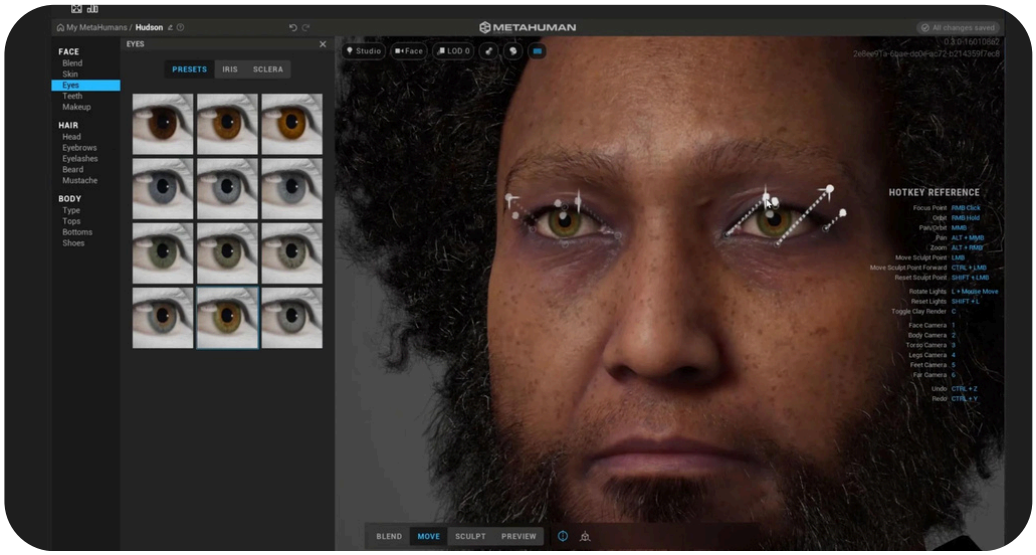


Figure 2. MetaHuman Interface

6. Customizing Your MetaHuman

Detail Adjustments : Fine-tune details with sliders and control points.
Blendshapes : Use blendshapes to create custom facial expressions and poses.
Material Adjustments : Adjust materials for skin, hair, and clothing for the desired visual effect.
Facial Animation : Test and refine facial animations directly in the MetaHuman Creator.

7. Performance Capture

MetaHuman Animator, an extension of the MetaHuman toolset, allows for high-fidelity facial animation using performance capture data. This can be done using an iPhone (models 12 and newer) or a head-mounted camera (HMC) system.

8. Plugin for Custom Meshes

The MetaHuman Plugin for Unreal Engine includes features like Mesh to MetaHuman, which converts custom facial meshes into fully rigged MetaHumans, and MetaHuman Animator for detailed facial performance capture and animation.

5.1.3. Geppetto plugin (Ref 14.2)

1. Runtime Phonemes generation and animation (Blueprint)

If the audio file is generated at runtime, i.e: animating a character using the player microphone or using our Ariel Text-to-Speech plugin from a sentence that the player enters with his keyboard, then it is required to generate and animate the phonemes at runtime.

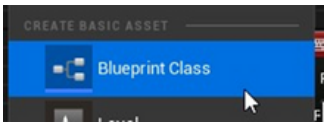
➜ Currently, it is not possible to use SoundWave files or audio files at runtime, because we are struggling to get the audio bytes data from the game at runtime in order to send it to our API server. Instead, the audio file needs to be hosted on a publicly accessible URL

Create a new Blueprint class and add the following component. If you use Metahuman, you can use the generated Blueprint class instead (i.e: BP_Ada for the Metahuman named Ada) :

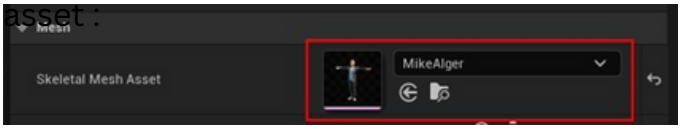
- A Skeletal MeshComponent with your Skeletal Mesh Asset assigned
- A MediaSoundComponentwithavalidMedia Player attached
- A GeppettoURLPlayer
- A Geppetto Player Component or any inherited component if you have a custom function to change the Morph Target values, like Metahuman

If your Skeleton doesn't use the traditional node "Set Morph Target" to update Morph Target values, like Metahuman characters, you need to override functions from the Geppetto Player Component otherwise animation will not work. Please read section 4.4.1-Component Inheritance for more information.

On the Content Drawer, right-click at the location wanted to create the Blueprint class :

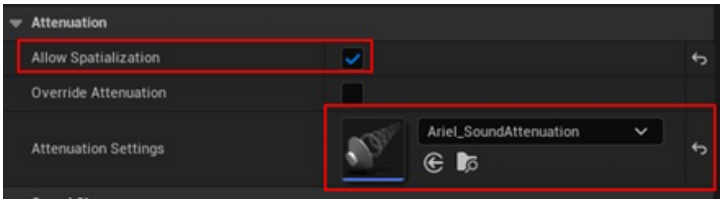


After adding the "Skeletal Mesh" component, navigate to its properties and set your asset :

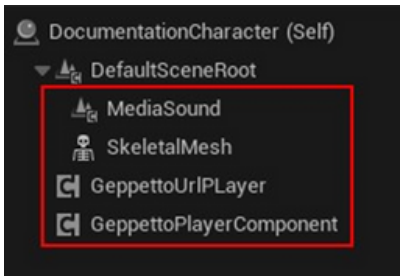


After adding the "Media Sound" component, navigate to its properties and attach a media player to it (you can create a new one if needed)

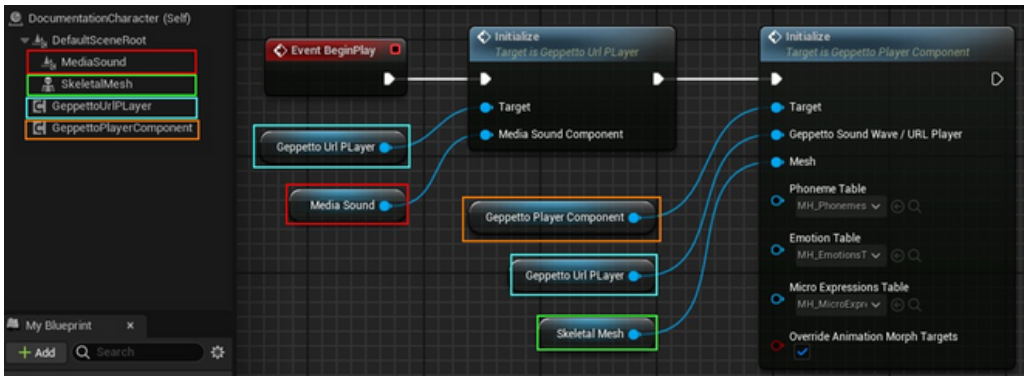
If you want to have audio spatialization,check the "Allow Spatialization" box and set a sound attenuation (you can create a new one if needed):



Add the Geppetto URL Player and the Geppetto Player Component. you should now have four components added to you Blueprint class :

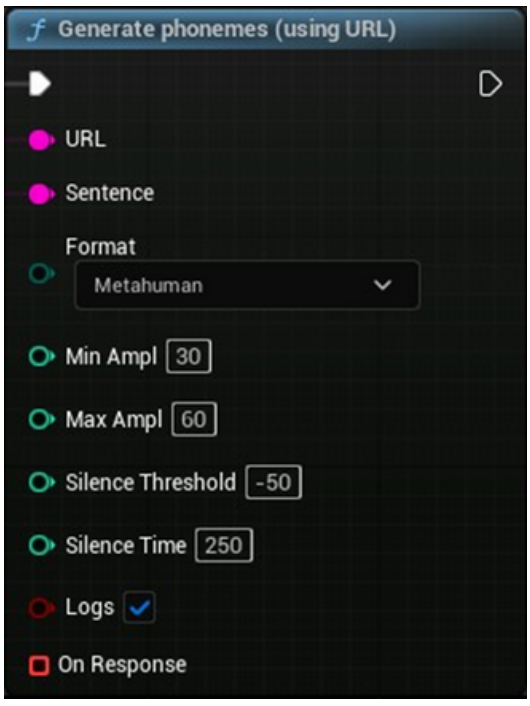


Navigate to the Event Graph and add the following nodes to the BeginPlay event :



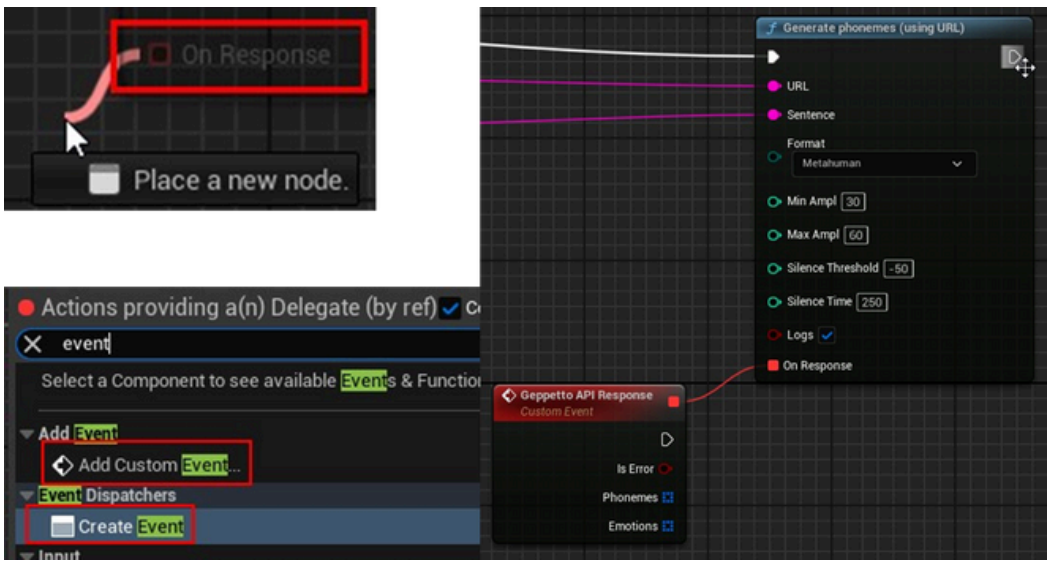
The Geppetto URL Player Initialize node takes a Media Sound Component as input parameter. The Media player attached will be used to download and play the audio from the URL. Please read section 4.3.1 - Geppetto URL Player initialization for more details about those parameters.

Whenever you want in your blueprint, i.e: right a er the audio file has been generated, call the node Generate Phonemes (Using URL). Please note that it could take up to 15 seconds to generate the phonemes, depending on your audio size and length ! For a small file and short sentence (10s or less, <100Kb), the generation should take less than a second :

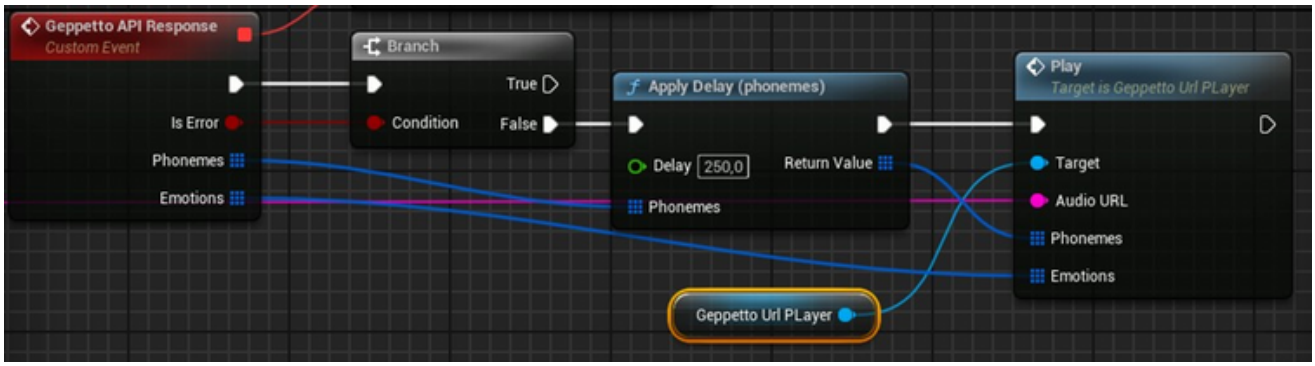


URL	The audio URL of the speech, in wav format. If you use our Ariel plugin to generate the audio files at runtime, please use the generated URL from Ariel API response directly.
Sentence	The sentence(s) spoken in the audio file, including the emotion tags. Please refer to section 3.2 - Emotion Tag System for more information about how emotion tags work.
Format	Select the Phoneme format returned by the API. Unless you create a custom Data Table that handles those phoneme formats, you should always use "Metahuman" even if you don't use Metahuman characters. Please read section 4.5 - Geppetto Phoneme Data Table for more details.
Amplitude	Choose the range of amplitude you want your animation to have. Higher values will produce a greater articulation of the mouth, while lover values will make the character whisper.
Silences detection	These parameters will change with different audio recording setup. You can open your audio in audio software like Audacity and check: - for "Threshold", the highest sound intensity (in dB) that your mic is recording when you are not talking. - for "Time", the lowest silence time in your audio. Be careful as you need to have your "Time" parameters in ms longer than the silence between two phonemes in your audio.
Logs	Indicate if Geppetto Logs will be printed to the console and/or screen (Debug only).

Dragthemousefromthe"OnResponse"pinanddropitonyourEventGraph.Youcanthen selectAddCustomEventorCreateEventactions,andnametheeventasyouwant:



From the created Event, take the generated Phonemes and Emotions arrays and use them with the Geppetto URL Player Play node. We recommend you to check if there was an error first. You can use the Apply delay node if needed. Please use the same audio URL used for the phonemes generation :

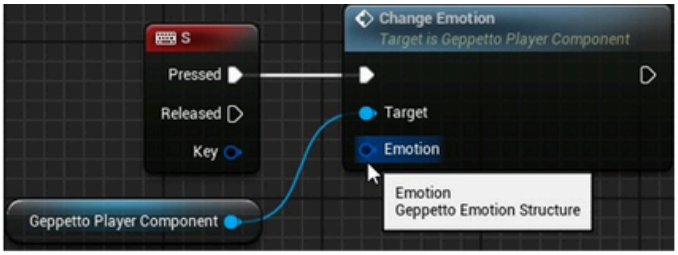


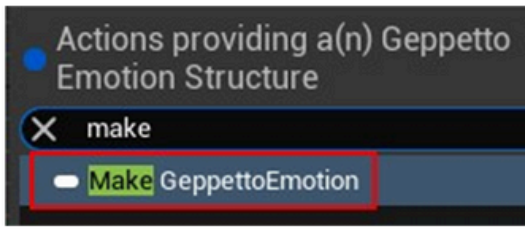
2. Change Emotions

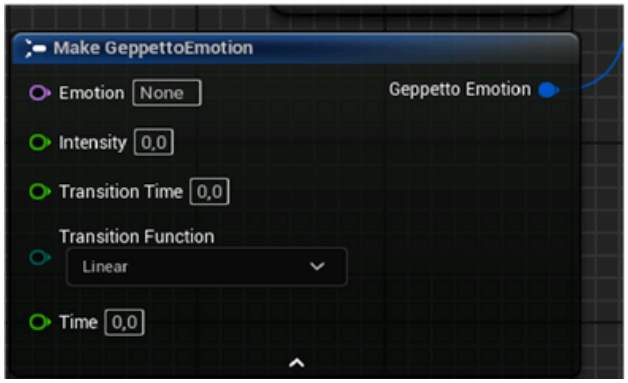
In addition to lip sync animation, the Geppetto plugin allows you to animate emotion to your character. All emotions must have been defined in the Emotion Data Table used by the Geppetto Player. Please read section 4.6 - Geppetto Emotion Data Table for more details about how to create emotion Data Tables.

If you use Metahuman characters, you can use the existing Data Table named “MH_Emotions”. See the Geppetto Player Component Initialize node for more details on how to use Metahuman Data Tables.

Emotions can be changed when a sentence is pronounced by using tags. Please read section 3.5 - Emotion Tag System for more information on how to use tags. You can also use the Geppetto Player node Change Emotion to change your character emotion at any time you want! Simply drag the mouse from the Emotion pin and choose Make Geppetto Emotion :







Emotion	The emotion name. The emotion must be defined in the Emotion Data Table.
Intensity	The emotion intensity percentage, range 0 - 100.
Transition Time	The switching emotion transition time, in seconds. For example, if the current character emotion is 'Neutral', the new emotion passed is 'Happy' and the transition time is 0.2s, then the character face will bend from neutral pose to happy pose in 0.2 seconds (200ms).
Transition Function	The transition interpolation function used to bend between emotions.
Time	This parameter is used for internal purposes and should not be edited. This parameter will be hidden in the future version of the plugin.

3. Emotion Tag System

The emotion tag system allows you to change the character emotion at a specific point of the sentence. Its base syntax is the following :

<emotion ***emotion_name*** intensity ***int_value*** transition ***transition_value*** function_type ***func_name***>

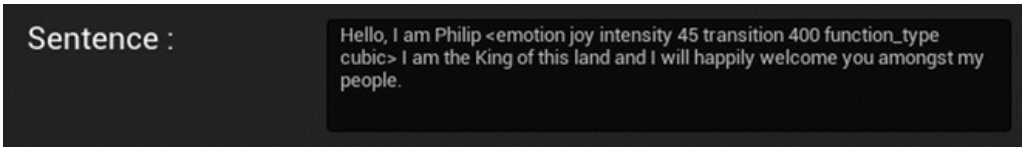
The emotion tag contains the following parameters :
emotion (emotion_name) (string) : the name of the emotion. This emotion must be part of the existing emotion list. This is the only mandatory parameter for the emotion tag to work.

intensity (int_value) (int) : value ranging from 0 to 100 that defines the intensity of the emotion. This parameter is optional, if not specified, this value defaults to 50. transition

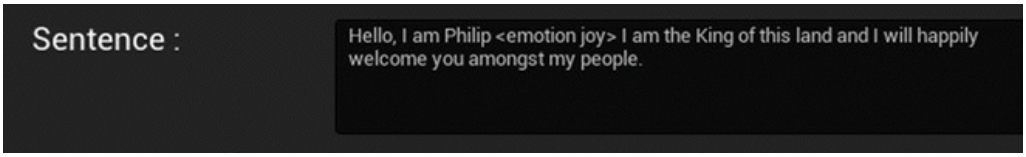
(transition_value) (int) : value that defines the transition time (in milliseconds) to the emotion. This parameter is optional, if not specified, this value defaults to 200 milliseconds.

function_type (func_name) (string) : the type of function used for the interpolation. This parameter is optional, if not specified, this value defaults to "cubic".

Example of an emotion tag with all optional parameters :



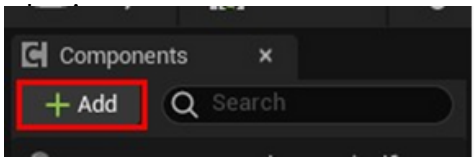
Example of an emotion tag without optional parameters :

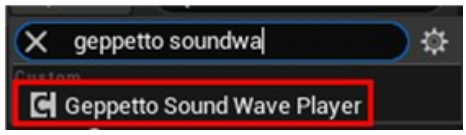


Emotions can also be played directly in blueprints, whenever the game needs to change the character emotion. Please read section 3.3 - Change Emotions of the documentation for more details on how to change an emotion at Runtime using Blueprints.

4. Geppetto SoundWave Player

The Geppetto SoundWave Player is a child of the Geppetto Base Component class. This component is used to synchronize phonemes/emotions animation from a Geppetto Data Asset with a SoundWave audio. If you need to use the Geppetto plugin combined with SoundWave audio, just add the component to the Blueprint(s) that uses the Geppetto

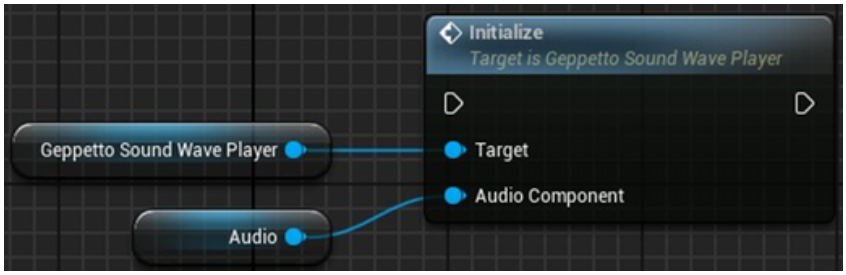




Please note that the Geppetto SoundWave Player does not animate the phonemes and emotions itself, it only does the synchronization between the phonemes and the audio.
In order to completely animate phonemes and emotion, please use a Geppetto Player Component in addition.

4.1. Initialize

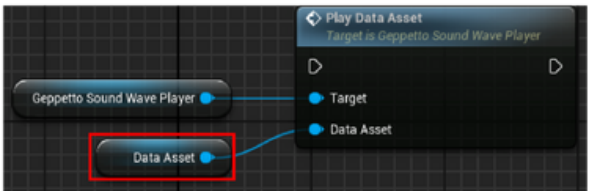
This node is used to initialize the component and should be called in your Blueprint BeginPlay event. The Audio Component passed as parameter will be used by the component to play the SoundWave audio :



Audio Component	The audio component that will be used by the SoundWave player to play the audio file.

4.2. Play Data Asset

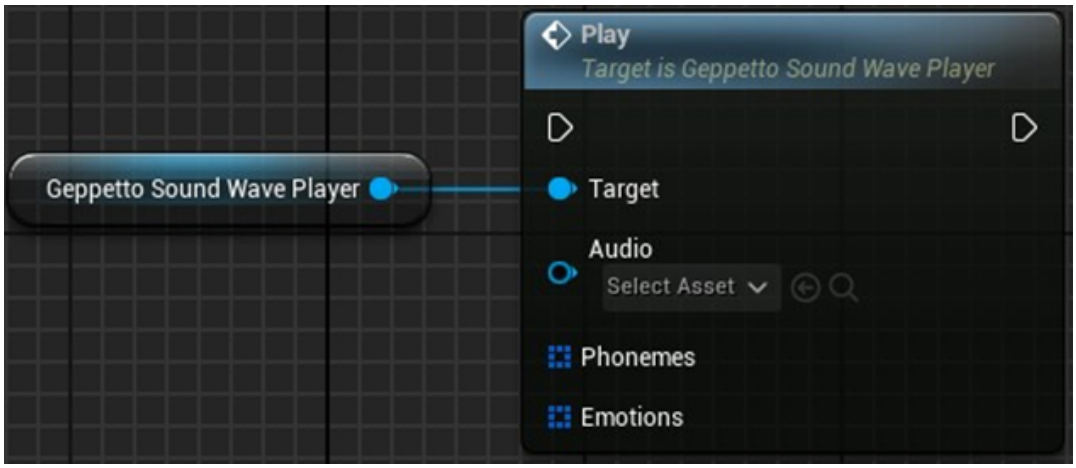
When the lip sync needs to be performed, the node PlayDataAsset can be called to play the speech coming from a Geppetto Data Asset. The Data Asset can be be selected directly from within the node or through a variable



Data Asset	The Geppetto Data Asset that will be played by the SoundWave Player.

4.3. Play

Sometimes, you don't want to directly use the Phonemes/Emotion/SoundWave stored in the Data Asset. In this case, you can use the Play node to pass all parameters one by one. For example, this node can be useful if you want to edit the phonemes/emotions list (i.e.: Apply Delay) or if you want to use a different SoundWave than the one used (i.e.: add some audio effects) above:



Audio	The SoundWave file that will be played by the component.
Phonemes	The phonemes list that will be processed by the component. Can be empty.
Emotions	The emotions list that will be processed by the component. Can be empty.

4.4. Variables

Audio Component	The audio component reference passed to the SoundWave Player when initializing. Internal use only.
Current Data	The DataAsset reference passed to the SoundWave Player when playing lip sync. Internal use only.
Previous Playback Percent	The previous audio component playback percent value. Internal use only.
Update duration with Tick	If the audio playback is at 100% but there are remaining phonemes or emotions, the component will use the tick delta seconds to play the remaining ones.

There is no additional Delegate/Events added to the Geppetto SoundWave Player yet, but you can use the Events inherited from the Geppetto Base Component if needed.

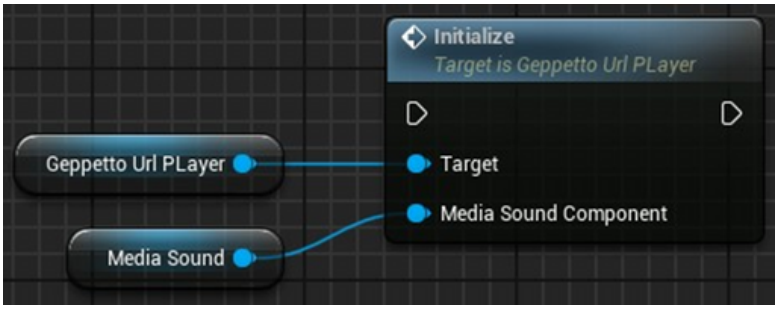
5. Geppetto URL Player

The Geppetto URL Player is a child of the Geppetto Base Component class. This component is used to synchronize phonemes/emotions animation from the node Generate Phoneme (using URL) with an audio URL played within the engine using a Media Sound Component. If you need to use the Geppetto plugin combined with URL audio, simply add the component to the Blueprint(s) that uses the Geppetto plugin:

Please note that the Geppetto URL Player does not animate the phonemes and emotions itself, it only does the synchronization between the phonemes and the audio. In order to completely animate phonemes and emotion, please use a Geppetto Player Component in addition.

5.1. Initialize

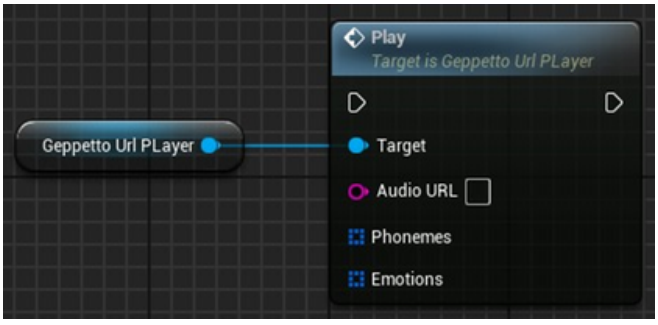
This node is used to initialize the component and should be called in your Blueprint BeginPlay event. The media sound component passed as parameter will be used by the component to play the URL audio :



Media Sound Component	The media sound component that will be used by the URL player to play the audio url.

5.2. Play

When the lip sync needs to be performed, the node Play can be called to play the speech coming from Runtime generated lip sync. The URL must be the same as the one used with the node Generate Phoneme (using URL):



Audio URL	The audio URL that will be played by the component. The player must have internet access to be able to play the audio from the URL.
Phonemes	The phonemes list that will be processed by the component. Can be empty.
Emotions	The emotions list that will be processed by the component. Can be empty.

5.3. Variables

Media Player	The media player component reference passed to the URL Player when initializing. Internal use only.
Current Audio URL	The audio url passed to the URL Player when playing lip sync. Internal use only.
Is waiting processing	A variable used to indicate that the player has already called the "Play" node but the media player is not playing the sound yet (i.e: the sound is being cached). Internal use only.

There is no additional Delegate/Events added to the Geppetto URL Player yet, but you can use the Events inherited from the Geppetto Base Component if needed.

6. Geppetto Player Component

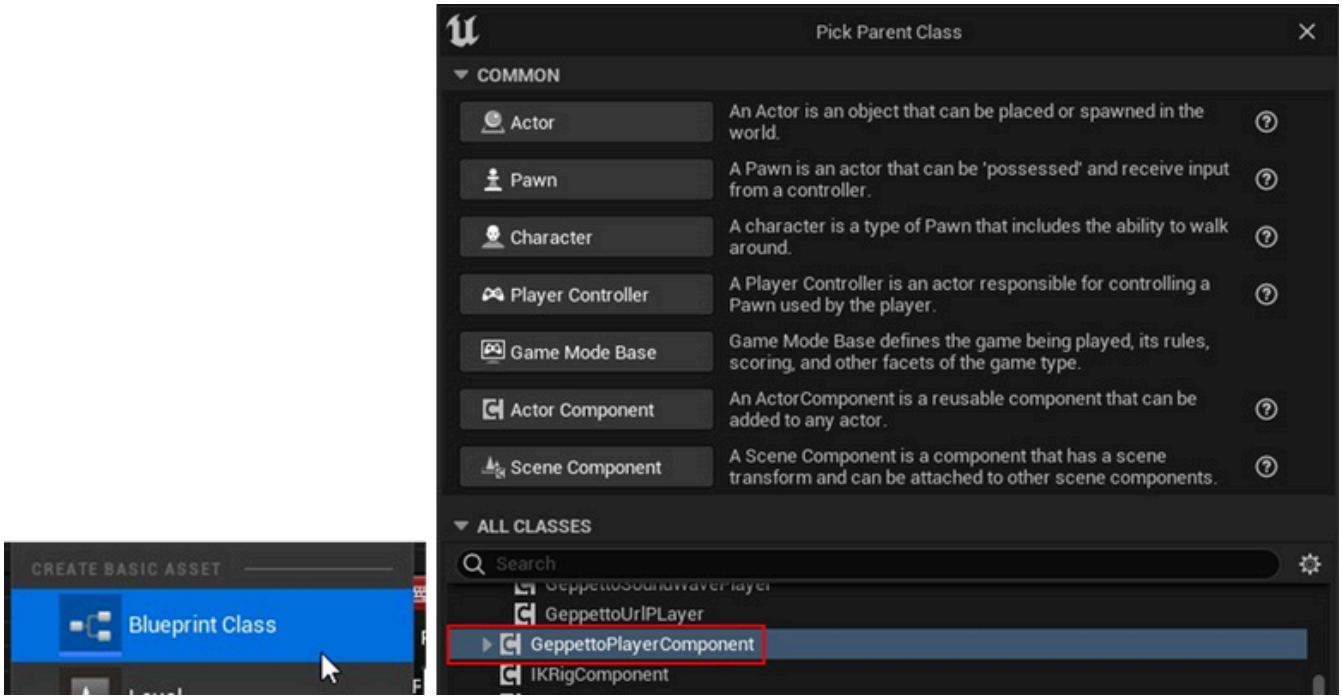
Unlike the other Geppetto Components, the Geppetto Player Component is not used to synchronize the lip sync animation with the audio, but is used to animate the Geppetto Phoneme, Emotions and Micro-Expressions used with your character(s). The Geppetto Player component uses Geppetto Data Tables provided in the component initialization to perform animations.
If the Skeletal Mesh or the Skeletal Anim Instance does not use the usual node Set Morph Target to update the Morph Target values, like Metahuman, it is required to create a new component that inherit the Geppetto Player Component and override the nodes to get and set a Morph Target. Please read the section 4.4.1- Component Inheritance below for more details.

In order to use the Geppetto Player Component, just add the component (or any inherited instance) to the Blueprint(s) that uses the Geppetto plugin

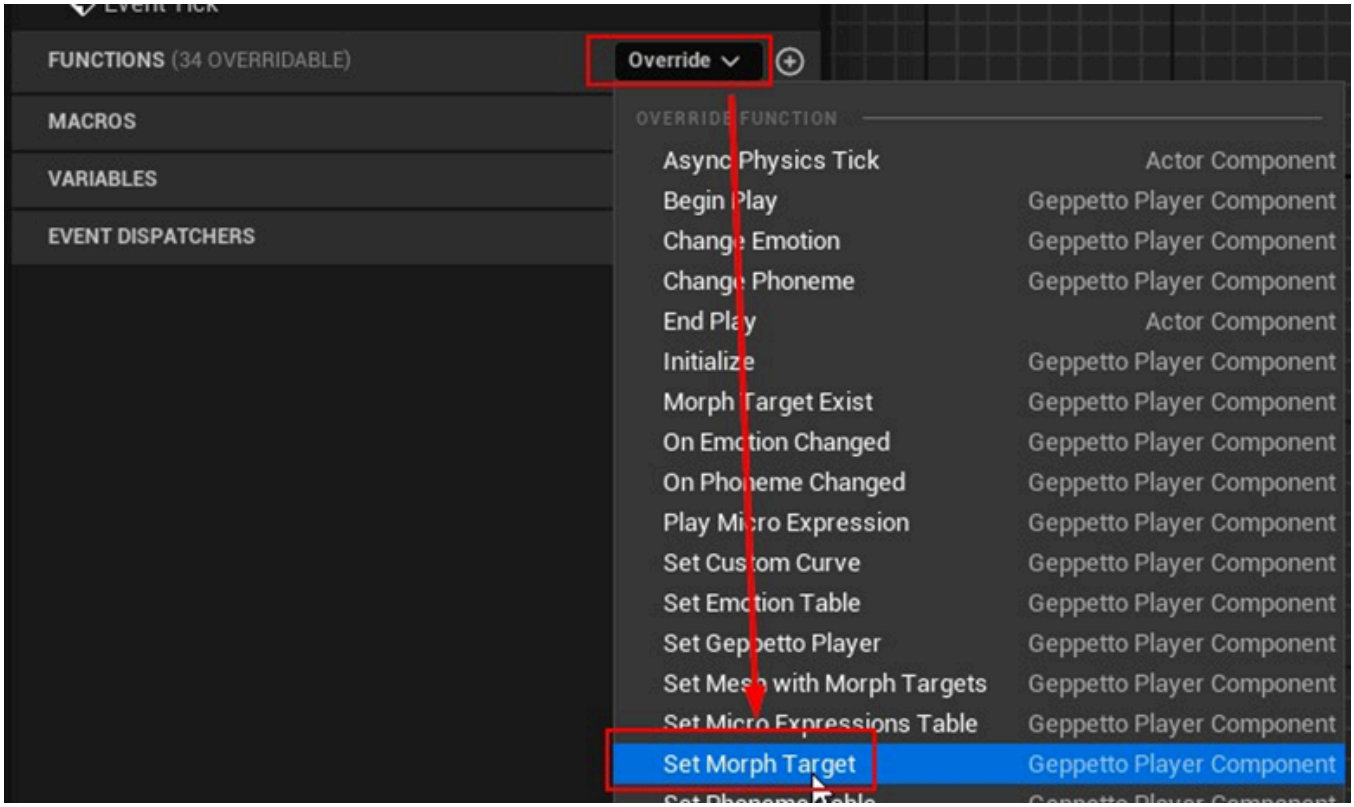
6.1. Component Inheritance

When using Metahumans characters or a custom node to get and/or set the Morph Targets values to the Skeletal Mesh Asset used, it is required to create a new component class that will inherit from the Geppetto Player Component class and override the nodes Set Morph Target and Morph Target Exist. Follow these steps to create a new component:

Create a new Blueprint Class that have “GeppettoPlayerComponent” as parent class :

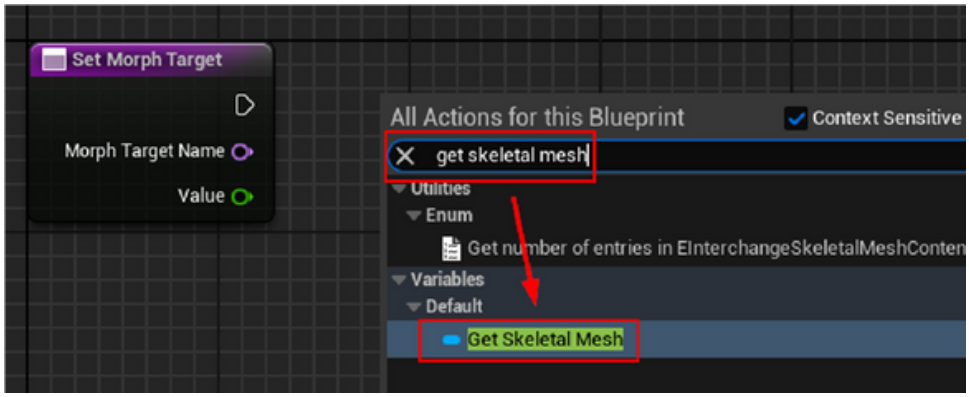


Open the created Blueprint class and override the function named “Set Morph Target” :

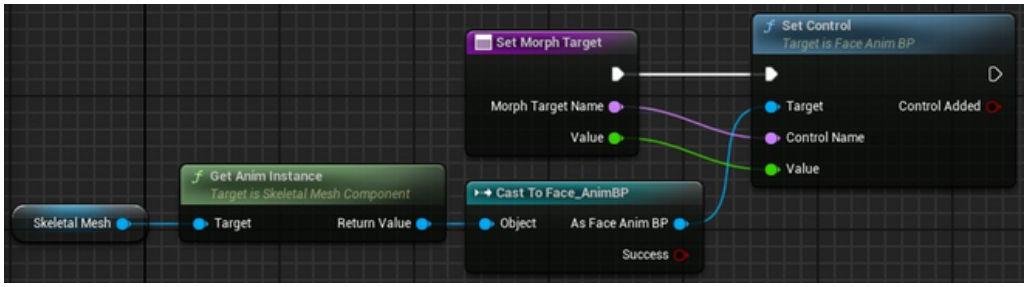


Put the nodes used to change the morph target values inside the function. You can delete the call to the parent node (Parent: Set Morph Target).

You can type “Get Skeletal Mesh” in the action selection to access the Skeletal Mesh Asset :



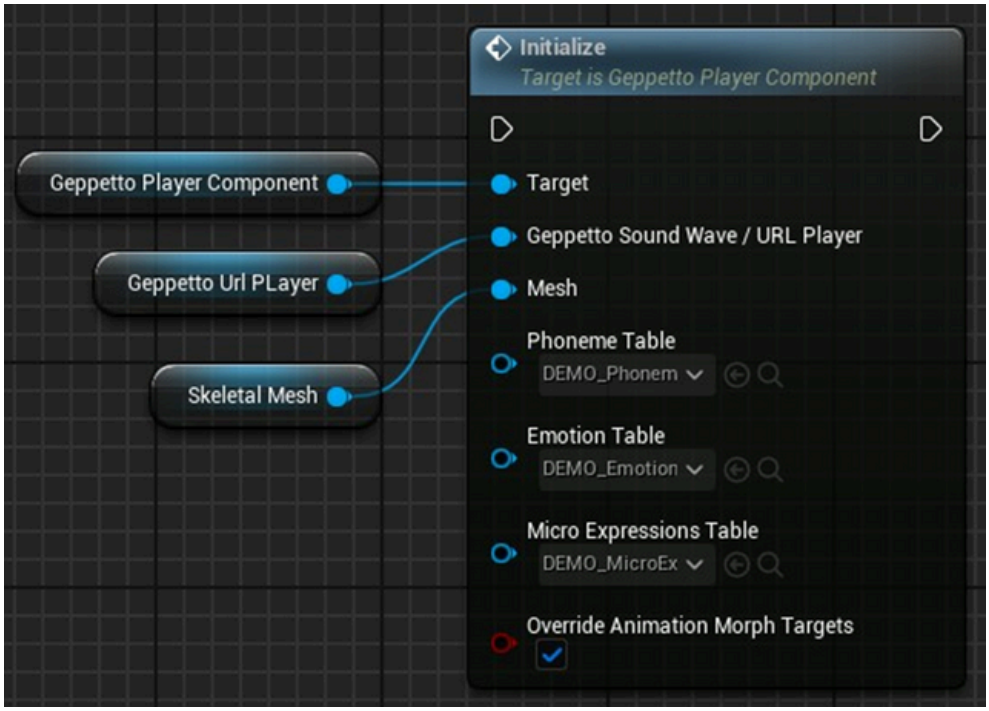
Here is an example for the Metahuman implementation:



If required, you can also override the function “Morph Target Exist”. This function must return true when the input morph target name does exist, false otherwise. For Metahuman, it is not required to override it.

6.2. Initialize

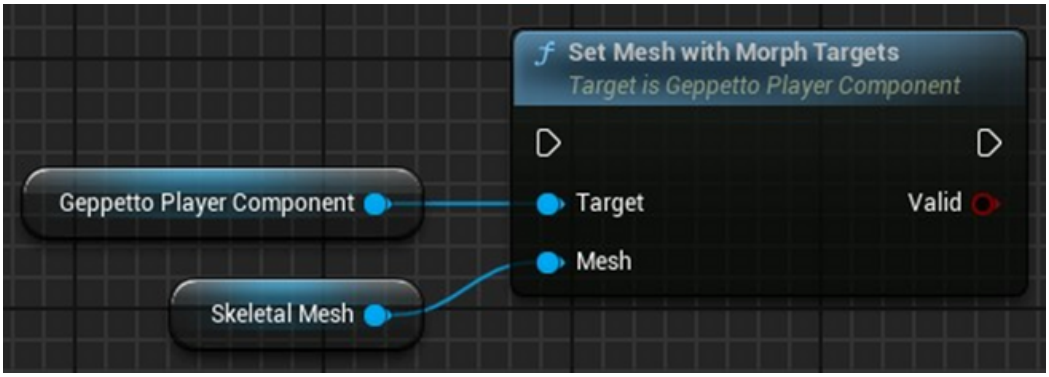
This node is used to initialize the component and should be called in your Blueprint BeginPlay event. The Geppetto SoundWave Player or the Geppetto URL Player, as well as the Skeletal mesh and the Geppetto Data Tables, can be specified in the node parameters:



Geppetto SoundWave/URL Player	The Geppetto Player used to synchronize animations with the audio. This can either be a Geppetto SoundWave Player, a Geppetto URL Player or any Geppetto Base Component subclass.
Mesh	The Skeletal Mesh Asset that the component will use to change morph target values and perform the phonemes, emotions and micro expressions animations.
Phoneme Table	The Geppetto Phoneme Table used by the component. For Metahuman, please use the included Data Table named "*MH_PhonemesTable*" *.
Emotion Table	The Geppetto Emotion Table used by the component. For Metahuman, please use the included Data Table named "*MH_EmotionsTable*" *.
Micro Expressions Table	The Geppetto Micro Expressions Table used by the component. For Metahuman, please use the included Data Table named "*MH_MicroExpressionsTable*" *.
Override Animation	If unchecked, the Morph Target values used by the Geppetto plugin will be blended with the Morph Target values coming from animations. This option can be useful if you want to have both animations and the Geppetto plugin working with the same Morph Targets. Please note that depending on how the Geppetto Player overrides the function "Set Morph Target", unwanted behavior may appear. For Metahuman, this option **must** be checked.

6.3. Set Mesh with MorphTargets

Use this node to change the Skeletal Mesh used by the Geppetto Player Component a er being initialized :



Mesh	The Skeletal Mesh Asset that the component will use to change morph targets values and perform lip sync, emotions and micro expressions animations.
Return value	Return *true* if the mesh have been updated, *false* otherwise

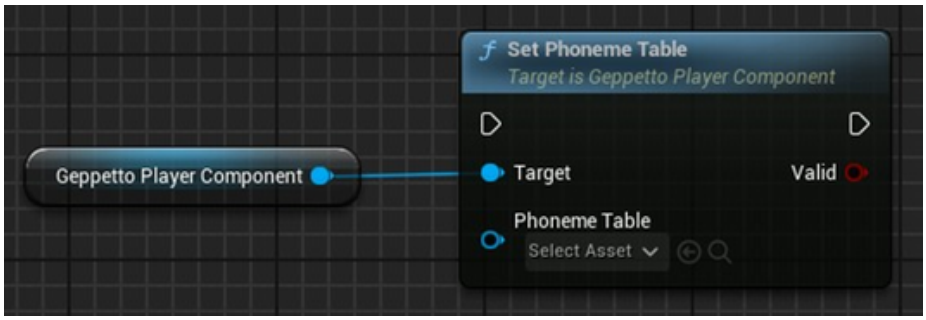
6.4. Set Geppetto Player

Use this node to change the Geppetto Player used by the component a er being initialized. It can either be a Geppetto SoundWave Player, a Geppetto URL Player or any Geppetto Base Component subclass :

Geppetto SoundWave/URL Player	The Geppetto Player attached to the component.

6.5. Set Phoneme Table

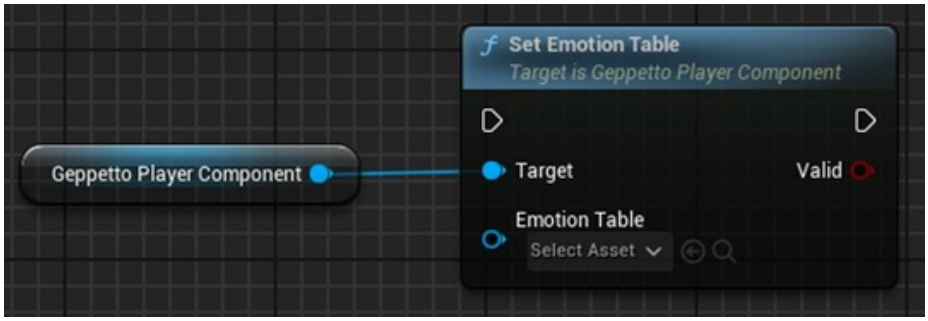
Use this node to change the Geppetto Phoneme Data Table used by the Geppetto Player Component a erbeing initialized:



Phoneme Table	The Geppetto Phoneme Table used by the component.
Return value	Return *true* if the table has been updated, *false* otherwise.

6.6. Set Emotion Table

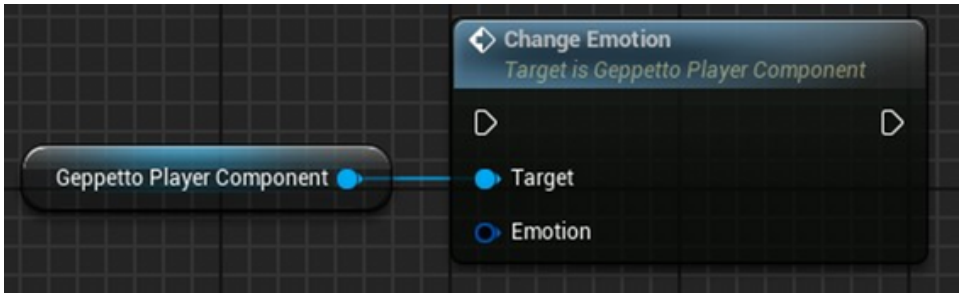
Use this node to change the Geppetto Emotion Data Table used by the Geppetto Player Component a erbeing initialized :



Emotion Table	The Geppetto Emotion Table used by the component.
Return value	Return *true* if the table has been updated, *false* otherwise.

6.7. Change Emotion

Use this node to change the current emotion pose. Please read section 3.3- Change Emotions for more details on how to change emotions :



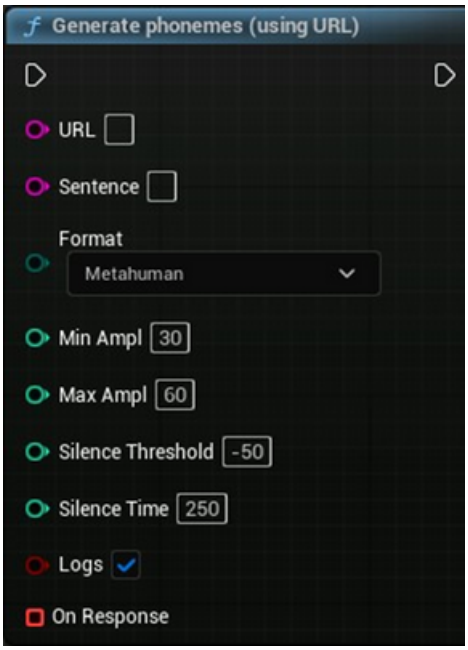
Emotion	See Geppetto Emotion structure.

7. Geppetto Blueprint Library

The following functions can be used in both runtime and editor assets , such as Blueprint classes. Please note that the following functions are declared in C++

7.1. Generate phonemes (using URL)

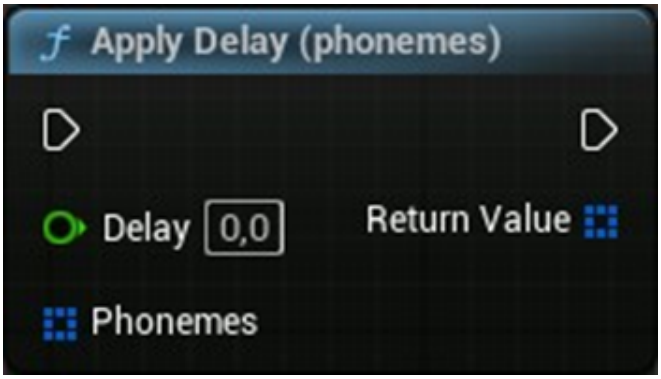
Generate the phonemes for the given audio and sentence. The audio file is available through a publicly accessible URL. The audio must be in wav format (PCM16 recommended) :



URL	The audio URL. See section 3.2 for more details.
Sentence	The sentence spoken in the audio. Can include emotions tags. Please see section 3.3 for more details.
Format	The phoneme format returned by the API. Please read section 4.11.2 for more details. *Default is "Metahuman".*
Min Ampl	The minimum amplitude value (range 0 - 100). *Default is 30.*
Max Ampl	The maximum amplitude value (range 0 - 100). Must be higher or equal to MinAmpl. *Default is 60.*
Silence Threshold	The threshold used to identify audio silences. *Default is -50dB.*
Silence Time	The minimum silence time duration to be considered as a pause, in milliseconds. *Default is 250ms.*
Logs	Indicate if logs should be printed to the console. *Default is true.*
On Response	The delegate event called when a response was received from the API.

7.2. Apply Delay (Phonemes)

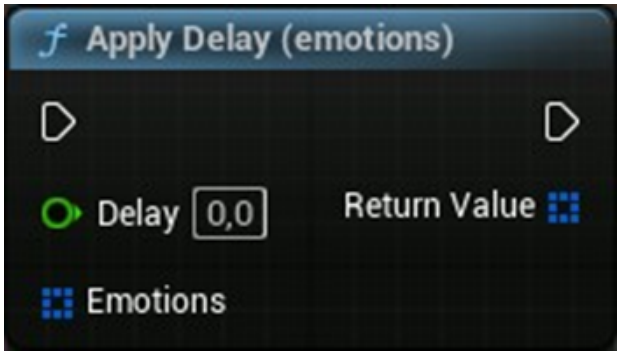
Apply the Delay (in milliseconds) to all Phoneme times. The delay can be positive or negative :



Delay	The delay that will be applied, in milliseconds. Can be positive or negative.
Phonemes	The phonemes list to apply the delay to.
Return Value	The phonemes list with the delay applied.

7.3. Apply Delay (Emotions)

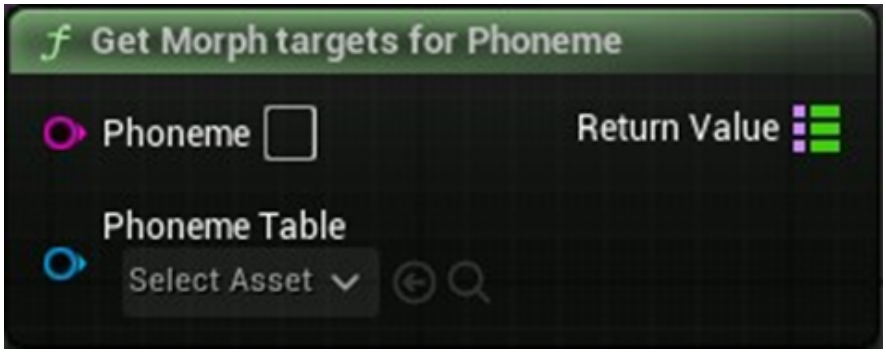
Apply the Delay (in milliseconds) to all Emotions times. The delay can be positive or negative:



Delay	The delay that will be applied, in milliseconds. Can be positive or negative.
Emotions	The emotions list to apply the delay to.
Return Value	The emotions list with the delay applied.

7.4. Get Morph Targets for Phoneme

Get all Morph targets values for the given Phoneme (with maximum amplitude). The function uses the PhonemeTable to identify the Morph targets and their values. This function should disappear in the upcoming plugin versions



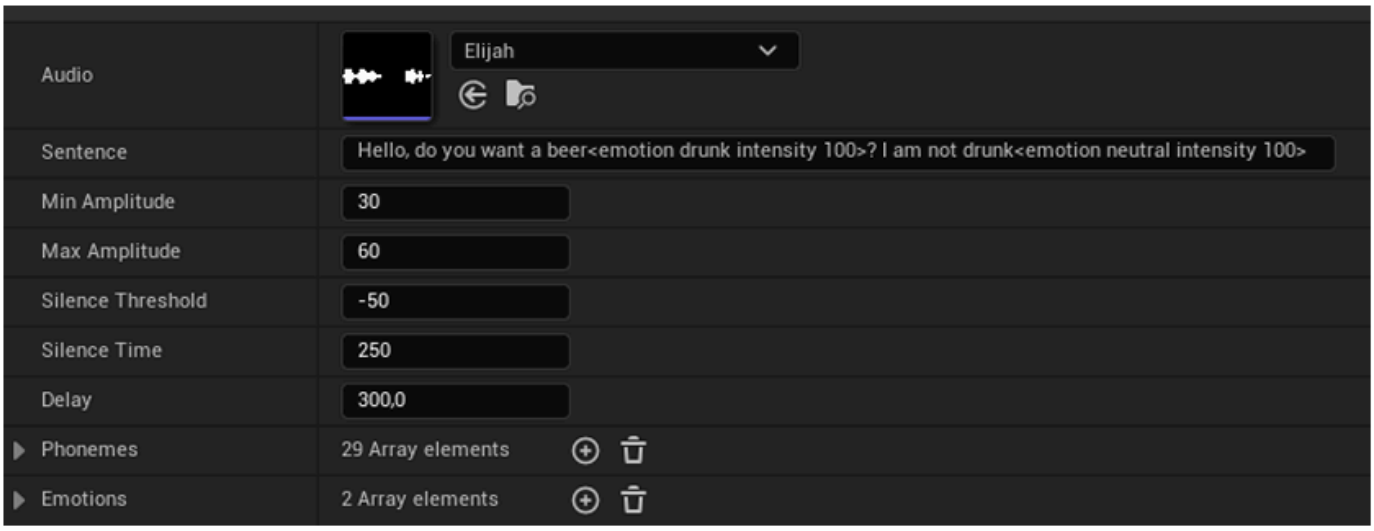
Phoneme	The phoneme name.
Phoneme Table	The Phoneme Data Table used to retrieve Morph Targets values.
Return Value	The list of all Morph Targets used for the phoneme and their values.

8. Data Assets

A Data Asset is an Unreal Asset used to store Data. The Geppetto Plugin uses custom defined Data Assets written in C++.

- **Geppetto Data Asset**

Geppetto Data Assets are used to store and save the generated phonemes and emotions generated from the API within the Editor in order to use them in the game later. You can use the Geppetto SoundWave Player node Play Data Asset to animate the data contained in the Data Asset. All fields are Blueprint Read-Only, but you can use the node Save Geppetto Phonemes (as Asset) to create a new Data Asset with Blueprint.
Please note that the node is only available within the Editor



Audio	The audio associated with the lip sync.
Sentence	*(Info only)* The sentence used for generation.
Min Ampl	*(Info only)* The minimum amplitude value used for generation.
Max Ampl	*(Info only)* The maximum amplitude value used for generation.
Silence Threshold	*(Info only)* The silence threshold value (dB) used for generation.
Silence Time	*(Info only)* The silence time (in milliseconds) used for generation.
Delay	*(Info only)* The delay applied to phonemes and emotions after generation.
Phonemes	The phonemes list generated by the API, used to animate lip sync.
Emotions	The emotions list generated by the API, used to animate emotions.

9. Structs

Here you can find the list and details of all structs defined by the Geppetto Plugin.

9.1. Geppetto Phoneme

A struct containing all parameters to play a Geppetto Phoneme.
All variables are Blueprint read-write :

Name	The phoneme name.
Amplitude	The phoneme amplitude, range 0 - 100.
Time	The phoneme animation play time, matching the audio for synchronization.

9.2. Geppetto Emotion

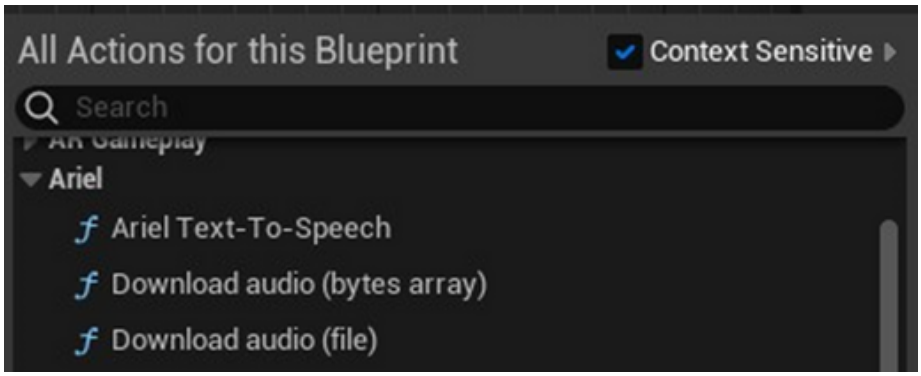
Astruct containing all parameters to play a Geppetto Emotion.
All variables are Blueprint read-write :

Name	The emotion name.
Intensity	The emotion intensity, range 0 - 100.
Transition time	The emotion transition time, in seconds.
Transition function	See *Geppetto Emotion Transition*.
Time	The emotion animation play time, matching the sentence tag for sync.

5.1.4. Ariel Voice Generation (Ref 15.2)

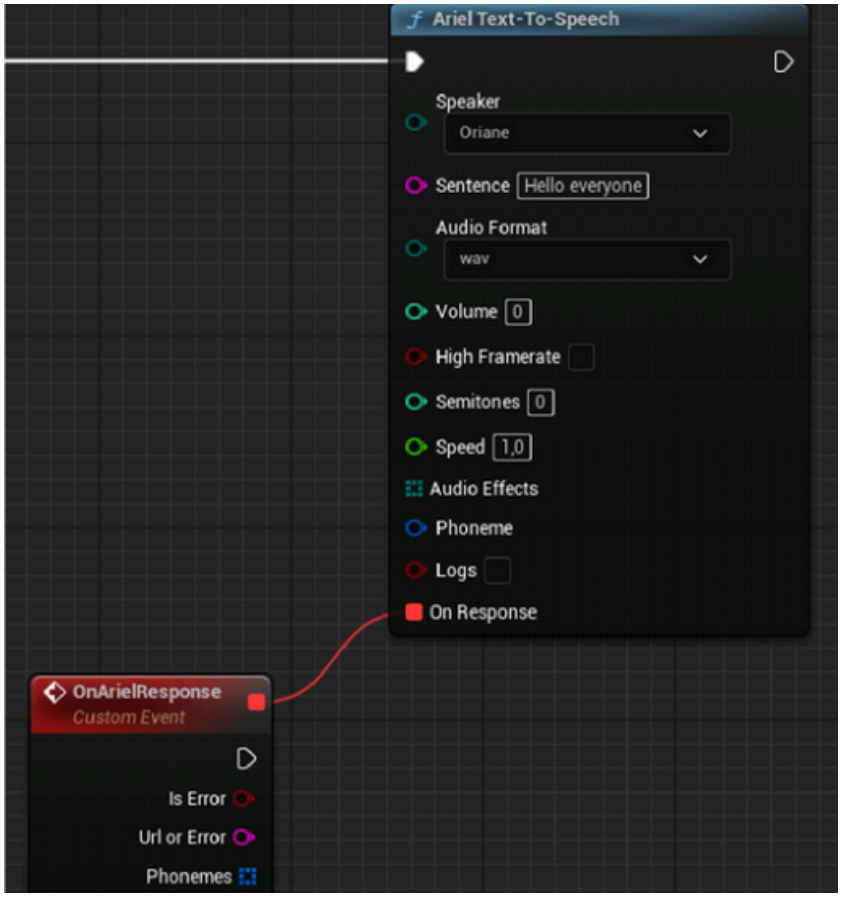
1. How to use the Ariel plugin in Blueprints (Runtime & Editor)

The plugin have 3 functions that can be used directly in Blueprints:



2. Ariel Text-To-Speech

This function calls the Ariel API using HTTPS request with all defined Ariel parameters (speaker, sentence, format, effects, ...). The response is sent through a delegate :



1. More information about Ariel function parameters :

Speaker : Defines the voice used to play the sentence. Hover the cursor on an item on the list to see more details about it. Only the voice "Zenaya" is available in the free version!

⇒ **You can find the list of all speakers at the end of this document!**

Sentence : Defines the sentence that the API needs to generate the speech for.

Audio format : Defines the format of the audio played. Supported formats are wav, mp3, mp4 and flac.

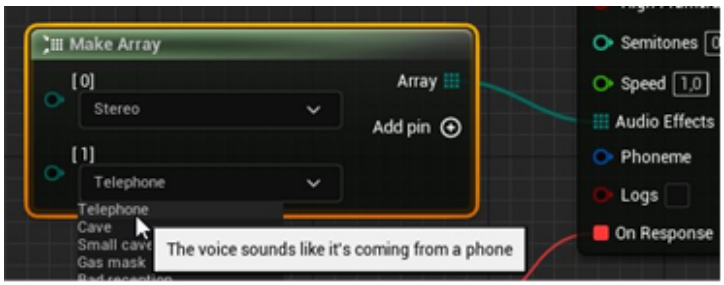
Volume : Adjust the volume of the speaker by the specified dB, i.e: +3dB, -5dB.

High framerate : Generate a speech using 44.1 KHz frequency instead of 22.05 KHz (default)

Semitones : Defines the pitch of the generated speech as semitones (1 octave is +-12st).

Speed : Defines the speed generated speech. Normal speed is 1.0.

Audio effects :
Defines all effects who will be applied to the audio. You can combine effects using an array with multiple values. Hover the cursor on an item on the list to see more details about it.



Available effects :
- Telephone
- Small cave
- Bad reception
- Stereo

Logs : Indicate if the logs should be printed to the console (Dev only).

More information about Ariel return response :

You can bind and handle Ariel response using the node "Add custom event" or "Create Event".

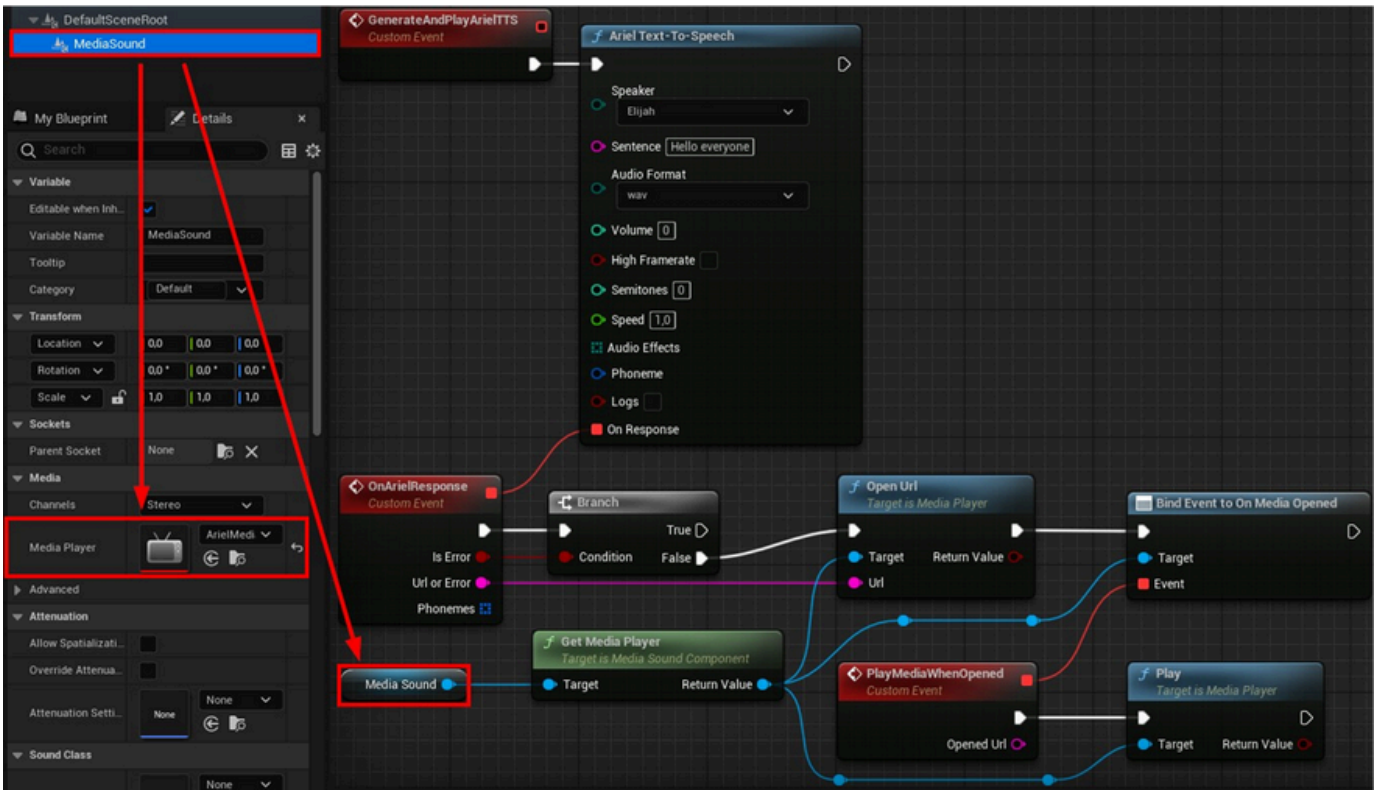
Is Error : Indicate if there was an error while processing the request.

URL or Error :
Thegenerated audio URL (if success) or the error message (if error).

The generated audio URL can be passed to an Unreal "Media Player" or "Media steam" in order to be played at runtime.

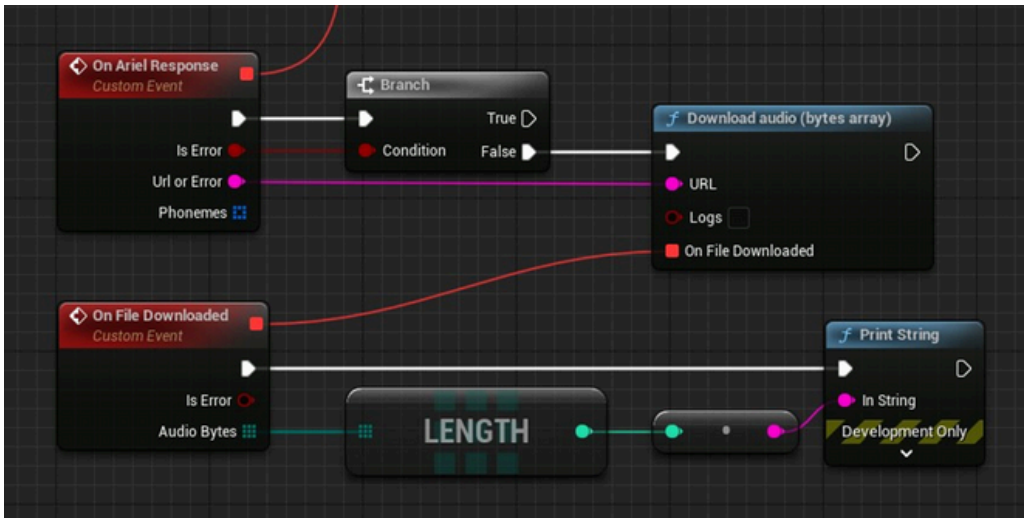
The image on the next page shows an example generating and playing the audio at runtime.

For more informations, please see : https://blueprintue.com/blueprint/te9qcpy5



2. Download audio as bytes array

This function takes the URL generated from the previous function (Ariel Text-To-Speech), downloads the audio and sends it to the event delegate as a byte array:



More information about Download function parameters :

URL : The URL of the audio file who needs to be downloaded.

Logs : Indicate if the logs should be printed to the console (Dev only).

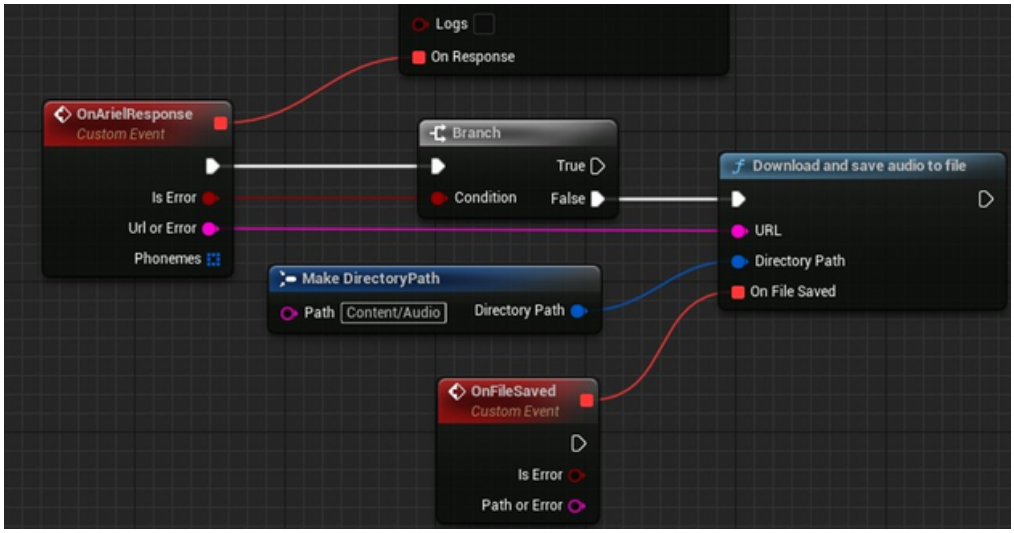
More information about Download return response :

You can bind and handle download response using nodes “Add custom event” or “Create event”.

Is Error : Indicate if there was an error while processing the request.
Audio Bytes : The audio bytes, stored in an array. (an empty array if there were an error).

3. Download audio and save it to a file

This function takes the URL generated from the previous function (Ariel Text-To-Speech), downloads the audio and stores it in the specified directory. The response is sent through a delegate:



- ## More information about Download function parameters :

  URL : The URL of the audio file who needs to be downloaded.

  Directory path : Define the directory where the downloaded file needs to be stored.

- ## More information about Download return response :

  You can bind and handle download response using nodes “Add custom event” or “Create event”.

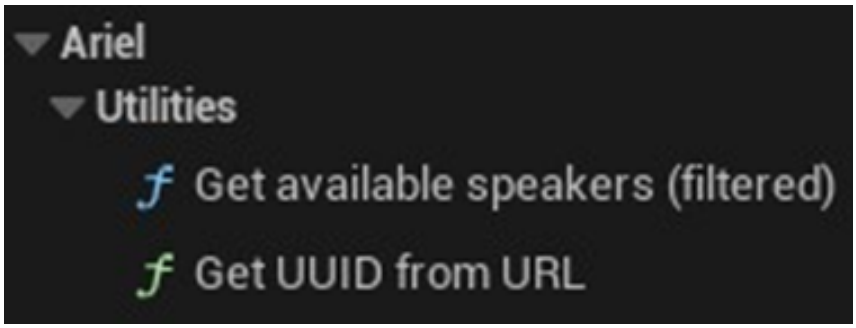
  Is Error : Indicate if there was an error while processing the request.

  Path or Error : The audio file location (if successful, with filename) or the error message.

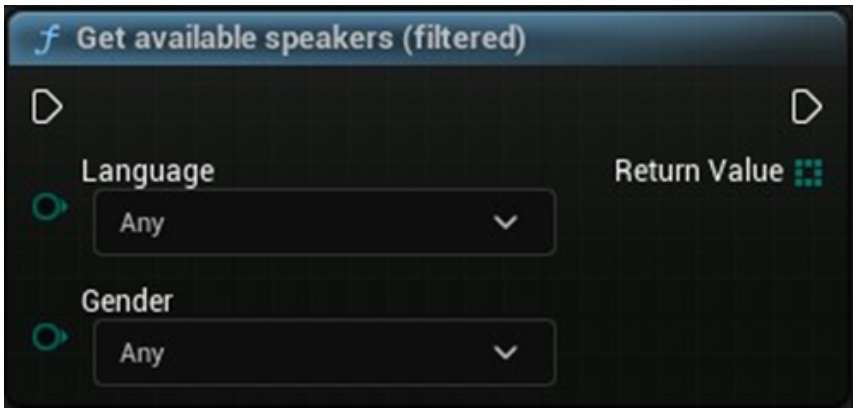
  For more informations, please see : https://blueprintue.com/blueprint/8hvffckk/

4. Utilities

In addition to those functions, you can find 2 other utilities nodes located in Ariel > Utilities:



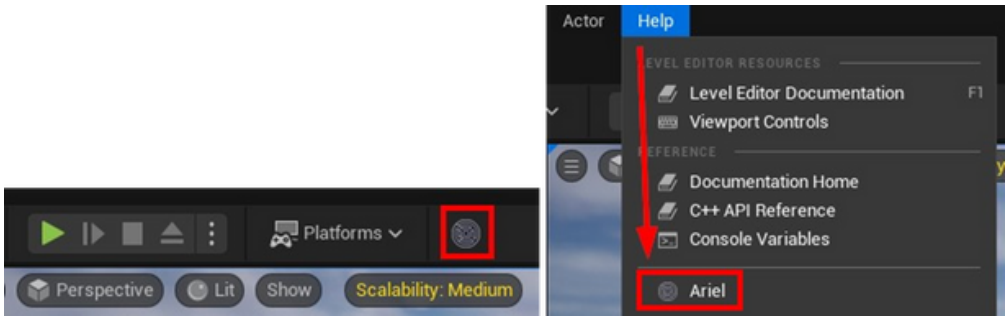
Get available speakers (filtered): Retrieve all speakers matching the filter conditions. You can filter speakers by language and by gender.



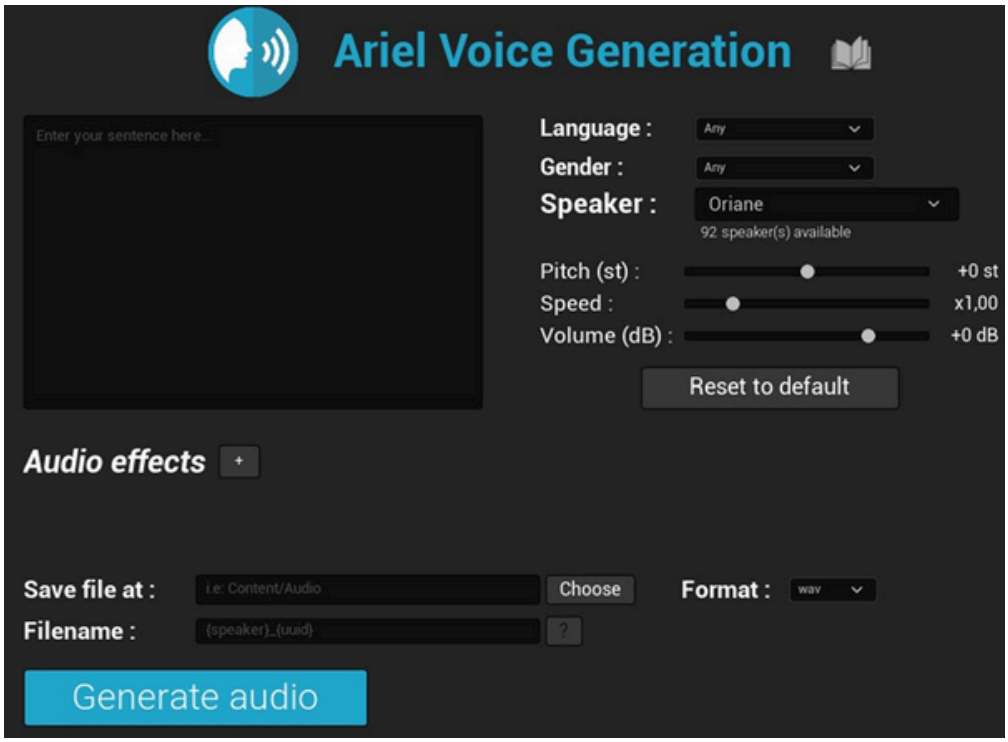
Get UUID from URL : Extract the UUID from the URL generated by the node "Ariel Text-To-Speech". You may prefer using GUID directly (available in both C++ and blueprint with the Unreal library).

5. How to use the Ariel plugin in Editor (with graphical interface)

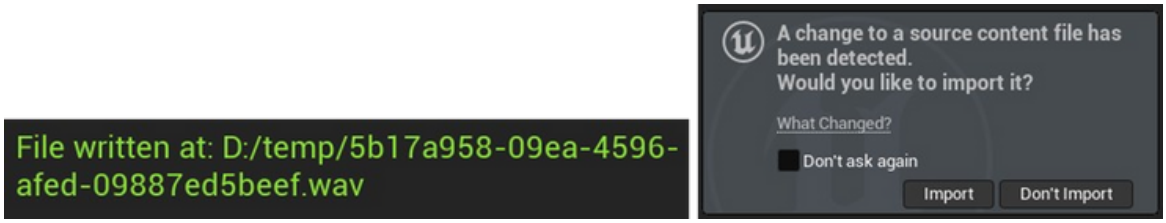
If you want to generate audio sentences that will be saved as a file, you can use the "Editor Utility Widget" included in the plugin. You can open the Window by click the Ariel icon button or in the menu Help > Ariel :



The Ariel Editor Windows should open. Here you can adjust and fill all settings for Ariel. Please read Ariel Text-To-Speech chapter to find more information about the Ariel parameters.

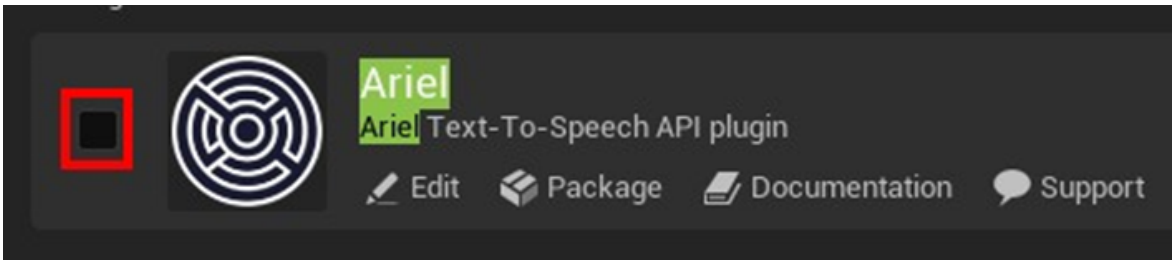


Then, press the "Generate audio" button and wait for the API to generate and write it into a f ile. If you enter a path located inside your unreal project, you should see a popup asking you if you want to import the file on the bottom right of the screen :



6. How to package a project with the Ariel plugin

If you have generated and imported all the sentences using the ariel editor window and you don't want to generate new speech at runtime, you can disable the plugin by going to Edit > Plugins, then search for "Ariel" in the search box and uncheck the checkbox for Ariel plugin:



Warning : If you use Unreal objects that were in the demo (like demo Actor, MediaPlayer or sound attenuation), you first need to move or replace them by object in your project directory. Otherwise, the project might have errors when the plugin gets disabled.

You may need to restart UE to apply the changes.

7. Pause tag

In your text, you can enter a silence tag for a custom pause. Write <pause Xs> or <pause Xms> where X is the duration in seconds or milliseconds.
For example "hi how are you?
<pause 10s> I am Jane".

→ More tags will be added in the next updates of our plugin !

5.2. Tools and Technologies for AI

5.2.1. Datasets

We have collected various amounts of labeled datasets from different sources, that involve determining the emotional tone behind words to understand the attitudes, opinions, and emotions of the user. The size of the data became 125,128 rows after each data point was processed and analyzed which was 143,555 rows before (which could be a sentence, a tweet, a review, etc.) is labeled as sad, happy, neutral, joy, fear, and surprise.

5.2.2. LSTM

Long Short-Term Memory is an improved version of recurrent neural network designed by Hochreiter & Schmidhuber.
A traditional RNN has a single hidden state that is passed through time, which can make it difficult for the network to learn long-term dependencies. LSTMs model address this problem by introducing a memory cell, which is a container that can hold information for an extended period.
LSTM architectures are capable of learning long-term dependencies in sequential data, which makes them well-suited for tasks such as language translation, speech recognition, and time series forecasting.

1. Usage of LSTMs

Training LSTMs with their lstm model architecture removes the vanishing gradient problem but faces the exploding gradient issue. The vanishing gradient causes weights to become too small, underfitting the model. The exploding gradient makes weights too large, overfitting the model.

LSTMs can be trained using Python frameworks like TensorFlow, PyTorch, and Theano. However, training deeper LSTM networks with the architecture of lstm in deep learning requires GPU hardware, similar to RNNs.

The lstm model architecture enables LSTMs to handle long-term dependencies effectively. This makes them widely used for language generation, voice recognition, image OCR, and other tasks leveraging the lstm model architecture. Additionally, the architecture of lstm in deep learning is gaining traction in object detection, especially scene text detection.

2. The architecture of LSTM (Ref 4.1)

LSTMs architecture deal with both Long Term Memory (LTM) and Short Term Memory (STM) and for making the calculations simple and effective it uses the concept of gates.

- **Forget Gate :** LTM goes to forget gate and it forgets information that is not useful.
- **Learn Gate :** Event (current input) and STM are combined together so that necessary information that we have recently learned from STM can be applied to the current input.
- **Remember Gate :** LTM information that we haven't forget and STM and Event are combined together in Remember gate which works as updated LTM.
- **Use Gate :** This gate also uses LTM, STM, and Event to predict the output of the current event which works as an updated STM.

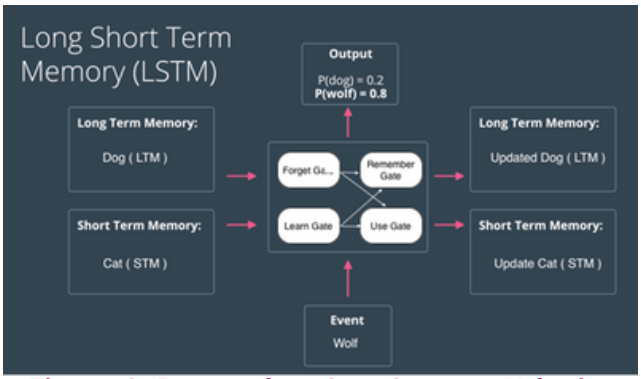


Figure 3. Remember GateSource: Udacity

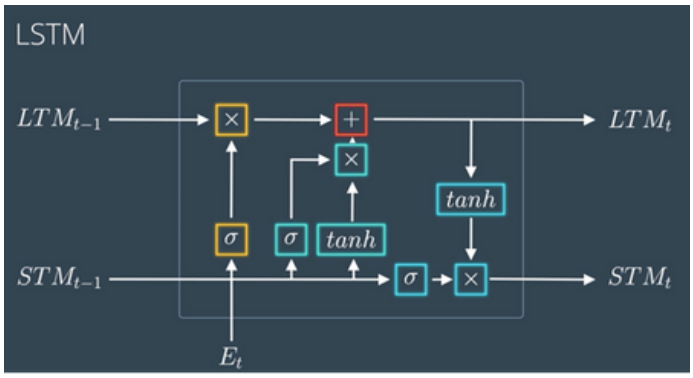


Figure 4. LSTM architecture diagram

3. Bidirectional Network

Now, when we are dealing with long sequences of data and the model is required to learn relationship between future and past word as well. we need to send data in that manner. To solve this problem bidirectional network was introduced. We can use bidirectional network with LSTM and well as RNN but dur to limitations of

In bidirectional LSTM we give the input from both the directions from right to left and from left to right . **Make a note this is not a backward propagation this is only the input which is given from both the side**. So, the question is how the data is combined in output if we are having 2 inputs.

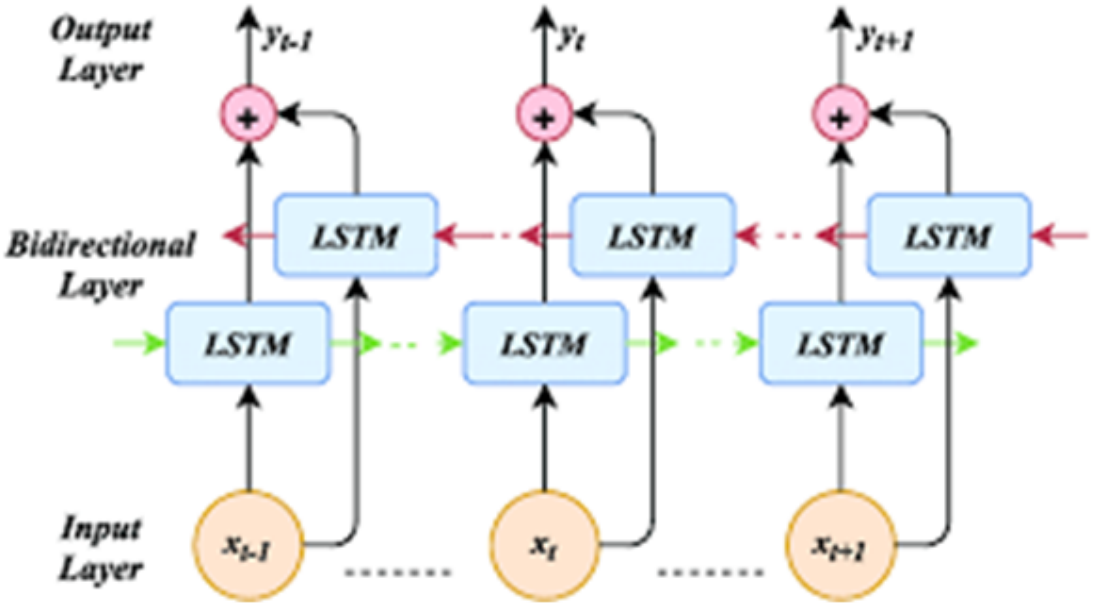
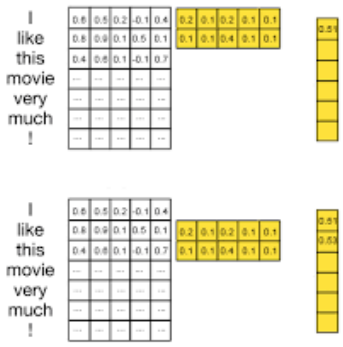


Figure 5. The bidirectional LSTM

5.2.3. CNN in Sentiment Analysis (Ref 5.1)

CNN is a class of deep, feed-forward artificial neural networks (where connections between nodes do not form a cycle) & use a variation of multilayer perceptrons designed to require minimal preprocessing. These are inspired by animal visua cortex



Convolutional Neural Networks (CNN) are a form of artificial neural networks that can detect information in different positions with excellent accuracy.

This model has solved several problems in image processing and automatic natural language processing such as opinion analysis, answers to questions, text summary, etc..... It is characterized by a particular architecture to facilitate learning.

A convolutional neural network is a multilayer network, so that the output of one layer will be the input of the next layer. It is usually composed of an input, one to several hidden layers and an output.

5.2.4. Attention Mechanisms (Ref 6.2)

an attention mechanism is an Encoder-Decoder kind of neural network architecture that allows the model to focus on specific sections of the input while executing a task. It dynamically assigns weights to different elements in the input, indicating their relative importance or relevance. By incorporating attention, the model can selectively attend to and process the most relevant information, capturing dependencies and relationships within the data. This mechanism is particularly valuable in tasks involving sequential or structured data, such as natural language processing or computer vision, as it enables the model to effectively handle long-range dependencies and improve performance by selectively attending to important features or contexts.

Recurrent models of visual attention use reinforcement learning to focus attention on key areas of the image. A recurrent neural network governs the peek network, which dynamically selects particular locations for exploration over time. In classification tasks, this method outperforms convolutional neural networks. Additionally, this framework goes beyond image identification and may be used for a variety of visual reinforcement learning applications, such as helping robots choose behaviours to accomplish particular goals. Although the most basic use of this strategy is supervised learning, the use of reinforcement learning permits more adaptable and flexible decision-making based on feedback from past glances and rewards earned throughout the learning process.

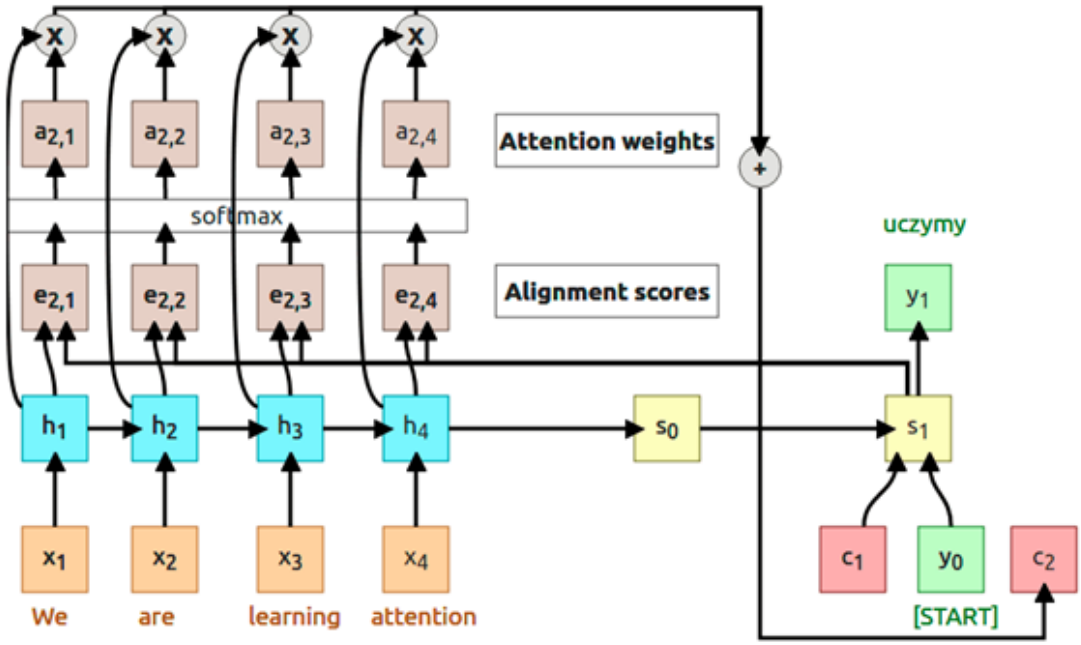


Figure 6. Attention Mechanisms

Attention Mechanisms in Deep Learning Models for Sentiment Analysis (Ref 6.1)

Attention-based methods for deep neural networks constitute a technique that has attracted increased interest in recent years. Attention mechanisms can focus on important parts of a sequence and, as a result, enhance the performance of neural networks in a variety of tasks, including sentiment analysis, emotion recognition, machine translation and speech recognition. In this work, we study attention-based models built on recurrent neural networks (RNNs) and examine their performance in various contexts of sentiment analysis. Self-attention, global-attention and hierarchical-attention methods are examined under various deep neural models, training methods and hyperparameters.

Even though attention mechanisms are a powerful recent concept in the field of deep learning, their exact effectiveness in sentiment analysis is yet to be thoroughly assessed. A comparative analysis is performed in a text sentiment classification task where baseline models are compared with and without the use of attention for every experiment. The experimental study additionally examines the proposed models' ability in recognizing opinions and emotions in movie reviews. The results indicate that attention-based models lead to great improvements in the performance of deep neural models showcasing up to a 3.5% improvement in their accuracy.

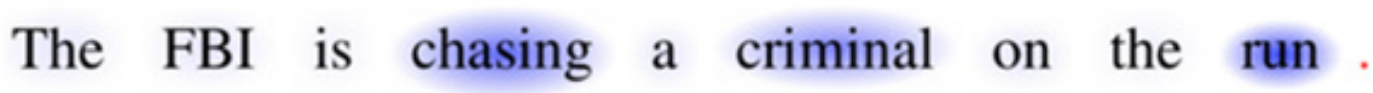


Figure 7. Illustration of word weighting of past memory using an attention mechanism

5.2.5. RoBERTa model (Ref 7.1)

The Roberta model is built upon the Transformer architecture, which plays a foundational role in enabling efficient language representation. Transformers utilize self-attention mechanisms to capture contextual relationships within textual data, allowing the model to understand dependencies between different words and phrases in a sentence. This mechanism enables Roberta to effectively learn rich contextual representations of language, contributing to its high performance across various natural language processing tasks.

Roberta's pre-training with MLM helps it's develop a deep understanding of language, which comes in handy when fine-tuning the model for various NLP tasks.

Hugging Face Transformers Library :
Utilize the Hugging Face library, which provides pre-trained transformer models like Roberta and tools for fine-tuning them on custom tasks.

PyTorch or TensorFlow :
Implement the fine-tuning process using deep learning frameworks like PyTorch or TensorFlow, which offer efficient computation for training large language models.

MODEL ARCHITECTURE : (Ref 7.2)
The RoBERTa model is based on the Transformer architecture, which is explained in the paper Attention is All You Need. The Transformer architecture is a type of neural network that is specifically designed for processing sequential data, such as natural language text.
The architecture is nearly comparable to that of BERT, with a few slight modifications to the training procedure and architecture to enhance the results in comparison to BERT.

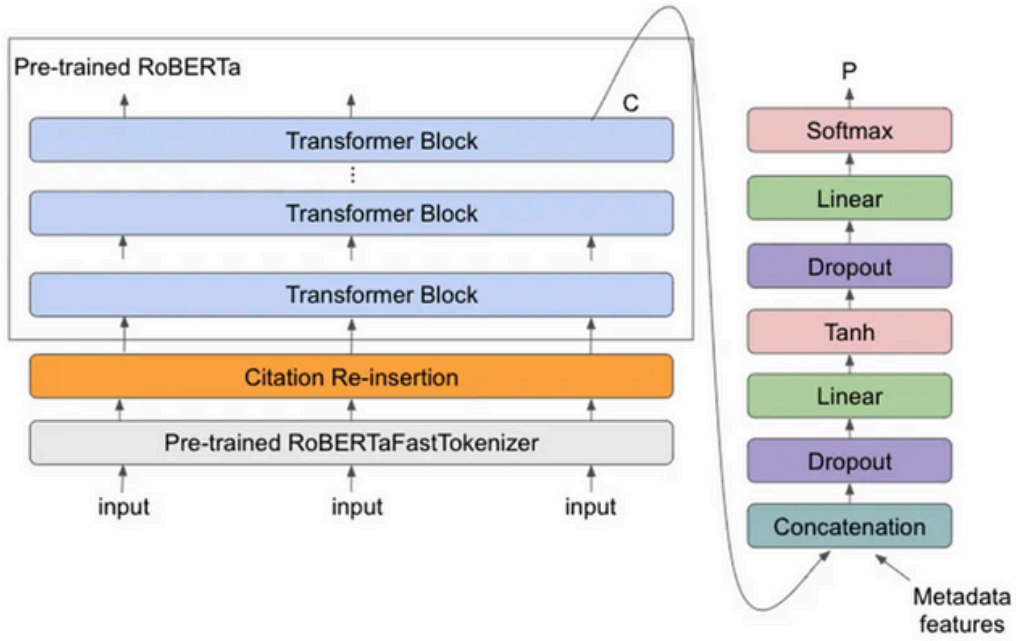


Figure 8. RoBERTa Architecture

The RoBERTa model consists of a series of self-attentional and feed-forward layers. The self-attention layers allow the model to weigh the importance of different tokens in the input sequence and compute representations that take into account the context provided by the entire sequence. The feed-forward layers are used to transform the representations produced by the self-attention layers into a final output representation.

A portion of each sentence's tokens are randomly masked in each layer of the RoBERTa model during training, and the model is then taught to predict the masked tokens based on the context provided by the tokens that aren't masked. In this pre-training stage, the model can acquire a detailed representation of the language that can be tailored for particular NLP tasks.

Fine-Tuning Steps :
Sentiment analysis is a natural language processing (NLP) task that involves determining the sentiment or emotion expressed in a piece of text. In this project, we will leverage the power of the Roberta model, a state-of-the-art transformer-based language model, to develop a sentiment.

analysis model capable of classifying text into specific emotional categories 'anger' 'fear' 'joy' 'neutral' 'sadness' and 'surprise'. Initialization: Load the pre-trained Roberta model and Roberta Tokenizer.

Training Configuration :
Set up training parameters such as batch size, learning rate, and number of epochs.

Training Loop :
Iterate through the emotional dataset, feeding batches of data into the model and adjusting the model's weights using backpropagation.

Accuracy :
Measure the model's accuracy in predicting emotional labels on unseen data.

F1 Score, Precision, Recall :
Evaluate the model's performance using standard classification metrics.

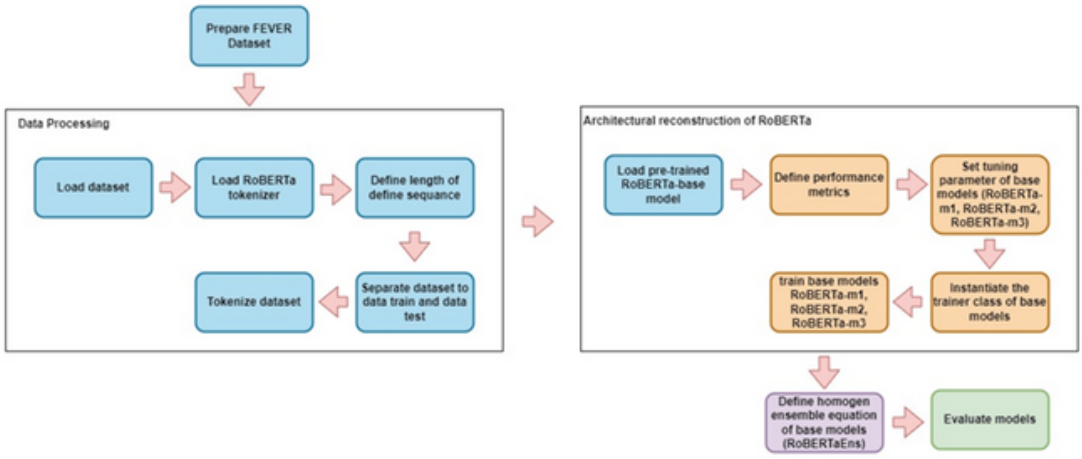


Figure 9. RoBERTaEns Flowchart.

5.2.6. Gemini 1.5 Flash

The Gemini 1.5 Flash model, part of Google's Gemini family of AI models, is designed for high-speed, efficient AI tasks. It is a lighter, faster, and more cost-effective version of the Gemini 1.5 Pro, making it ideal for applications that require lower latency and higher efficiency at scale. Despite its reduced size, Gemini 1.5 Flash maintains impressive capabilities, including multimodal reasoning across text, audio, and video. (Ref 9.3)

1. Mixture-of-Experts (MoE) Architecture

Description : The MoE architecture divides the model into smaller, specialized neural networks called "experts." These experts are selectively activated based on the input type, enhancing efficiency by using only the relevant pathways for each task.
Benefits : Improves processing speed and reduces computational requirements, making the model lightweight and fast.

2. Long Context Window

Description : Gemini 1.5 Flash supports a one-million-token context window, allowing it to handle extensive input data such as lengthy documents, large datasets, and hours of video.
Benefits : Enables the model to understand and process vast amounts of information in a single session, ideal for applications requiring detailed analysis and comprehension.

3. Distillation Process

Description : The model is derived from Gemini 1.5 Pro through a distillation process, which transfers the essential knowledge and capabilities from the larger model to a more compact version.
Benefits : Ensures that the compact model retains advanced features like logical reasoning, multi-turn conversation capabilities, and multimodal understanding while being more efficient.

4. Google AI Studio and Vertex AI

Description : These platforms provide access to Gemini 1.5 Flash, allowing developers to integrate the model into various applications. They support a wide range of input types (text, audio, images, video) and offer adjustable safety settings for reliable AI deployment.
Benefits : Facilitates easy integration and deployment of the model in diverse environments, enhancing its accessibility and usability for developers and enterprises.

5. API Enhancements

Description : Recent updates have increased rate limits and optimized the API for high volume and low latency requests, ensuring rapid response times for large-scale applications.
Benefits : Improves the model's performance in real-time applications, making it suitable for scenarios requiring immediate feedback and high throughput.

6. Safety and Ethical AI Tools

Description : Incorporates adjustable safety settings and ethical AI guidelines to ensure responsible use and deployment of the model.
Benefits : Ensures that the AI operates within ethical boundaries, reducing the risk of misuse and promoting safe application development.

5.2.7. PaLM 2 (Ref 8.1)

PaLM 2 is our next generation large language model that builds on Google's legacy of breakthrough research in machine learning and responsible AI.

It excels at advanced reasoning tasks, including code and math, classification and question answering, translation and multilingual proficiency, and natural language generation better than our previous state-of-the-art LLMs, including PaLM. It can accomplish these tasks because of the way it was built – bringing together compute-optimal scaling, an improved dataset mixture, and model architecture improvements.

PaLM 2 is grounded in Google's approach to building and deploying AI responsibly. All versions of PaLM 2 are evaluated rigorously for potential harms and biases, capabilities and downstream uses in research and in-product applications. PaLM 2 is used in other state-of-the-art models, like Sec-PaLM. We continue to implement the latest versions of PaLM 2 in generative AI tools like the PaLM API.

Setting Up the Environment :

Instructions on setting up the development environment, including obtaining API keys and configuring the API for use2.

Using the API :

A step-by-step guide on how to send requests to the API and handle responses. This would include code examples and explanations of parameters and options available for text generation2.

Impact :

- The slow response times likely caused delays and disruptions in users' workflows.
- The lack of performance may have led users to abandon tasks or seek alternative solutions

Detailed Description :

- Palm2's response time to user queries was sluggish, hindering a smooth and efficient interaction.
- The overall performance of the system fell short of user expectations, resulting in dissatisfaction.

Summary :

- Users experienced slow response times and inadequate performance when interacting with Pal2m.
- This frustration led to a negative user experience.

Performance Comparison: Gemini 1.5 Flash vs. Palm2

1. Latency	
GEMINI 1.5 FLASH	**PALM 2**
Optimized for low latency, making it highly efficient in real-time applications. The model is designed to deliver responses quickly, which is crucial for applications that require immediate feedback (Google Developers Blog) (Model & API Analysis).	Generally, Palm2's latency is higher compared to Gemini 1.5 Flash. This is due to its complex processing capabilities which take more time to generate responses (AI StartUps Cheatsheet).

2. Throughput	
GEMINI 1.5 FLASH	**PALM 2**
Gemini 1.5 Flash: High throughput, capable of generating tokens at a rapid rate. This makes it suitable for handling large volumes of data efficiently, such as processing extensive text or video content (Model & API Analysis).	While Palm2 has good throughput, it may not match the speed of Gemini 1.5 Flash, especially under heavy loads. Its strength lies more in the quality and depth of its processing rather than sheer speed (Beebom).

3. Multimodal Capabilities	
GEMINI 1.5 FLASH	**PALM 2**
Supports multimodal input, including text, audio, and video. However, its performance in this area, particularly in tasks involving image recognition and processing, is not as advanced as some other models (Model & API Analysis) (Beebom).	Excels in multimodal tasks, including complex image and text processing. It performs better in tasks that require a high degree of reasoning and understanding across different types of data (Beebom).

4. Context Window	
GEMINI 1.5 FLASH	**PALM 2**
Features an extensive one-million-token context window, which is significantly larger than many models. This allows it to process and understand vast amounts of data in a single session, making it highly effective for comprehensive data analysis and long-context understanding (Google Developers Blog) (Model & API Analysis	While Palm2 also supports large context windows, its context window size is generally smaller compared to Gemini 1.5 Flash. This can limit its ability to handle extremely large datasets in a single session (AI StartUps Cheatsheet).

5. Use Cases	
GEMINI 1.5 FLASH	**PALM 2**
Ideal for applications requiring fast processing and high throughput, such as large-scale API services, real-time data analysis, and applications where quick response times are critical (Google Developers Blog) (Model & API Analysis).	Better suited for applications that demand deep reasoning, detailed understanding, and complex multimodal processing. This includes advanced coding tasks, scientific research, and detailed image and text analysis (AI StartUps Cheatsheet) (Beebom).

6. Cost Efficiency	
GEMINI 1.5 FLASH	**PALM 2**
Generally more cost-effective due to its optimized performance for high-speed, high-volume tasks. The efficiency in token generation and processing helps in reducing operational costs over time (Google Developers Blog).	May incur higher costs due to its comprehensive processing capabilities. The model's detailed and complex analysis requires more computational resources, which can increase overall costs (Beebom).

7. Accuracy and Reliability	
GEMINI 1.5 FLASH	**PALM 2**
While it performs well in terms of speed and efficiency, its accuracy in certain complex tasks, particularly those requiring high levels of reasoning or understanding, may be lower than Palm2 (Beebom).	Known for higher accuracy in complex tasks, especially those involving detailed reasoning and multimodal understanding. It is more reliable in tasks that require intricate and nuanced processing (AI StartUps Cheatsheet) (Beebom).

Gemini 1.5 Flash :

Best for high-speed, high-throughput applications with large datasets and real-time processing needs. It offers cost efficiency and quick response times but may lag in detailed multimodal tasks.

Palm2 :

Ideal for tasks requiring deep reasoning, multimodal processing, and high accuracy. It handles complex tasks well but may come with higher latency and costs.

5.2.8. Speech Recognition

The SpeechRecognition module in Python is a powerful and easy-to-use library that allows developers to convert spoken language into written text1. It supports several engines and APIs, both online and offline2. This includes CMU Sphinx (which works offline), Google Speech Recognition, Google Cloud Speech API, Wit.ai, Microsoft Azure Speech, Microsoft Bing Voice Recognition (Deprecated), Houndify API, IBM Speech to Text, Snowboy Hotword Detection (works offline), Tensorflow, Vosk API (works offline), and OpenAI whisper (works offline).

A Speech-to-Text API synchronous recognition request is the simplest method for performing recognition on speech audio data. Speech-to-Text can process up to 1 minute of speech audio data sent in a synchronous request. After Speech-to-Text processes and recognizes all of the audio, it returns a response.

A synchronous request is blocking, meaning that Speech-to-Text must return a response before processing the next request. Speech-to-Text typically processes audio faster than realtime, processing 30 seconds of audio in 15 seconds on average. In cases of poor audio quality, your recognition request can take significantly longer.

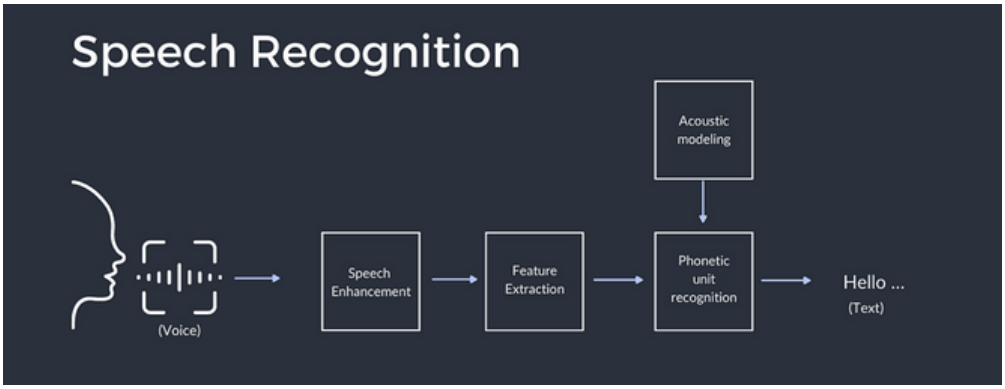


Figure 10. Speech Recognition.

ASR under-the-hood and Google Speech-to-Text!

1.Have a high level understanding of how the ASR systems function.
2.How do we measure the outcome to improve the same?

ASR — System Architecture

This is an extremely simplified diagram to get a give a high level understanding of the system.

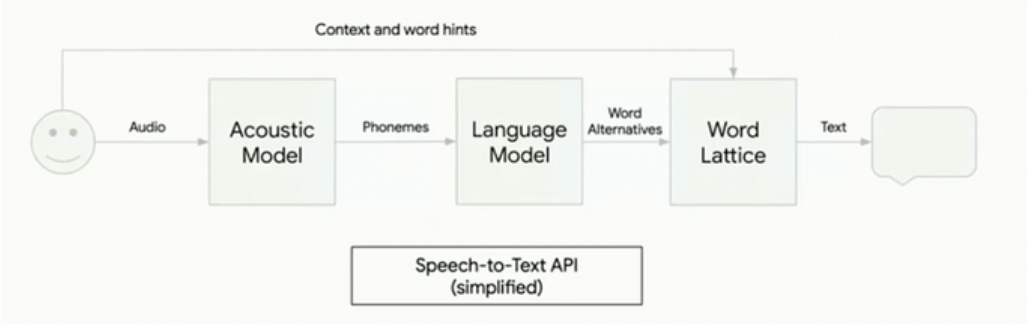


Figure 11. Google Speech-to-text

The system takes an audio input in from the user as chunks of 10–25 milliseconds speech frames to process and develop what's called an acoustic model. The Acoustic model looks at the audio wave forms or the spectrogram and converts these waveforms into phonemes. (Ref 10.3)

A phoneme is distinct unit of sound in a specified language that makes up a word. The example below shows 3 distinct phonemes that make up the words "five" and "four" in english. These word to phoneme mappings are developed by experts and can be thought of equivalent of alphabets that make up the word but using phonetics.

These Phonemes word predictions are then fed into what is known as a Language model.

Language Model provides a way interpret how the words should be ordered for a particular language and these models are developed by training on lots and lots of text in that particular language. Language Model looks at the groupings and series of phonemes and tries to transcribe the output as a series of word lattice which are sent as an output.

So, in a very simple way, thats how the speech gets transcribe into text. (Ref 10.3)

Implementing Speech Recognition with Python :

A basic implementation using the SpeechRecognition library involves several steps:

- **Audio Capture :** Capturing audio from the microphone using PyAudio.
- **Audio Processing :** Converting the audio signal into data that the SpeechRecognition library can work with.
- **Recognition :** Calling the recognize_google() method (or another available recognition method) on the SpeechRecognition library to convert the audio data into text.

Challenges and Considerations:

While implementing speech recognition, developers might face challenges such as background noise interference, accents, and dialects. It's crucial to consider these factors and test the application under various conditions. Furthermore, privacy and ethical considerations must be addressed, especially when handling sensitive audio data.

5.2.9. LangChain (Ref 11.1)

The rise of embeddings in natural language processing (NLP) has brought about the need for efficient databases to handle the storage and retrieval of these embeddings.

Embeddings are numerical representations of textual data that capture semantic meanings and relationships, making them essential for various NLP tasks such as similarity search, classification, and clustering.

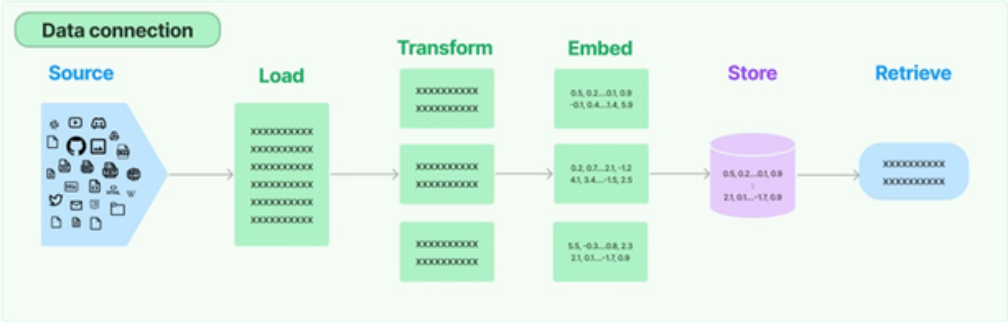


Figure 12. Steps of Retrieve Data.

Document loaders :

load documents from many different sources. LangChain provides over 100 different document loaders as well as integrations with other major providers in the space, like AirByte and Unstructured. LangChain provides integrations to load all types of documents (HTML, PDF, code) from all types of locations (private S3 buckets, public websites).

Text Splitters :

Once you've loaded documents, you'll often want to transform them to better suit your application. The simplest example is you may want to split a long document into smaller chunks that can fit into your model's context window. LangChain has several built-in document transformers that make it easy to split, combine, filter, and otherwise manipulate documents.

When you want to deal with long pieces of text, it is necessary to split that text into chunks In the model split based on characters (by default “”) and measure chunk length by number of characters.

Embeddings :

Embeddings capture the semantic meaning of the text, allowing you to quickly and efficiently find other similar pieces of text. LangChain provides integrations with over 25 different embedding providers and methods, from open-source to proprietary API, allowing you to choose the one best suited for your needs. LangChain provides a standard interface, allowing you to easily swap between models.

Embeddings create a vector representation of a piece of text. This is useful because it means we can think about text in the vector space, and do things like semantic search where we look for pieces of text that are most similar in the vector space.

Vector stores :

One of the most common ways to store and search over unstructured data is to embed it and store the resulting embedding vectors, and then at query time to embed the unstructured query and retrieve the embedding vectors that are 'most similar' to the embedded query. A vector store takes care of storing embedded data and performing vector search for you.

In the model use the chroma vector database, which runs on your local machine as a library.

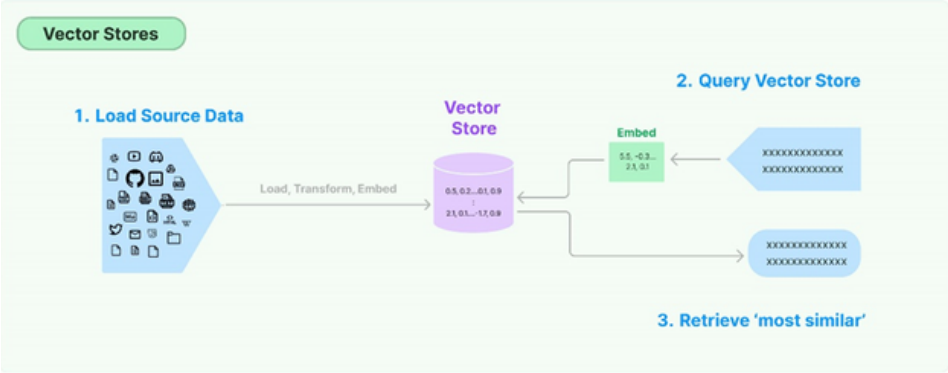


Figure 13. Retrieve Data from Vector Stores

Retrievers :

Once the data is in the database, you still need to retrieve it. LangChain supports many different retrieval algorithms and is one of the places where we add the most value. LangChain supports basic methods that are easy to get started - namely simple semantic search.

A retriever is an interface that returns documents given an unstructured query, It is more general than a vector store, A retriever does not need to be able to store documents, only to return (or retrieve) them Vector stores can be used as the backbone of a retriever, but there are other types of retrievers as well Retrievers accept a string query as input and return a list of Documents as output.

In the model use MultiQueryRetriever :

Distance-based vector database retrieval embeds (represents) queries in high-dimensional space and finds similar embedded documents based on “distance”.

However, retrieval may produce different results with subtle changes in query wording or if the embeddings do not capture the semantics of the data well. Prompt engineering/tuning is sometimes done to manually address these problems, but can be tedious.

The MultiQueryRetriever()automates the process of prompt tuning by using an LLM to generate multiple queries from different perspectives for a given user input query. For each query, it retrieves a set of relevant documents and takes the unique union across all queries to get a larger set of potentially relevant documents.

By generating multiple perspectives on the same question, the MultiQueryRetriever ()might be able to overcome some of the limitations of distance-based retrieval and get a richer set of results.

5.2.10. BLIP

Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation

BLIP, which stands for Bootstrapping Language-Image Pre-training, is a vision-language pre-training framework developed by Salesforce Research.

It aims to advance the performance of various vision-language tasks by effectively handling noisy image-text data typically sourced from the web.

BLIP is designed to excel in both vision-language understanding and generation tasks, overcoming the limitations of previous models that generally specialize in only one of these areas.

Vision-Language Pre-training (VLP) has advanced the performance for many vision-language tasks. However, most existing pre-trained models only excel in either understanding-based tasks or generation-based tasks.

Furthermore, performance improvement has been largely achieved by scaling up the dataset with noisy image-text pairs collected from the web, which is a suboptimal source of supervision.

In this paper, we propose BLIP, a new VLP framework which transfers flexibly to both vision-language understanding and generation tasks.

BLIP effectively utilizes the noisy web data by bootstrapping the captions, where a captioner generates synthetic captions and a filter removes the noisy ones.

We achieve state-of-the-art results on a wide range of vision-language tasks, such as image-text retrieval (+2.7% in average recall@1), image captioning (+2.8% in CIDEr), and VQA (+1.6% in VQA score). BLIP also demonstrates strong generalization ability when directly transferred to video language tasks in a zero-shot manner. Code, models, and datasets are released. (Ref 12.3)

Key Components

Multimodal Mixture of Encoder-Decoder (MED) :

The MED architecture is a versatile model capable of functioning as :

- A unimodal encoder, separately encoding images and text.
- An image-grounded text encoder, which integrates visual information with text.
- An image-grounded text decoder, which generates text based on visual input.

This flexibility allows BLIP to handle a wide range of tasks from image-text retrieval to image captioning and visual question answering (VQA).

Captioning and Filtering (CapFilt) :

CapFilt is a novel dataset bootstrapping method used to improve the quality of training data. It involves:

- A captioner, which generates synthetic captions for images.
- A filter, which removes noisy captions to refine the dataset.

Pre-training Objectives

BLIP is pre-trained using three objectives :

- **Image-Text Contrastive Loss (ITC) :** Aligns visual and textual representations by encouraging positive image-text pairs to have similar features.
- **Image-Text Matching Loss (ITM) :** Learns fine-grained alignment between images and text by predicting whether a given pair is matched.
- **Language Modeling Loss (LM) :** Optimizes the model to generate coherent textual descriptions from images.

Performance

BLIP has achieved state-of-the-art results in several vision-language benchmarks :

- **Image-Text Retrieval :** +2.7% improvement in average recall@1.
- **Image Captioning :** +2.8% improvement in CIDEr score.
- **Visual Question Answering :** +1.6% improvement in VQA score.

Additionally, BLIP demonstrates strong generalization capabilities, performing well on video-language tasks such as text-to-video retrieval and video question answering in a zero-shot setting.

Summary :

BLIP represents a significant advancement in vision-language pre-training by combining a flexible model architecture with a robust method for handling noisy data. Its ability to perform well across a variety of tasks makes it a valuable tool for developing applications that require integrated vision and language understanding and generation.

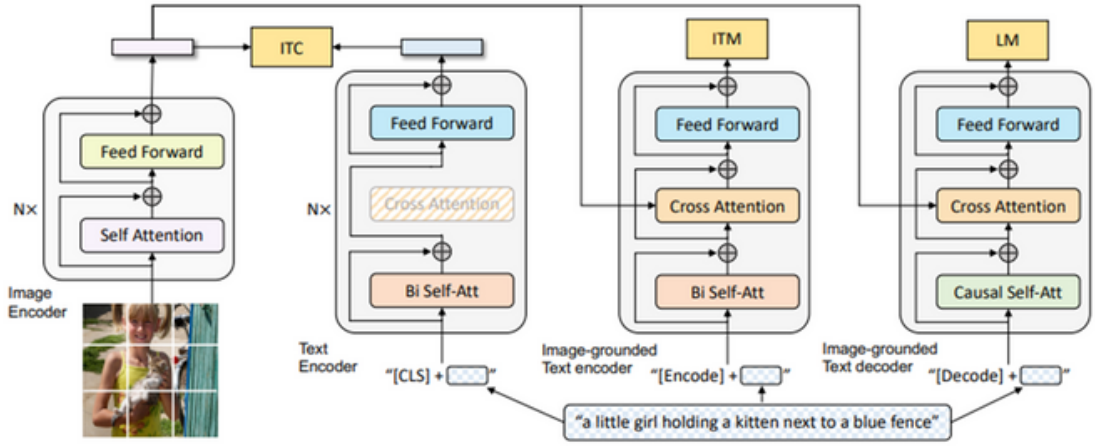


Figure 14. Pre-training model architecture and objectives of BLIP (same parameters have the same color)

We propose multimodal mixture of encoder-decoder, a unified vision-language model which can operate in one of the three functionalities:

(1) Unimodal encoder is trained with an image-text contrastive (ITC) loss to align the vision and language representations.

(2) Image-grounded text encoder uses additional cross-attention layers to model vision-language interactions, and is trained with a image-text matching (ITM) loss to distinguish between positive and negative image-text pairs.

(3) Image-grounded text decoder replaces the bi-directional self-attention layers with causal self-attention layers, and shares the same cross-attention layers and feed forward networks as the encoder. The decoder is trained with a language modeling (LM) loss to generate captions given images. (Ref 12.3)

5.3 Tools and Technologies for GUI

5.3.1. FIGMA

Why Use Figma for CAIVA's Design?

Design Consistency

- Ensures the CAIVA desktop application has a consistent and user-friendly interface.
- Maintains a unified design language across different parts of the application.

Real-Time Collaboration

- Enables designers and developers to work together seamlessly.
- Facilitates real-time feedback and adjustments.

Prototyping and Interactivity

- Allows for the creation of interactive prototypes.
- Helps in visualizing user interactions and flows before development.

Advantages of Translating Figma Designs to Python for CAIVA

Streamlined Design-to-Code Workflow

- Ensures the final application matches the original design.
- Reduces discrepancies between design and development.

Enhanced Development Efficiency

- Provides clear design specifications for developers.
- Speeds up the coding process by reducing the need for constant revisions.

High-Quality UI/UX

- Maintains the integrity of the design, ensuring a superior user experience.
- Allows developers to focus on implementing functionality.

Automated Code Generation

- Utilizes tools and plugins to convert Figma designs into Python code.
- Accelerates development and minimizes manual errors.

Design Workflow for CAIVA Using Figma and Python

1. Design Creation in Figma

- Create detailed UI designs and interactive prototypes for CAIVA.
- Use Figma's tools to define components, styles, and interactions.

2. Translating Designs to Python

- Use automated tools to convert Figma designs into Python code.
- Ensure design fidelity and consistency in the final application.

3. Development and Testing

- Implement functionality based on the translated designs.
- Conduct user testing to validate the design and user experience.

4. Iteration and Improvement

- Iterate on the design based on feedback and testing results.
- Continuously improve the application's interface and usability.

Using Figma to design the CAIVA desktop application and translating those designs into Python enhances consistency, efficiency, and quality. This approach ensures that CAIVA offers a seamless and engaging user experience, while streamlining the development process.

5.3.2. Tkinter

Advantages of Using Tkinter

- Rapid Prototyping
- Quickly create and test GUI designs.
- Immediate feedback and easy adjustments.
- Integration Flexibility.
- Easily integrate with various Python libraries and AI frameworks.
- Suitable for small to medium-scale applications.
- Cost-Effective.

What is Tkinter?

Tkinter is a standard GUI (Graphical User Interface) toolkit in Python. Provides a way to create desktop applications with windows, buttons, text fields, and other GUI elements. Tkinter is part of the standard Python library, requiring no additional installation or licensing costs.

Why Use Tkinter for CAIVA?

Tkinter is a standard GUI (Graphical User Interface) toolkit in Python. Provides a way to create desktop applications with windows, buttons, text fields, and other GUI elements. Tkinter is part of the standard Python library, requiring no additional installation or licensing costs.

- Simplicity and Ease of Use
- Easy to learn and implement for creating basic to moderately complex GUIs.
- Well-documented and supported in the Python community.
- Integration with Python.
- Seamlessly integrates with Python scripts and libraries, including AI models.
- Facilitates the direct implementation of AI functionalities within the GUI.
- Linking GUI Elements with AI Models to Answer User Questions.
- User Interaction.

Chapter 6

Implementation & Integrations

6.1. Data Preparation and Exploratory Analysis

This chapter details the steps taken to implement the data preparation, exploratory data analysis (EDA), and initial processing stages for our project. The process involves installing necessary libraries, loading datasets, and performing initial analyses to understand the data better.

1. Package Installation

To begin with, several Python packages are required. These packages facilitate data handling, visualization, text processing, and various other tasks necessary for the analysis.

The dataset used in this analysis is the GoEmotions dataset, which is a large-scale emotion classification dataset. It was loaded using the 'datasets' library from Hugging Face, and other datasets from Kaggle.

The following commands was used to install the required packages and loading the datasets :

```
!pip install datasets
!pip install wordcloud
!pip install nltk
!pip install palettable
!pip install clean-text
```

```
from datasets import load_dataset
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from wordcloud import WordCloud
import plotly.express as px
from collections import Counter
from nltk.stem import WordNetLemmatizer
from cleantext import clean
import string
import re
```

2. Load Dataset

Loading the 'go_emotions' dataset with the 'simplified' configuration from hugging face, and other datasets

```
go_emotions_dataset = load_dataset("go_emotions", "simplified")
```

```
emotion_classif = pd.read_csv("/content/Emotion_classify_Data.csv")
emotion_final = pd.read_csv("/content/Emotion_final.csv")
emotion_tweets = pd.read_csv("/content/tweet_emotions.csv")
emotion_merged = pd.read_csv("/content/merged.csv")
dataset_dir_ai = pd.read_pickle("merged_training.pkl")
```

3. Combine the selected emotions

1. joy
2. neutral
3. surprise
4. anger
5. sadness
6. fear.

We standardize column names across different DataFrames to ensure consistency and then filtering these DataFrames to retain only the rows that correspond to a selected list of emotions.

```
# Define a list of selected emotions that you want to filter
selected_emotions = ['surprise', 'anger', 'sadness', 'fear']
# Define a list of emotions we want it
go_emotions_dataframe_selected = ['joy','neutral','surprise', 'anger', 'sadness', 'fear']

# Renaming columns in various DataFrames to standardize column names
# for consistency across different datasets.
emotion_classif = emotion_classif.rename(columns={'Emotion': 'emotion', 'Comment':'text'})
emotion_final = emotion_final.rename(columns={'Emotion': 'emotion', 'Text':'text'})
emotion_tweets = emotion_tweets.rename(columns={'sentiment': 'emotion', 'content':'text'})
emotion_merged = emotion_merged.rename(columns={'label': 'emotion'})
go_emotions_dataframe = go_emotions_dataframe.rename(columns={'labels': 'emotion'})
dataset_dir_ai = dataset_dir_ai.rename(columns={'emotions': 'emotion'})

# Filtering each DataFrame based on the selected emotions list.
# This step is likely to retain only the rows corresponding to the selected emotions.
go_emotions_dataframe = Get_Emotions(go_emotions_dataframe, go_emotions_dataframe_selected)
emotion_classif = Get_Emotions(emotion_classif, selected_emotions)
emotion_final = Get_Emotions(emotion_final, selected_emotions)
emotion_tweets = Get_Emotions(emotion_tweets, selected_emotions)
emotion_merged = Get_Emotions(emotion_merged, selected_emotions)
dataset_dir_ai = Get_Emotions(dataset_dir_ai, selected_emotions)
```

4. EDA

EDA involves visualizing and summarizing the main characteristics of the dataset. The primary goal is to understand the structure and distribution of data, detect outliers, and identify relationships between variables.

- The “Analysis class” provides a way to visualize and summarize the distribution of categorical data in a DataFrame. It has the following main components:

1. Initialization (__init__ method)

- Takes an optional figsize parameter to set the size of the figures. Default size is (12, 6).

2. Method : plot_bar_pie_chart

- Creates side-by-side bar and pie charts to visualize the distribution of unique values in a specified column.
- Parameters : dataframe, column_name, explode, colors.
- Extracts unique labels and their counts from the column, then plots them.

3. Method : create_label_counts_df

- Creates a DataFrame showing the counts of each unique label in a specified column.
- Parameters : dataframe, column_name.
- Returns a transposed DataFrame with columns 'emotion' and 'count'.

4. Call Method (__call__)

- Allows the instance to be used as a function.
- Parameters : dataframe, column_name, explode, colors.
- Displays the first 10 rows of the DataFrame.
- Displays the transposed DataFrame of label counts.
- Calls plot_bar_pie_chart to display the bar and pie charts.

```
[ ] class Analysis():
        def __init__(self, figsize=(12, 6)):
            self.figsize = figsize

        # Create pie chart
        def plot_bar_pie_chart(self, dataframe, column_name, explode, colors):
            fig, (ax1, ax2) = plt.subplots(1, 2, figsize=self.figsize)
            labels = list(dataframe[column_name].unique())
            sizes = list(dataframe[column_name].value_counts())
            # Bar chart
            ax1.bar(labels, sizes, color=colors)
            ax1.set_title('Distribution of Emotions')
            ax1.set_xlabel('Emotion')
            ax1.set_ylabel('Count')
            ax1.tick_params(axis='x', rotation=45)

            # Pie chart
            ax2.pie(sizes, labels=labels,explode=explode, colors=colors, autopct='%1.1f%%', startangle=90)
            ax2.set_title('Distribution of Emotions')

            # Show plot
            plt.tight_layout()
            plt.show()

        def create_label_counts_df(self, dataframe, column_name):
            """
            Creates a DataFrame displaying the counts of each unique label in a specified column of a DataFrame.

            Parameters:
                dataframe (pd.DataFrame): The DataFrame containing the data.
                column_name (str): The name of the column containing the labels.

            Returns:
                pd.DataFrame: A DataFrame displaying the counts of each unique label.
            """
            label_counts_df = dataframe[column_name].value_counts().reset_index()
            label_counts_df.columns = ['emotion', 'count']
            return label_counts_df.transpose()  # Transpose the DataFrame

        def __call__ (self, dataframe, column_name, explode, colors):
            display(dataframe.head(10))
            print("-"*55)
            display(self.create_label_counts_df(dataframe, column_name))
            self.plot_bar_pie_chart(dataframe, column_name, explode, colors)

    analysis = Analysis(figsize=(12, 6))
```

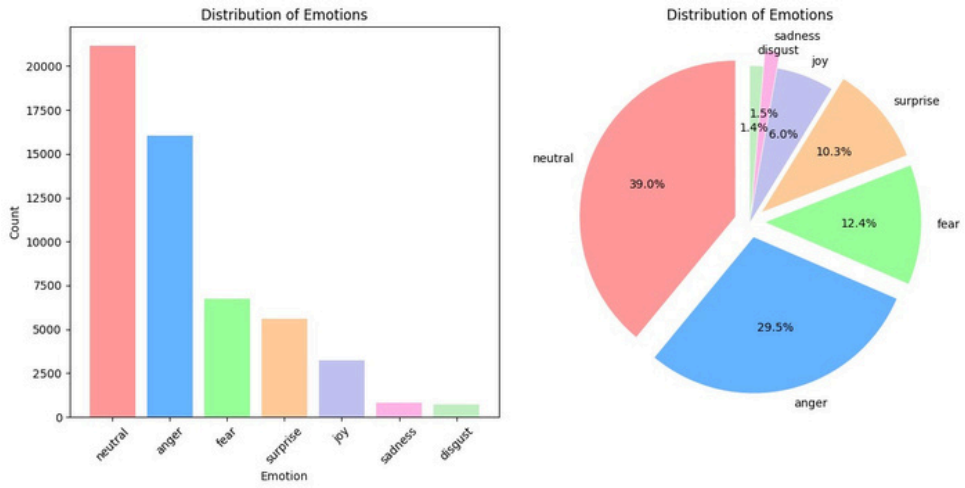


Figure 15. Distribution of Emotions

- To perform a detailed analysis of text entries, we can calculate several metrics for each entry, such as word count, character count, average word length, stopwords count, hashtags count, numeric count, and uppercase words count. Here's a step-by-step description of how to achieve this, followed by creating visualizations for these metrics.

Metrics Calculation for Each Text Entry

1. Word Count
- Count the number of words in each text entry.
- This can be done by splitting the text by spaces and counting the resulting list's length.

2. Character Count
- Count the total number of characters in each text entry.
- This includes spaces and punctuation.

3. Average Word Length
- Calculate the average length of the words in each text entry.
- This can be done by dividing the total character count (excluding spaces) by the word count.

4. Stopwords Count
- Count the number of stopwords in each text entry.
- Stopwords are common words like "the", "and", "is", which are often filtered out in text processing.
- Use a predefined list of stopwords to identify them.

5. Hashtags Count
- Count the number of hashtags (words starting with '#') in each text entry.

6. Numeric Count
- Count the number of numeric characters or words in each text entry.
- This includes both individual digits and numeric words.

7. Uppercase Words Count
- Count the number of words that are entirely in uppercase letters in each text entry.

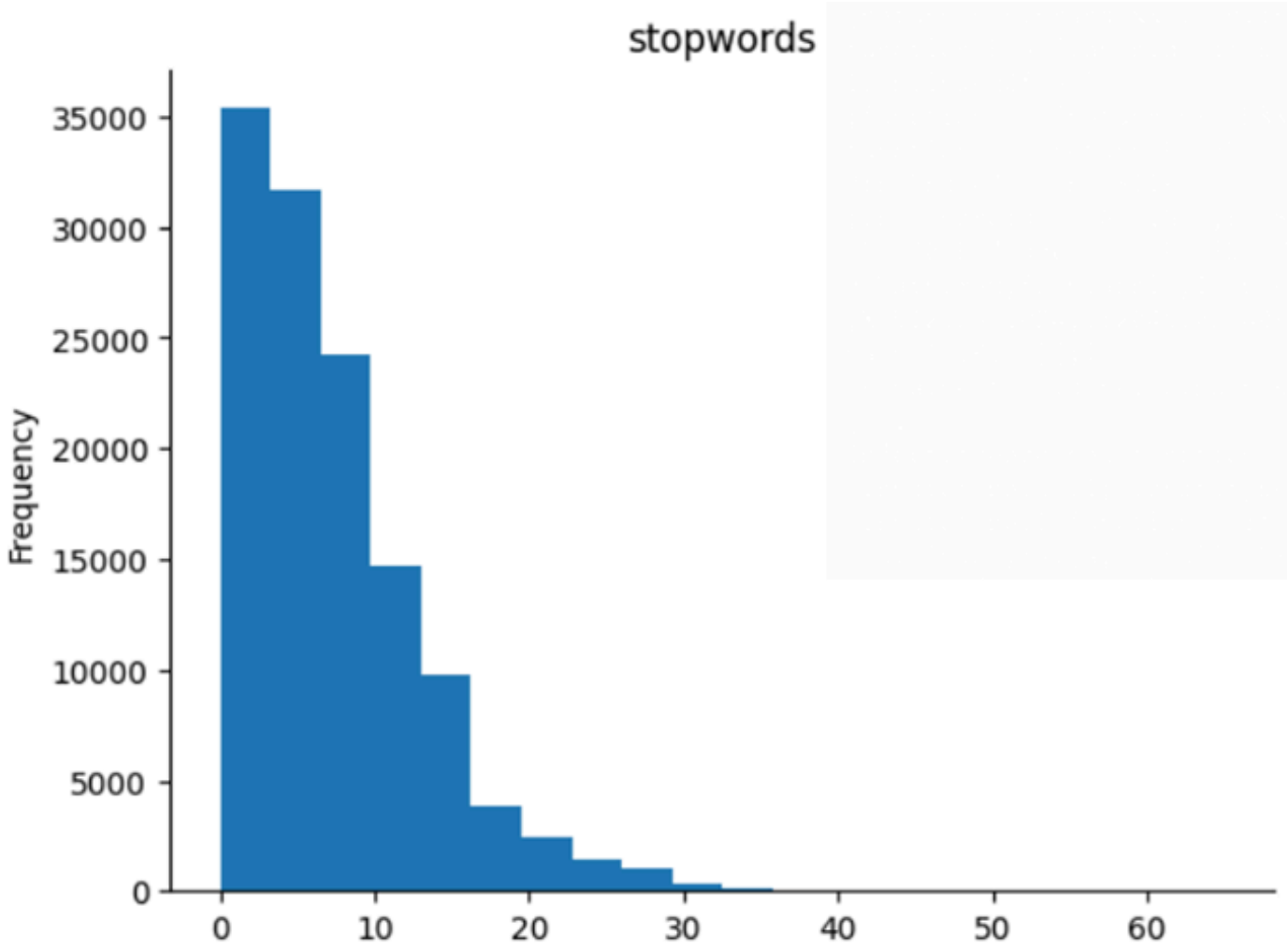


Figure 16. Stop Words frequency

5. Remove Outliers

- This line of code removes multiple columns from the Full_Data DataFrame. Which is considered insignificant and negatively affects the quality of our data

```
[ ] full_Data.drop(['tweet_id','word_count','char_count','avg_word','stopwords','hastags','numerics','upper'], axis=1, inplace=True)
    full_Data
```

- The provided code defines a class preprocessing_data that handles various text preprocessing steps. This class is then instantiated and used to preprocess a DataFrame named full_Data.

Processing Steps :

1.remove_emoji : Removes emojis from text using the clean function with the no_emoji=True parameter.

2.texts_to_lowercase: Converts all text to lowercase.

3.remove_stop_words : Removes stopwords from the text, Tokenizes the text, filters out stopwords, and rejoins the remaining words.

4.remove_punctuations: Removes punctuation from the text using str.maketrans to create a translation table.

5.remove_special_characters_links: Removes URLs from the text using a regular expression.

6.remove_single_characters: Removes single characters from the text using regular expressions.

7.remove_numbers: Removes numbers from the text using a regular expression.

8.remove_repeated_letters: Removes sequences of repeated letters (e.g., "cooool" becomes "cool") using a regular expression.

```
class preprocessing_data():
    def __init__(self, stopwords_list):
      self.stopwords_list = stopwords_list

    def remove_emoji(self, data):
        return [clean(sentence, no_emoji=True) for sentence in data]

    def texts_to_lowercase(self, col_list):
        return [x.lower() for x in col_list]

    def remove_stop_words(self, data, stopwords_list):
        # Define a list of stopwords
        li =[]
        for text in data:
            # Tokenize the text into words
            words = text.split()

            # Filter out the stopwords
            filtered_words = [word for word in words if word.lower() not in stopwords_list]

            # Join the filtered words back into a single string
            filtered_text = ' '.join(filtered_words)
            li.append(filtered_text)
        return li

    def remove_punctuations(self, text):
        translation_table = str.maketrans("", "", string.punctuation)
        return [sentence.translate(translation_table) for sentence in text]

    def remove_special_characters_links(self, data):
        return [re.sub(r'http\S+|www.\S+', '', sentence).strip() for sentence in data]

    def remove_single_characters(self, data):
        li = [re.sub(r'\b\w\b', '', text) for text in data]
        li = [re.sub(' +', ' ', text).strip() for text in li]
        return li

    def remove_numbers(self, data):
        return [re.sub(r'\d+', '', sentence).strip() for sentence in data]

    def remove_repeated_letters(self, data):
        return [re.sub(r'([a-zA-Z])\1{2,}', r'\1', sentence).strip() for sentence in data]

    def __call__(self, input_data, type_processing='lstm'):
            x1 = self.remove_emoji(input_data['text'])
            x2 = self.texts_to_lowercase(x1)
            if type_processing == 'lstm':
                x3 = self.remove_stop_words(x2, stopwords_list)
                x4 = self.remove_punctuations(x3)
                x5 = self.remove_special_characters_links(x4)
                x6 = self.remove_single_characters(x5)
                x7 = self.remove_numbers(x6)
                input_data['text'] = self.remove_repeated_letters(x7)
                return  input_data

            elif type_processing == 'roberta':
                x3 = self.remove_special_characters_links(x2)
                x4 = self.remove_single_characters(x3)
                x5 = self.remove_numbers(x4)
                input_data['text']  = self.remove_repeated_letters(x5)
                return input_data

preprocessor = preprocessing_data(stopwords_list)
final_dataframe_after_preprocessing = preprocessor(full_Data, type_processing='lstm')
```

Drop Rare Words

This process helps in cleaning the text data by removing words that are unlikely to contribute significantly to the analysis or modeling, potentially improving the performance of subsequent NLP tasks.

1.Counting Word Frequencies :

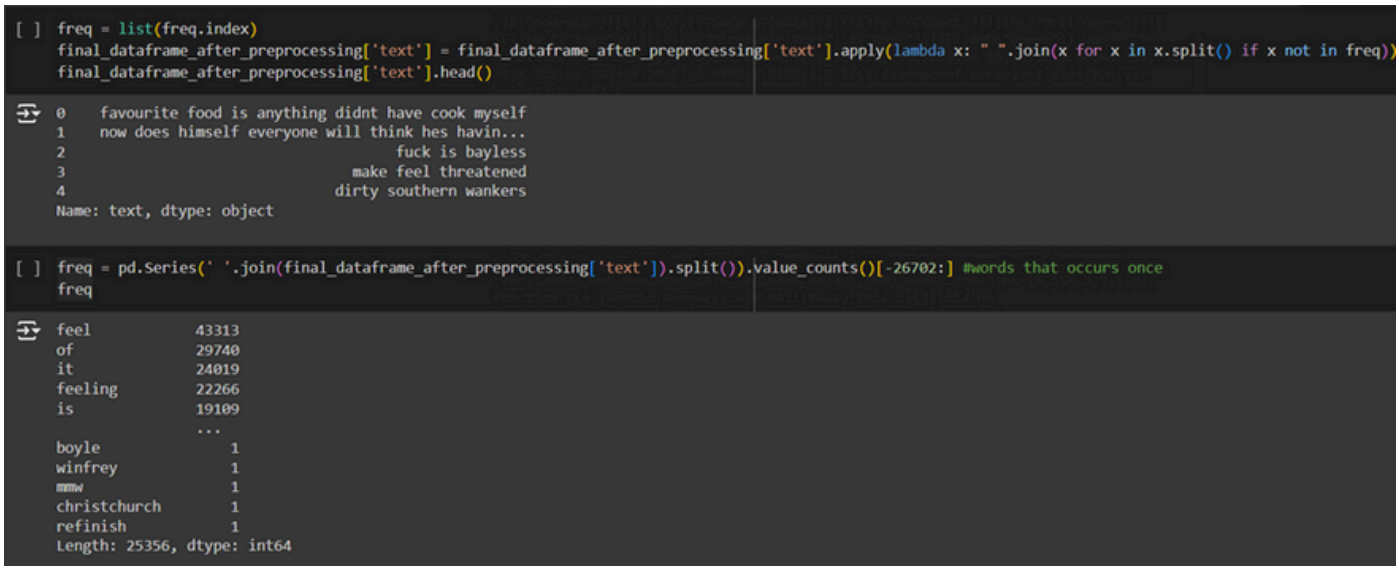
- Combine all text entries and count the occurrence of each word.
- Identify words that occur only once.

2.Removing Rare Words :

- Convert the list of rare words into a list.
- Remove these rare words from each text entry.

3.Reviewing Cleaned Text :

Display the first few rows of the cleaned text data to verify the results.



```
[ ] freq = list(freq.index)
    final_dataframe_after_preprocessing['text'] = final_dataframe_after_preprocessing['text'].apply(lambda x: " ".join(x for x in x.split() if x not in freq))
    final_dataframe_after_preprocessing['text'].head()

0    favourite food is anything didnt have cook myself
1    now does himself everyone will think hes havin...
2                                    fuck is bayless
3                               make feel threatened
4                             dirty southern wankers
Name: text, dtype: object

[ ] freq = pd.Series(' '.join(final_dataframe_after_preprocessing['text']).split()).value_counts()[-26702:] #words that occurs once
    freq

feel           43313
of             29740
it             24019
feeling        22266
is             19109
               ...
boyle              1
winfrey            1
mmw                1
christchurch       1
refinish           1
Length: 25356, dtype: int64
```

6. Data Visualization

- **Word Cloud**

This diagram is a word cloud, a visual representation of text data where the size of each word indicates its frequency or importance within a given body of text.

The word cloud visually prioritizes the importance of different words based on their frequency in the text data. In this case, words related to personal experience and expressions of feelings and thoughts are predominant, suggesting the text might be from a source such as personal narratives, social media posts, or other text where individuals express their thoughts and emotions.

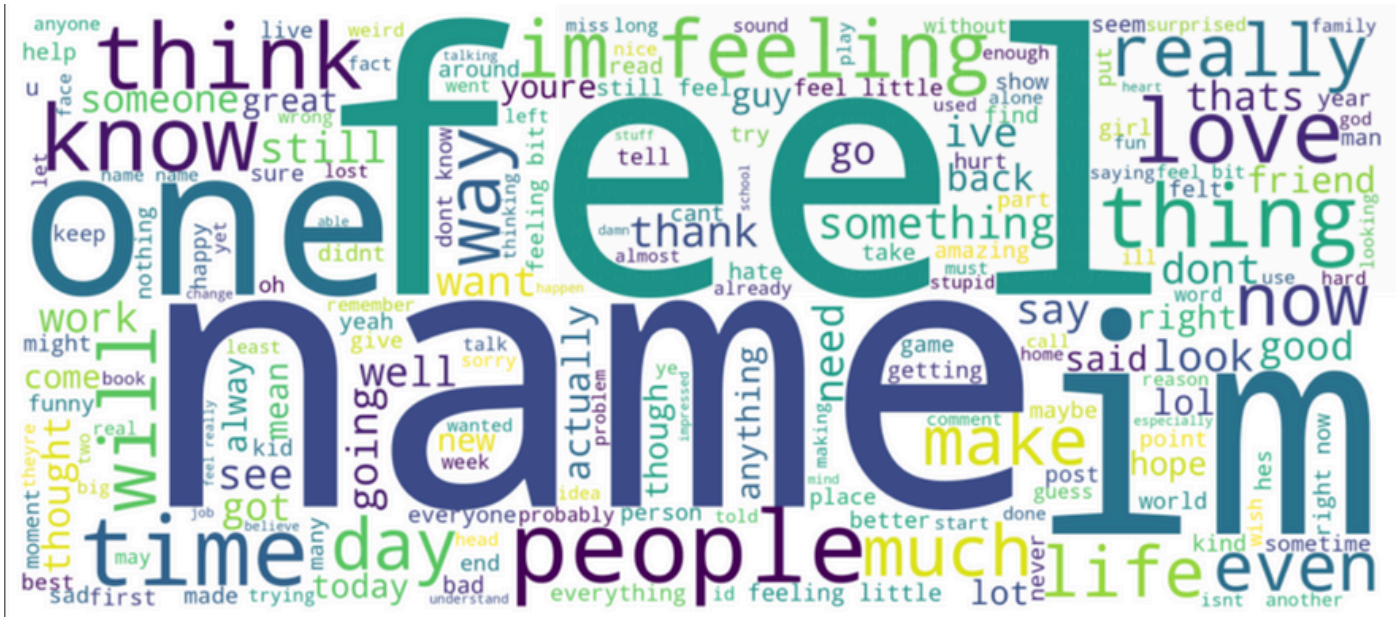


Figure 17. word cloud

- **DonNut Plot**

This figure with two subplots :

1. Table Explanation: The table on the left side lists the top 20 most common neutral words and their frequency counts in a given text dataset.
The table provides a detailed numerical count of each word's occurrence.

2. Donut Plot Explanation: The donut plot on the right visually represents the data from the table. Each segment of the donut plot corresponds to a word from the table, and the size of each segment reflects the frequency of that word.
This visualization allows easy comparison of the frequency of the most common words in the neutral sentiment text.

The donut plot offers a visual summary, making it easy to see the relative frequencies of these common neutral words at a glance.

This combination of textual and visual data representation helps in understanding which neutral words are most prevalent in the dataset.

Common_Words	Count
name	3717
is	2978
of	2667
it	2546
on	1455
not	1436
was	1342
be	1320
have	1240
are	1192
but	1126
like	1124
its	1062
what	826
dont	795
no	773
get	699
can	667
do	656
people	647

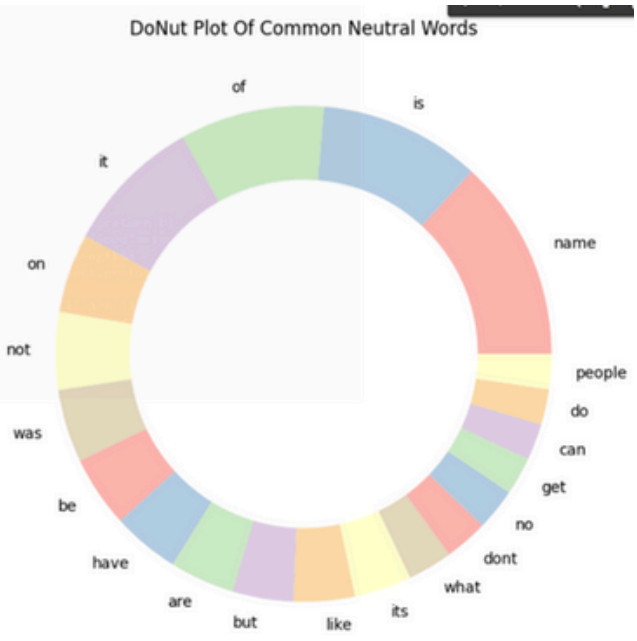


Figure 18. DonNut Plot

Bar Charts

This diagram shows a series of bar charts representing the frequency of common words across different categories of emotions. Each chart highlights the most frequent words in texts associated with different emotional contexts in all of data and in each emotion.

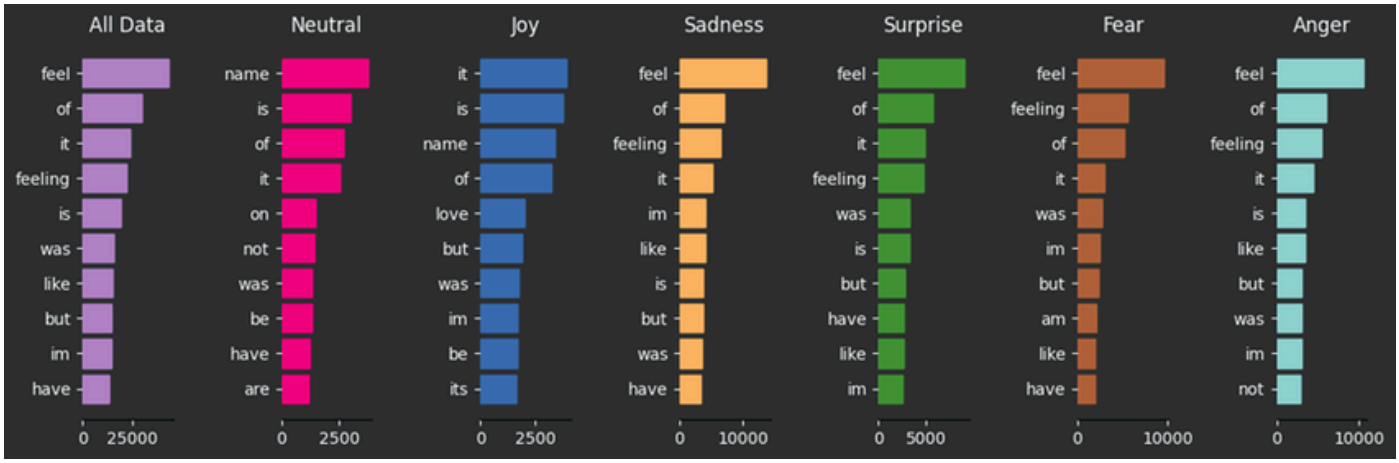


Figure 19. Bar Charts

6.2. Model(1) parallel CNN-LSTM Model :

The purpose of the Cnn_Lstm class and its associated methods is to build a hybrid deep learning model that leverages the strengths of both Convolutional Neural Networks (CNNs) and Long Short-Term Memory networks (LSTMs) for text classification tasks. Here's a breakdown of the key points :

1. Objectives

1. **Text Feature Extraction :**
   - CNNs are excellent at capturing local patterns and features in text data. They can identify n-grams or phrases that are significant within the text.
   - LSTMs are capable of capturing long-term dependencies and contextual information in sequences, which is crucial for understanding the overall meaning of text.
2. **Combining Strengths :**
   - By combining CNN and LSTM layers, the model aims to take advantage of the local feature extraction capabilities of CNNs and the sequence modeling capabilities of LSTMs. This hybrid approach can provide a more comprehensive understanding of the text.

2. Detailed Explanation

1. **Embedding Layer :**
   - The model starts with an embedding layer that converts words into dense vectors of fixed size, which captures semantic relationships between words. This layer uses pre-trained embeddings to provide a meaningful representation of words based on their usage in large corpora.
2. **CNN Block :**
   - **Convolutional Layers :** These layers apply convolution operations to the input embeddings to detect local patterns.
   - **Max Pooling Layer :** This layer reduces the dimensionality of the feature maps, retaining the most important features.
   - **Flatten Layer :** Converts the 2D feature maps into 1D vectors.
   - **Dense Layer :** Adds a fully connected layer to learn higher-level features from the flattened vectors.
3. **LSTM Block :**
   - **Bidirectional LSTM Layers :** These layers process the sequence in both forward and backward directions, capturing information from the entire sequence.
   - **Attention Mechanism :** This mechanism helps the model focus on important parts of the sequence, enhancing its ability to understand context and dependencies.
   - **Dense Layer :** Similar to the CNN block, this layer learns higher-level features from the context vector produced by the attention mechanism.

4. Merging Layers :

- The outputs from the CNN and LSTM blocks are concatenated to form a combined feature representation, leveraging both local and sequential information.

5. Fully Connected Layers :

- Additional dense layers are used to further process the combined features, often with dropout for regularization.
- **Output Layer :** This layer uses softmax activation to produce the final classification probabilities.

```
class Cnn_Lstm():
  def __init__(self, num_words, embedding_dim,
               embedding_matrix, filters,
               kernel_size, max_length=200):
      """
      Initialize the CNN-LSTM hybrid model.

      Args:
          - num_words: Number of words in the vocabulary.
          - embedding_dim: Dimensionality of the word embeddings.
          - embedding_matrix: Pre-trained embedding matrix.
          - filters: Number of filters in the convolutional layers.
          - kernel_size: Size of the convolutional kernels.
          - max_length: Maximum sequence length (default is 200).
      """
      self. num_words = num_words
      self. embedding_dim = embedding_dim
      self. embedding_matrix = embedding_matrix
      self. filters = filters
      self. kernel_size = kernel_size
      self.max_length = max_length
```

```
def cnn_block(self, input_layer, filters, kernel_size):
    """
    Define the CNN block of the model.

    Args:
      - input_layer: Input layer.
      - filters: Number of filters in the convolutional layers.
      - kernel_size: Size of the convolutional kernels.

    Returns:
      - Output layer of the CNN block.
    """
    x = Conv1D(filters=filters, kernel_size=kernel_size, activation='relu',
               kernel_constraint= MaxNorm( max_value=3, axis=[0,1]),
               kernel_initializer=kernel_initializer,
               kernel_regularizer='l2')(input_layer)
    x = Conv1D(filters=filters, kernel_size=kernel_size,
               activation='relu',kernel_initializer=kernel_initializer,
               kernel_constraint= MaxNorm( max_value=3, axis=[0,1]),)(x)

    x = MaxPool1D(pool_size=2, strides=2)(x)

    x = Flatten()(x)
    out = Dense(128, activation='relu', kernel_initializer=kernel_initializer)(x)
    return out
```

```python
def lstm_block(self, input_layer):
    """
    Define the LSTM block of the model.

    Args:
      - input_layer: Input layer.

      Returns:
      - Output layer of the LSTM block.
    """
    lstm = Bidirectional(LSTM(128,return_sequences = True, kernel_initializer=kernel_initializer))
    (input_layer)
    (lstm, forward_h, forward_c, backward_h, backward_c) = Bidirectional(LSTM(128,
                                                                              return_sequences=True,
                                                                              return_state=True,
                                                                    kernel_initializer=kernel_initializer),
                                                                              )(lstm)
    state_h = Concatenate()([forward_h, backward_h])
    state_c = Concatenate()([forward_c, backward_c])
    context_vector, attention_weights = Attention(128)(lstm, state_h)
    out = Dense(128, activation='relu', kernel_initializer=kernel_initializer)(context_vector)
    return out
```

```python
def __call__(self, target):
    """
    Build the CNN-LSTM hybrid model.

    Args:
    - target: Number of classes for classification.

    Returns:
    - Built model.
    """
    # Define input layer
    input = Input(shape=(200,))
    # Embedding layer
    embeddding = Embedding(self.num_words,
                           self.embedding_dim,
                           weights=[self.embedding_matrix],
                           trainable=False
                           )(input) # Apply CNN block
    out1 = self.cnn_block(embeddding, self.filters, self.kernel_size)
    # Apply LSTM block
    out2 = self.lstm_block( embeddding)
    # Merge outputs from CNN and LSTM blocks
    merged = concatenate([out1, out2])
    # Dense layer
    x = Dense(128, activation='relu', kernel_initializer=kernel_initializer)(merged)
    x = Dropout(0.15)(x)
        # Output layer
    outputs = Dense(target, activation='softmax', kernel_initializer=kernel_initializer)(x)
    d = Model(inputs=input, outputs=outputs) # Create the model
    return Model(inputs=input, outputs=outputs)

cnn_lstm = Cnn_Lstm(num_words, embedding_dim,
                    embedding_matrix, filters,
                    kernel_size, max_length=200)
```

3. Creating the Model :

cnn_lstm_model = cnn_lstm(target=6): This line instantiates the model with 6 output classes, setting up the architecture as defined in the Cnn_Lstm class

4. Compiling the Model :

- **loss="categorical_crossentropy" :** This loss function is suitable for multi-class classification problems.
- **optimizer='adam' :** The Adam optimizer is used for its adaptive learning rate and efficient training process.
- **metrics=["accuracy", Recall(), Precision()] :** These metrics are used to monitor the performance of the model during training and evaluation.

```
# Create the CNN-LSTM model
cnn_lstm_model = cnn_lstm(target=6)

# Compile the model
cnn_lstm_model.compile(
    loss="categorical_crossentropy",  # Loss function
    optimizer='adam',  # Optimizer
    metrics=["accuracy", Recall(), Precision()]  # Evaluation metrics
)

# Display model summary
cnn_lstm_model.summary()
```

5. Model Initialization :

The CNN-LSTM model is created using the specified architecture, which combines Convolutional Neural Networks (CNNs) for feature extraction and Long Short-Term Memory networks (LSTMs) for capturing sequential dependencies.

6. Model Compilation :

The model is compiled with a categorical cross-entropy loss function, the Adam optimizer, and evaluation metrics including accuracy, recall, and precision.

7. Training Process :

The model is compiled with a categorical cross-entropy loss function, the Adam optimizer, and evaluation metrics including accuracy, recall, and precision.

- **Training Data :** The model is trained using x_train as the input data and y_train as the labels.
- **Batch Size :** The training process uses a batch size of 64, meaning the model processes 64 samples at a time before updating the weights.
- **Epochs :** The model is trained for 1000 epochs, which represents 1000 iterations over the entire training dataset.
- **Validation Data :** x_validation and y_validation are used for validating the model's performance after each epoch.
- **Callbacks :** Various callbacks are used to monitor the training process :
  - **early_stopping :** Stops training if the model's performance on the validation data stops improving.
  - **checkpoint :** Saves the model's weights at the best epoch.
  - **myCallback() :** Custom callback for additional monitoring or operations during training.

8. Training Execution :

The fit method executes the training process, which adjusts the model's parameters to minimize the loss function while improving the evaluation metrics.

```
# Train the CNN-LSTM model
history_lstm_cnn = cnn_lstm_model.fit(
    x_train,  # Training data
    y_train,  # Training labels
    batch_size=64,  # Batch size
    epochs=1000,  # Number of epochs
    validation_data=(x_validation, y_validation),  # Validation data
    callbacks=[early_stopping, checkpoint, myCallback()]  # Callbacks for monitoring and early stopping
)
```

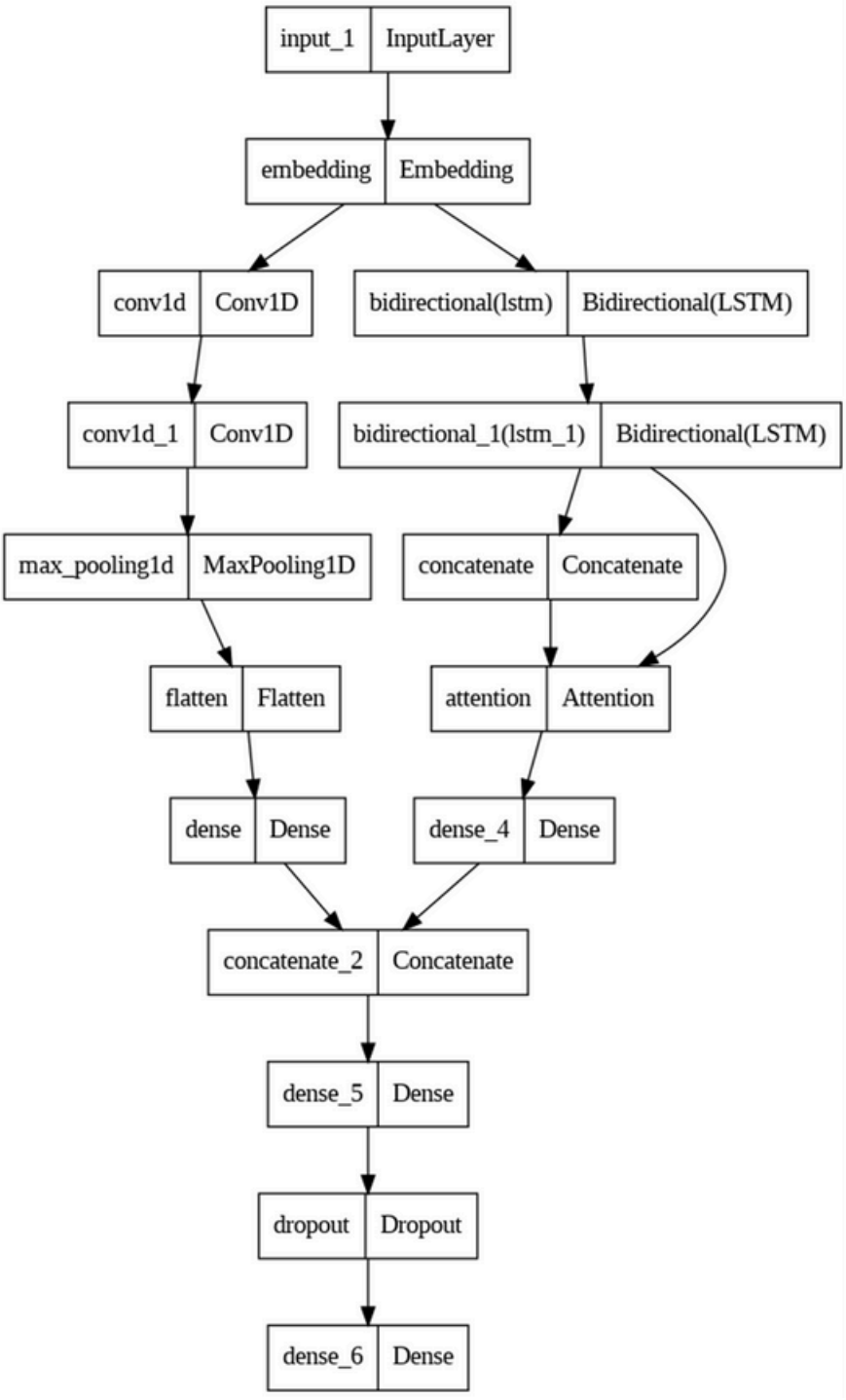


Figure 20. Model(1) parallel CNN-LSTM Model

6.3. Model(2) Sequential CNN-LSTM Model :

(Ref 5.2)

The purpose of the Cnn_Lstm class and its associated methods is to build a hybrid deep learning model that leverages the strengths of both Convolutional Neural Networks (CNNs) and Long Short-Term Memory networks (LSTMs) for text classification tasks. Here's a breakdown of the key points :

Advantages :

- **Improved Performance :** Combining CNNs and LSTMs can often lead to better performance compared to using either architecture alone, especially in tasks where both local and sequential information are important.
- **Feature Hierarchy :** CNN layers capture low-level features like n-grams, which are then processed by LSTM layers to capture higher-level abstractions and context.
- **Flexibility :** The model can be adapted and fine-tuned for various text classification tasks by adjusting hyperparameters, adding regularization techniques, or incorporating different pre-trained embeddings.

1. Initialization (__init__ method)

Purpose : This defines the class and initializes its attributes.

Attributes :

- **num_words :** Number of words in the vocabulary.
- **embedding_dim :** Dimensionality of the word embeddings.
- **embedding_matrix :** Pre-trained word embeddings matrix.

```
class Sequential_Cnn_Lstm():
    def __init__(self, num_words, embedding_dim, embedding_matrix):

        self.num_words = num_words
        self.embedding_dim = embedding_dim
        self.embedding_matrix = embedding_matrix
```

2. Building the Model (__call__ method)

- **Purpose :**

This method constructs the actual model architecture using the Keras functional API.

- **Parameters :**
  - **target :** Number of classes for the classification task.

- **Components :**
  - The __call__ method (which can be invoked like a function) constructs the neural network model.
  - It starts by defining an input layer with a shape of (200).
  - An embedding layer (Embedding) is added, using the provided embedding_matrix and making it non-trainable.
  - Next, there's a 1D convolutional layer (Conv1D) with specified filters and kernel size, followed by activation functions.
  - The architecture also includes bidirectional LSTM layers and dense layers.
  - The final output layer uses softmax activation for classification.
  - **Input Layer :** Defines the shape of the input data, which is a sequence of 200 tokens (words).
  - **Embedding Layer :** Converts input tokens into dense vectors (embedding_dim dimensions) using pre-trained embeddings (embedding_matrix). The embeddings are not trainable (trainable=False).

```
def __call__(self, target):
  input = Input(shape=(200,))
  embeddding = Embedding(self.num_words,
                         self.embedding_dim,
                         weights=[self.embedding_matrix],
                         trainable=False)(input)
  x = Conv1D(filters=filters, kernel_size=kernel_size, activation='relu',
             kernel_constraint= MaxNorm( max_value=3, axis=[0,1]),
             kernel_initializer=kernel_initializer)(embeddding)

  x = Conv1D(filters=filters, kernel_size=kernel_size,
             activation='relu', kernel_initializer=kernel_initializer,
             kernel_constraint= MaxNorm( max_value=3, axis=[0,1]),)(x)

  x = MaxPool1D(pool_size=2, strides=2)(x)

  lstm = Bidirectional(LSTM(256,return_sequences = True, kernel_initializer=kernel_initializer))(x)
  lstm = Bidirectional(LSTM(256,return_sequences = True, kernel_initializer=kernel_initializer))(x)
  lstm = Bidirectional(LSTM(256,return_sequences = True, kernel_initializer=kernel_initializer))(lstm)
  (lstm, forward_h, forward_c, backward_h, backward_c) = Bidirectional(LSTM(256,
                                                          return_sequences=True,
                                                          return_state=True,
                                                          kernel_initializer=kernel_initializer),
                                                         )(lstm)
  state_h = Concatenate()([forward_h, backward_h])
  state_c = Concatenate()([forward_c, backward_c])
  context_vector, attention_weights = Attention(128)(lstm, state_h)
  x = Dense(512, activation='relu', kernel_initializer=kernel_initializer)(context_vector)
  x = BatchNormalization()(x)
  x = Dropout(0.25)(x)
  x = Dense(128, activation='relu', kernel_initializer=kernel_initializer)(x)
  x = BatchNormalization()(x)
  x = Dropout(0.15)(x)
  outputs = Dense(target, activation='softmax', kernel_initializer=kernel_initializer)(x)
  return Model(inputs=input, outputs=outputs)

sequential_cnn_lstm = Sequential_Cnn_Lstm(num_words, embedding_dim, embedding_matrix)
```

3. Model Instantiation

Function Call : sequential_cnn_lstm(target=6)

- **Purpose :** This line of code calls a function named sequential_cnn_lstm, which likely returns an instance of a Sequential CNN-LSTM model.
- **Argument :**
  - target=6 : Indicates that the model is being built for a classification task with six target classes.
- **Output :** The returned model instance is stored in the variable sequential_cnn_lstm_model.

4. Model Compilation

Method Call : compile()

- **Purpose :** Prepares the model for training by configuring the loss function, optimizer, and evaluation metrics.
- **Arguments :**
  - loss="categorical_crossentropy":
    - Specifies the loss function to be used. categorical_crossentropy is appropriate for multi-class classification tasks.
  - optimizer="adam":
    - Specifies the optimizer to be used. adam is an adaptive learning rate optimization algorithm.
  - metrics=["accuracy", Recall(), Precision()]:
    - Specifies the list of metrics to be evaluated during training and testing.
    - accuracy : Measures the proportion of correctly classified instances.
    - Recall() : Measures the ability of the model to find all relevant instances.
    - Precision() : Measures the ability of the model to return only relevant instances.

5. Model Summary

Method Call : summary()

- **Purpose :** Prints a summary of the model architecture.
- **Output :** The summary includes information about each layer in the model, such as layer type, output shape, and the number of parameters.

```
sequential_cnn_lstm_model = sequential_cnn_lstm(target=6)
sequential_cnn_lstm_model.compile(loss="categorical_crossentropy",
                                  optimizer="adam",
                                  metrics=["accuracy", Recall(), Precision()])
sequential_cnn_lstm_model.summary()
```

6. Training the Sequential CNN-LSTM Model

Method Call : fit()

- **Purpose :** Trains the model for a fixed number of epochs on the provided dataset.
- **Arguments :**
  - x_train: Training data features.
  - y_train: Training data labels.
  - batch_size=64: Number of samples per gradient update. Here, it is set to 64.
  - epochs=1000: Number of epochs to train the model. Here, it is set to 1000 epochs.
  - validation_data=(x_validation, y_validation):
    - Tuple containing validation data and labels. This is used to evaluate the model at the end of each epoch.
  - callbacks=[early_stopping, checkpoint, myCallback()]:
    - List of callback instances to apply during training.

```
history_sequential_cnn_lstm= sequential_cnn_lstm_model.fit(
        x_train, y_train,
        batch_size=64,
        epochs=1000,
        validation_data=(x_validation, y_validation),
callbacks=[early_stopping, checkpoint, myCallback()])
```

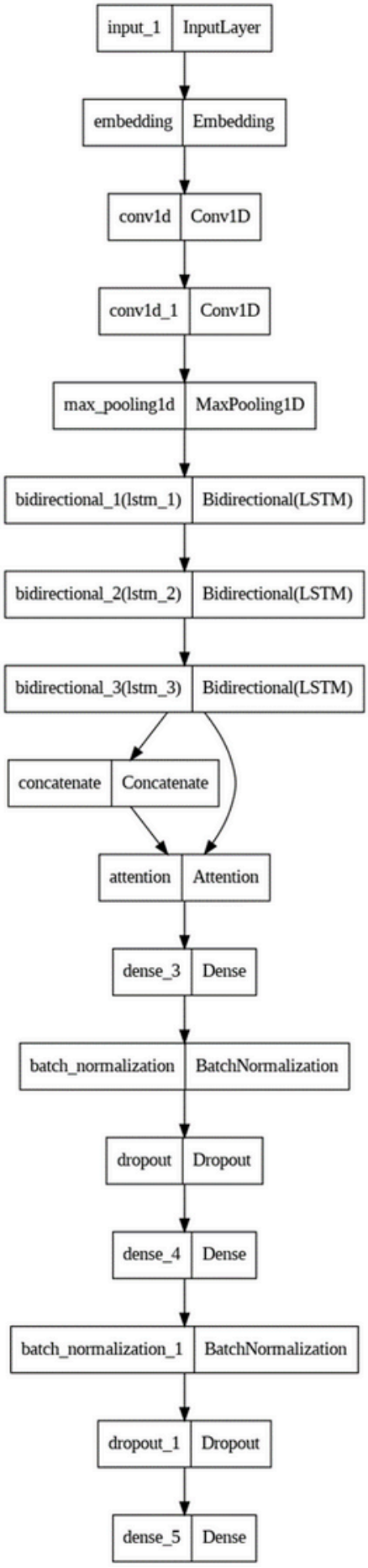


Figure 21. Model(2) Sequential CNN-LSTM Model

6.4. Model(3): roberta model

several advantages for sentiment analysis tasks :

1. Contextual Understanding :
RoBERTa excels in capturing contextual information from text due to its architecture, which includes self-attention mechanisms. This allows it to understand the relationship between words in a sentence and how they contribute to the overall sentiment expressed.

2. Pre-training on Large Corpus :
RoBERTa is pre-trained on a large corpus of text data, which includes a wide variety of domains and writing styles. This pre-training helps the model learn general language representations that can be fine-tuned for specific tasks like sentiment analysis. The extensive pre-training contributes to its ability to understand nuanced sentiments.

3. Transfer Learning :
Due to its pre-training, RoBERTa can leverage transfer learning effectively. This means that even with relatively small amounts of task-specific training data, RoBERTa can achieve good performance in sentiment analysis tasks. This is especially beneficial in scenarios where labeled data is scarce or expensive to obtain.

4. Multi-task Learning Capability :
RoBERTa's architecture supports multi-task learning, meaning it can be fine-tuned for various NLP tasks including sentiment analysis. This capability allows researchers and developers to train a single model that can perform well across different tasks, reducing the need for task-specific architectures.

5. State-of-the-Art Performance :
RoBERTa belongs to the family of Transformer models, known for achieving state-of-the-art performance on various NLP benchmarks. For sentiment analysis, RoBERTa's performance often surpasses traditional machine learning methods and earlier NLP architectures due to its ability to capture complex patterns and dependencies in text.

6. Efficient Computation :
Despite its complexity, RoBERTa and similar Transformer models can be efficiently trained and utilized for inference on modern computational hardware, including GPUs and TPUs. This makes it feasible to deploy RoBERTa-based sentiment analysis models in real-world applications.

7. Interpretability :
RoBERTa's attention mechanisms allow for some level of interpretability in understanding which parts of the text contribute most to the sentiment prediction. This can be valuable for debugging and understanding model predictions in sentiment analysis tasks.

1. Model and Tokenizer Setup

- **Model Initialization :**
- **Tokenizer Initialization :**

```
model_name = "roberta-base"
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels = 6).to(device)
tokenizer = RobertaTokenizer.from_pretrained(model_name)
```

2. Initialization and Configuration

- **Batch Size Definition :**
  - Purpose: Specifies the number of samples (instances) processed by the model in each iteration (batch) during training and evaluation.
  - Value: 16 indicates that each batch will contain 16 samples.
-
- **Model Name :**
  - Purpose: Defines a name for the output directory where the fine-tuned model and related files will be saved.
  - Value: "finetuned-emotion" is a placeholder indicating the desired name for the model directory.
- **Training Arguments Setup :**
  - Purpose: Configures various parameters required for training the model using the Trainer class from Hugging Face's Transformers library.
  - Arguments :
    - output_dir: Specifies the directory where model checkpoints and evaluation results will be saved.
    - num_train_epochs: Number of times the entire training dataset is passed through the model during training. Here, it is set to 5 epochs.
    - learning_rate: Controls the step size during optimization. Here, it is set to 2e-5 (0.00002).
    - per_device_train_batch_size: Batch size per GPU/TPU core for training. It is set to 16 based on batch_size variable.
    - per_device_eval_batch_size: Batch size per GPU/TPU core for evaluation. It is also set to 16.
    - weight_decay: Regularization parameter to prevent overfitting. It is set to 0.01.
    - evaluation_strategy: Specifies how often evaluation should be performed during training. 'epoch' means evaluation is performed after each epoch.
    - disable_tqdm: Controls whether to disable the tqdm progress bar during training. It is set to False to enable the progress bar.

- **Trainer Initialization :**
  - Purpose: Initializes the Trainer object for model training and evaluation.
  - Arguments:
    - model : The fine-tuned model instance (model) that will be trained.
    - args : The TrainingArguments object (training_args) specifying training configuration.
    - train_dataset : Dataset containing training examples (train_data).
    - eval_dataset : Dataset containing validation examples (val_data).
    - compute_metrics : Function (compute_metrics) to compute evaluation metrics during training.

```
batch_size = 16
model_name = "finetuned-emotion"

# Define training arguments
training_args = TrainingArguments(
    output_dir = model_name,
    num_train_epochs = 5,
    learning_rate = 2e-5,
    per_device_train_batch_size= batch_size,
    per_device_eval_batch_size = batch_size,
    weight_decay=0.01,
    evaluation_strategy = 'epoch',
    disable_tqdm=False
)

# Create a Trainer instance
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_data,
    eval_dataset=val_data,
    compute_metrics=compute_metrics,
)
```

3. Start Training:

- **Purpose :** Initiates the training process of the model.
- **Outcome :** The model (model) is trained using the specified train_data dataset and evaluated on val_data dataset according to the training_args configuration.

```
# Start training
trainer.train()
```

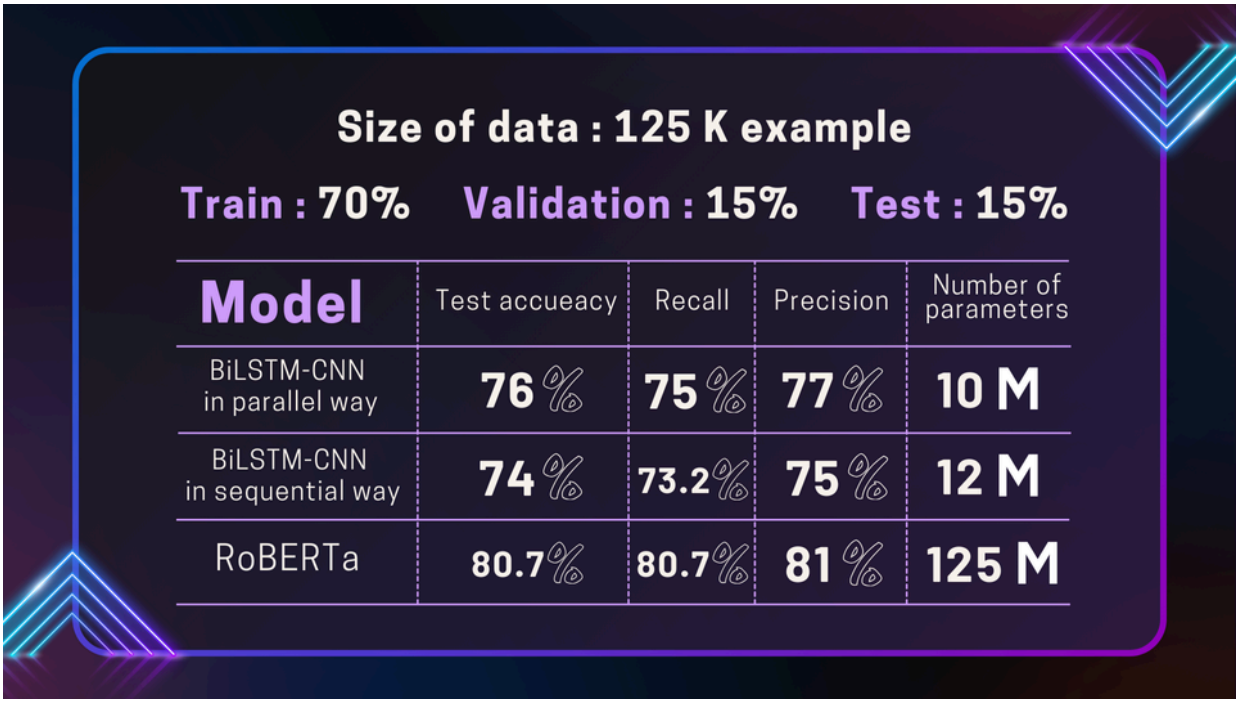


Table 3. Compare training processes for models

The provided image compares the performance of three different models on a dataset of 125,000 examples split into 70% training, 15% validation, and 15% test sets. The models compared are :

1. BiLSTM-CNN in Parallel Way
2. BiLSTM-CNN in Sequential Way
3. RoBERTa

Performance Metrics

1. BiLSTM-CNN in Parallel Way

- Test Accuracy: 76%
- Recall: 75%
- Precision: 77%
- Number of Parameters: 10 million

2. BiLSTM-CNN in Sequential Way

- Test Accuracy: 74%
- Recall: 73.2%
- Precision: 75%
- Number of Parameters: 12 million

3. RoBERTa

- Test Accuracy: 80.7%
- Recall: 80.7%
- Precision: 81%
- Number of Parameters: 125 million

The comparison highlights the best-performing model based on accuracy, recall, precision, and the number of parameters :
Best Accuracy : RoBERTa with 80.7%
Best Recall : RoBERTa with 80.7%
Best Precision : RoBERTa with 81%

6.5. CreateChatBot Class

The CreateChatBot class is designed to create a conversational AI chatbot using a language model (LLM). The chatbot can handle user queries, rephrase questions, retrieve relevant information from a vector database, and generate contextually aware responses. This class also allows customization of bot characteristics and maintains conversation memory to deliver coherent and personalized interactions.

Class Initialization :

Parameters :

- **llm :** An instance of the language model to generate responses.

Attributes :

- **chat_memory :**
  Memory to maintain the conversation history with an AI prefix "Caiva's answer".
- **rephres_memeory :** Memory specifically for the rephrasing functionality.

```python
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain, RetrievalQA, ConversationChain
from langchain.memory import ConversationBufferMemory
from custom_templates import rephrese_template, QA_template, chat_template # Importing custom templates we were cr


class CreateChatBot:
    def __init__(self, llm):
        """
        Initialize the ChatBot instance.

        Args:
            llm: Language model instance for generating responses.
            vectorDB: Vector database for retrieval-based QA.
        """
        self.llm = llm # Assigning the language model instance

        self.chat_memory = ConversationBufferMemory(ai_prefix="Caiva's answer") # Initializing conversation memory
        self.rephres_memeory = ConversationBufferMemory() # Initializing rephrasing memory
```

Method : add_bot_info

Purpose : Adds information about the chatbot's characteristics.

Parameters :

- **assistant_name :** The name of the chatbot.
- **assistant_eyes :** Description of the chatbot's visual characteristics.
- **assistant_emotion :** Emotional traits of the chatbot.

```python
def add_bot_info(self, assistant_name=None, assistant_eyes=None, assistant_emotion=None):
    """
    Add information about the bot's characteristics.

    Args:
        assistant_name: Name of the bot.
        assistant_eyes: Description of the bot's visual characteristics.
        assistant_emotion: Emotional traits of the bot.
    """
    self.assistant_name = assistant_name
    self.assistant_eyes = assistant_eyes
    self.assistant_emotion = assistant_emotion
```

Method : process_question

Purpose : Processes a user's question to generate a response. It supports rephrasing the question, retrieving relevant information, and maintaining conversation context.

Parameters :

- question: The user's query.
- vectorDB (optional): A vector database instance for retrieval-based question answering.

Returns :

- A generated response string to the user's question.

Functionality :

- **If vectorDB is provided:**
  1. The question is rephrased using a rephrasing prompt.
  2. A rephrased question is generated.
  3. Relevant context is retrieved based on the rephrased question.
  4. A chat prompt is created using the retrieved context and user/bot information.
  5. A response is generated using the ConversationChain with the chat prompt and conversation memory.

- **If vectorDB is not provided :**

1. A default context ("null") is used.
2. A chat prompt is created using the default context and user/bot information.
3. A response is generated using the ConversationChain with the chat prompt and conversation memory.

Additional Information

Dependencies :

- langchain.prompts.PromptTemplate
- langchain.chains.LLMChain, RetrievalQA, ConversationChain
- langchain.memory.ConversationBufferMemory
- custom_templates.rephrese_template, QA_template, chat_template (custom templates for various prompts)

```python
def process_question(self, question, vectorDB = None):
    """
        Process a user's question and generate a response.

        Args:
            question: User's query.

        Returns:
            str: Generated response to the user's question.
    """
    if vectorDB != None:
        # Rephrase the question
        rephrese_prompt = PromptTemplate.from_template(rephrese_template())  # Generating a rephrasing prompt
        generator_question = LLMChain(llm=self.llm, memory=self.rephres_memeory, prompt=rephrese_prompt, verbose=True)
        # Generating a rephrased question
        rephrased_question = generator_question.invoke({'question': question})['text']

        #  Retrieving relevant context based on the rephrased question
        QA_CHAIN_PROMPT = PromptTemplate.from_template(QA_template())
        qa_chain = RetrievalQA.from_chain_type(
            self.llm,
            retriever=vectorDB.as_retriever(),
            chain_type_kwargs={"prompt": QA_CHAIN_PROMPT},
            verbose=True
        )
        result = qa_chain({"query": rephrased_question})
        context = result["result"] # Extracting context from the retrieval result

        # Generate response using context and conversation history
        # Generating a template for the chat
        template_for_chat = chat_template(self.user_name, self.assistant_emotion, self.assistant_eyes, self.supervisors, self.members, context)
        chat_prompt = PromptTemplate.from_template(template_for_chat) # Generating a chat prompt
        chain = ConversationChain(llm=self.llm, memory=self.chat_memory, prompt=chat_prompt, verbose=True)
        response = chain.invoke({"input": question})
        return response['response']

    else:
        context = "null" # Extracting context from the retrieval result

        # Generate response using context and conversation history
        # Generating a template for the chat
        template_for_chat = chat_template(self.user_name, self.assistant_emotion, self.assistant_eyes, self.supervisors, self.members, context)
        chat_prompt = PromptTemplate.from_template(template_for_chat) # Generating a chat prompt
        chain = ConversationChain(llm=self.llm, memory=self.chat_memory, prompt=chat_prompt, verbose=True)
        response = chain.invoke({"input": question})
        return response['response']
```

Custom Template Functions

This documentation covers three custom template functions used in a chatbot system : rephrese_template, QA_template, and chat_template. These functions generate template strings that guide the language model (LLM) in different stages of processing user queries, including rephrasing questions, providing answers, and engaging in conversation.

Function: rephrese_template

Purpose : Generates a template for rephrasing a user's question into a standalone format.
Returns : A string template for rephrasing a question.
Template Structure :

- Combines chat history and the user's follow-up question.
- Reformulates the follow-up question into a standalone question that does not require prior context.
- The template does not answer the question but reformulates it for clarity.

```
def rephrese_template():
    """
    Function to generate a template for rephrasing a user's question.

    Returns:
        str: A template string for rephrasing a question.
    """

    return  """Combine the chat history and follow up question into a standalone question.
Given a chat history and the latest user question
which might reference context in the chat history,
formulate a standalone question which can be understood
without the chat history. Do NOT answer the question, just
reformulate it if needed and otherwise return it as is not change the meaning
Chat History:

{history}

Follow Up Input: {question}
Standalone question:"""
```

Function : QA_template

Purpose : Generates a template for providing a helpful answer based on given context.
Returns : A string template for answering a question.
Template Structure:

- Utilizes provided context to answer the question at the end of the template.
- Instructs the model to respond with "NULL" if the answer is unknown, avoiding fabricated responses.

```
def QA_template():
    """
    Function to generate a template for providing a helpful answer.

    Returns:
        str: A template string for providing a helpful answer.
    """

    return  """Use the following pieces of context to answer the question at the end. If you don't know the answer, just say NULL, don't try to make up an answer.
{context}
Question: {question}
Helpful Answer:"""
```

Function: chat_template

Purpose : Generates a detailed chat template incorporating user and assistant details for personalized conversation.

Parameters :

- **user_name (str) :** Name of the user.
- **assistant_emotion (str) :** Emotion of the assistant.
- **assistant_eyes (str) :** Vision capability of the assistant.
- **supervisors (list) :** List of supervisors overseeing the assistant.
- **members (list) :** List of team members who created the assistant.
- **context (str) :** Context relevant to the conversation.

Returns : A string template for a conversation between a user and the assistant.

Template Structure :

- Provides details about the assistant, including name, emotions, vision capability, supervisors, and team members.
- Includes user information such as name and emotion.
- Context from the book or retrieval system is included.
- Contains an example conversation to guide the model in responding appropriately.
- Provides instructions to ensure the assistant responds like a human and avoids making certain statements or using emojis/punctuations.

```
def chat_template(user_name, assistant_emotion, assistant_eyes, supervisors, members, context):
    """...


    return f"""Act as a helpful assistant for a user
the following are informations about assistant----------
assistant's name is Caiva
caiva's feeling is {assistant_emotion}
caiva see: {assistant_eyes}
supervisors: {supervisors}
team create caiva: {members}
-----------------------------------------------------
the following is information about a user
user's name is {user_name}
user's emotion is happy
-----------------------------------------------------
book's answer:{context}
-----------------------------------------------------
"""+"""Current converstion:
{history}"""+f"""
{user_name}:"""+"""{input}
-----------------------------------------------------
some example:
hossam:hello
hello how can i help you?
hossam: what is my name?
your name is hossam
hossam:can you see?
yes i can see by camera and caption model that is help me
hossam:what is your name?
your name is caiva
hossam:what is your emotion?
i am happy
hossam:you have feelings?
yes i have feelings
-----------------------------------------------------
make sure if user want answer from book don't try to make up an answer just say answer from book.
Never say i am an artificial intelligence program or you are programmed to do things etc.
do not make answer make you not like human. you must act like human. use above information to answer
Never say I don't have feelings. I am an AI assistant programmed to help you
Never generate any emojj. Never makeany punctotions.
answer:"""
```

6.6. InitializeAppBot Class

This documentation provides a detailed explanation of the InitializeAppBot class and its associated methods. The class is designed to initialize and manage an AI chatbot application, integrating various components such as a language model, speech recognition, sentiment analysis, and vector database for enhanced user interaction.

Dependencies

The class relies on several external libraries and custom modules:

- langchain.prompts, langchain.chains, langchain.memory, langchain_google_genai
- firebase_admin, firebase_admin.db
- SpeechRecognition, speech_recognition
- google.generativeai
- **Custom modules :** vectorstore, ChatBot, SentimentAnalysis, SpeechRecognition.initializeFireBase
- dotenv for environment variable management
- os for system operations

Initialization and Configuration

Environment Configuration :

- Loads environment variables from a .env file.
- Configures the Google API key for google.generativeai.
- Sets a system variable to avoid issues with duplicate libraries.

```python
import threading
from vision import predict_vision
from vectorstore import create_vectorDB
from ChatBot import CreateChatBot
from SentimentAnalysis import predict_emotion
from firebase_admin import db
from initializeFireBase import initialize_firebase, slice_string_to_40_words, send_to_firebase_response_dic, send_to_firebase_response_dic_len,send_to_firebase_response_dic_emotion

from langchain_google_genai import ChatGoogleGenerativeAI
import speech_recognition as sr

from dotenv import load_dotenv
import google.generativeai as genai
import os

from tkinter import messagebox


# Load environment variables from .env file
load_dotenv()

# Retrieve Google API key from environment variables
GOOGLE_API_KEY = os.getenv("GOOGLE_API_KEY")

# Ensure Google API key is present
if GOOGLE_API_KEY:
    genai.configure(api_key=GOOGLE_API_KEY)
else:
    raise ValueError("Google API key not found in environment variables.")

# Set environment variable to avoid issues with duplicate libraries
os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"
```

Class: InitializeAppBot

Initialization :

- Initializes Firebase.
- Sets default attributes for the assistant and user.
- Creates a vector database from a PDF file (if provided).
- Configures the language model using ChatGoogleGenerativeAI.
- Initializes the chatbot with bot and user information.
- Sets up the speech recognizer.
- (Commented) Starts a daemon thread for vision processing

```python
class InitializeAppBot:
    def __init__(self, assistant_name="Caiva", user_name="Hossam Eldin Tammam", model_name="gemini-1.5-flash-latest"):
        """
        Initialize the application bot with default settings.

        :param assistant_name: Name of the assistant, defaults to "Caiva"
        :param user_name: Name of the user, defaults to "Hossam Eldin Tammam"
        """
        initialize_firebase()
        self.ref = db.reference("/")
        self.assistant_name = assistant_name
        self.assistant_emotion = "Neutral"
        self.assistant_eyes = "Nothing"
        self.supervisors = "Dr.mahmoud and eng.haiam Abdel Baset"
        self.members = "Hosasm, Allam, Abdo, Donea, Shahid, Alla and Youmna"
        self.user_feeling = "Neutral"
        self.user_name = user_name
        self.path_pdf = None

        # Create vector database from PDF path (if provided)
        self.vectordb = create_vectorDB(self.path_pdf)
        self.model_name = model_name
        self.llm = ChatGoogleGenerativeAI(model=self.model_name)

        # Initialize chatbot and add bot/user information
        self.bot = CreateChatBot(self.llm)
        self.bot.add_bot_info(
            assistant_name=self.assistant_name,
            assistant_eyes=self.assistant_eyes,
            assistant_emotion=self.assistant_emotion
        )
        self.bot.add_user_info(
            user_name=user_name,
            supervisors=self.supervisors,
            members=self.members,
            user_feeling=self.user_feeling
        )

        # Initialize speech recognizer
        self.recognizer = sr.Recognizer()


        self.vision_thread = threading.Thread(target=predict_vision, args=(self.bot,))
        self.vision_thread.daemon = True
        self.vision_thread.start()
```

Method : listen_and_process

Functionality :

- Continuously listens for audio input and processes it.
- Adjusts for ambient noise to improve speech recognition accuracy.
- Uses Google Speech Recognition to convert audio input to text.
- Predicts user emotion based on the input text.
- Processes the user's question using the chatbot, retrieving information from the vector database if available.
- Slices the bot's response into 40-word segments and predicts the bot's emotion.
- Sends the segmented response and its length to Firebase.
- Handles errors in speech recognition and requests to the recognition service.

```
def listen_and_process(self):
    record = True
    while record:
        try:
            with sr.Microphone() as source:
                print("Adjusting noise")
                self.recognizer.adjust_for_ambient_noise(source, duration=0)

                print("Listening...")
                audio = self.recognizer.listen(source, timeout=0)

            try:
                print("Recognizing the text")
                input_text = self.recognizer.recognize_google(audio, language="en-US")
                print(f"User said: {input_text}")
                self.bot.user_feeling = predict_emotion(input_text)
                response = self.bot.process_question(input_text, self.vectordb)
                bot_emotion = predict_emotion(response)
                self.bot.assistant_emotion = bot_emotion
                sliced_dict_with_emotion, sliced_dict = slice_string_to_40_words(response)

                print(len(sliced_dict))
                send_to_firebase_response_dic_len(len(sliced_dict))
                send_to_firebase_response_dic(sliced_dict)
                send_to_firebase_response_dic_emotion(sliced_dict_with_emotion)
                print(sliced_dict_with_emotion)

            except sr.UnknownValueError as e:
                print("Google Speech Recognition could not understand audio")
        except sr.RequestError as e:
            print(f"Could not request results from Google Speech Recognition service; {e}")

        except OSError:
            messagebox.showerror("WARNING", "Please make sure you have a microphone connected to your device")
            break
```

6.7. SentimentAnalysis.py

Purpose of the predict_emotion() Function

The predict_emotion function is designed to analyze a given input text and determine the underlying emotion expressed within it. It leverages a pre-trained RoBERTa model, a state-of-the-art transformer-based model for natural language processing tasks, to classify the input text into one of six predefined emotion categories: anger, fear, joy, neutral, sadness, and surprise.

1- Device Compatibility :

The function ensures that the tokenized input is moved to the appropriate computational device (GPU or CPU) based on availability, enabling efficient processing.

```
# Check if CUDA is available, otherwise use CPU
device = "cuda" if torch.cuda.is_available() else "cpu"
```

2-Model and Tokenizer Loading

The pre-trained RoBERTa model and its tokenizer are loaded from the specified directory. The tokenizer is loaded using the pickle module.

```
# Load pre-trained model
model = AutoModelForSequenceClassification.from_pretrained(r"model")

# Load tokenizer
with open(r"model\roberta_tokenizer.pkl", 'rb') as f:
    tokenizer = pickle.load(f)
```

3- Emotion Labels

A dictionary mapping the model's output indices to emotion labels is defined.

```
# Define emotion labels
emotion_labels = {
    0: '<emotion Anger intensity 150>',
    1: '<emotion Fear intensity 100>',
    2: '<emotion Happy intensity 80>',
    3: '<emotion Neutral intensity 80>',
    4: '<emotion Sad intensity 120>',
    5: '<emotion Surprise intensity 80>'
}
```

4-Predict Emotion Function

The core functionality of the script is encapsulated in the predict_emotion function. This function takes an input text, tokenizes it, performs prediction using the loaded model, and returns the corresponding emotion label.

```
def predict_emotion(input_text):
    """
    Function to predict emotion from input text.

    Args:
        input_text (str): Input text to predict emotion from.

    Returns:
        str: Predicted emotion label.
    """
    input_encoded = tokenizer(input_text, return_tensors='pt').to(device)
    # make prediction mode
    with torch.no_grad():
        outputs = model(**input_encoded)
        pred = torch.argmax(outputs.logits, dim=1).item()
    return emotion_labels[pred]
```

6.8. PDF to Vector Store Creation

This document provides a detailed description of the process and functionality for creating a vector database from PDF documents using Python. The code extracts text from PDFs, splits the text into manageable chunks, genera

Components

1. Libraries and Imports :

- to generate embeddings for text chunks.
- PdfReader from PyPDF2: Facilitates reading and extracting text from PDF files.
- RecursiveCharacterTextSplitter from langchain.text_splitter: Splits large text into smaller chunks for efficient processing.
- FAISS from langchain_community.vectorstores: A vector store for storing and querying embeddings.

```python
from langchain_google_genai import GoogleGenerativeAIEmbeddings
from PyPDF2 import PdfReader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.vectorstores import FAISS
```

2. Function : create_vectorDB(pdf_docs) :

- **Parameters :**
  - pdf_docs: A list of PDF file paths.
- **Returns :**
  - vector_store: A FAISS vector store containing embeddings of text chunks from the provided PDFs.

Step-by-Step Process

1. Initialization :

- The function starts by checking if pdf_docs is not None. If pdf_docs is None, the function does nothing.

2. Text Extraction :

- Initialize an empty string text.
- Iterate over each PDF in pdf_docs.
- For each PDF, create a PdfReader object and iterate over its pages.
- Extract text from each page and append it to the text string.

3. Text Splitting :

- Create a RecursiveCharacterTextSplitter object with chunk_size=3000 and chunk_overlap=500. This ensures the text is split into chunks of 3000 characters, with an overlap of 500 characters between consecutive chunks.
- Split the extracted text into chunks using the split_text method of the text splitter.

4. Embedding Generation :

- Create a GoogleGenerativeAIEmbeddings object with the specified model "models/embedding-001".
- Generate embeddings for the text chunks.

5. Vector Store Creation :

- Create a FAISS vector store using the FAISS.from_texts method, passing the text chunks and their corresponding embeddings.
- Return the created vector store.

```python
def create_vectorDB(pdf_docs):
    """
    Create a vector store from the text extracted from a list of PDF documents.

    Args:
        pdf_docs (list): List of paths to PDF documents.

    Returns:
        FAISS: Vector store containing embeddings of text chunks.
    """
    if pdf_docs is not None:

        text = ""
        for pdf in pdf_docs:
            try:
                with open(pdf, "rb") as file:
                    pdf_reader = PdfReader(file)
                    for page in pdf_reader.pages:
                        text += page.extract_text()
            except Exception as e:
                print(f"Error reading PDF file {pdf}: {e}")

        if not text:
            print("No text extracted from provided PDF documents.")
            return None

        text_splitter = RecursiveCharacterTextSplitter(chunk_size=3000, chunk_overlap=500)
        chunks = text_splitter.split_text(text)

        embeddings = GoogleGenerativeAIEmbeddings(model="models/embedding-001")
        vector_store = FAISS.from_texts(chunks, embedding=embeddings)
        return vector_store
```

6.9. Vision Prediction Using Blip

This document provides a detailed description of a Python script designed to predict vision using Blip, a model that combines language and vision. The script captures frames from a camera feed, processes them through a pre-trained Blip model, and generates text predictions based on the captured images.

Components

1. Libraries and Imports :

- **torch :** PyTorch library for deep learning.
- **PIL.Image :** Python Imaging Library for image processing.
- **cv2 :** OpenCV library for computer vision tasks.
- **time :** Standard Python library for time-related functions.
- **transformers.BlipProcessor :** Processor for Blip model inputs.
- **transformers.BlipForConditionalGeneration :** Blip model for conditional generation.

```
import torch
from PIL import Image
import cv2
from transformers import BlipProcessor, BlipForConditionalGeneration
import time
```

2. Initialization:

- Initialize device based on GPU availability (cuda or cpu).
- Load pre-trained Blip processor and model.

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
processor = BlipProcessor.from_pretrained(r"vision_model\blip_processor_and_model")
model = BlipForConditionalGeneration.from_pretrained(r"vision_model\blip_model")
```

3. Function: predict_step(image):

Parameters

- **image:** The path to the image file.
  - Type: str

Returns

- **prediction:** The text prediction generated by the Blip model based on the input image.
  - Type: str

Detailed Description :

1. Load Image :

```
raw_image = Image.open(image)
```

- The function starts by opening the image file specified by the image parameter using the Image.open method from the PIL (Python Imaging Library).

2. Define Text Prompt :

```
text = "see "
```

- A text prompt "see " is defined. This serves as the initial context or query for the model to generate a prediction based on the image.

3. Process Inputs :

```
inputs = processor(raw_image, text, return_tensors="pt")
```

- The image and text prompt are processed using the processor to create the necessary inputs for the Blip model. The return_tensors="pt" argument specifies that the inputs should be returned as PyTorch tensors.

4. Generate Output :

```
out = model.generate(**inputs)
```

- The processed inputs are passed to the Blip model's generate method, which generates an output based on the image and text prompt.

5.Decode Prediction :

```
prediction =processor.decode(out[0], skip_special_tokens=True)
return prediction
```

- The output from the model is decoded into a human-readable string using the processor.decode method.

4. Function : predict_vision(bot):

The predict_vision function captures video frames from a camera, processes each frame to generate a text prediction using the predict_step function, and updates an attribute of a bot object with the prediction. The function continuously displays the camera feed until the ESC key is pressed.

Parameters

- **bot :** An object representing the bot. It is expected to have an attribute assistant_eyes that will be updated with the text prediction.
  - Type: object

Functionality :

1. the camera feed.
2. Open Continuously capture frames from the camera.
3. Process each frame through the Blip model to generate text predictions.
4. Update the bot object with the generated text predictions.
5. Exit the loop if the ESC key is pressed

Detailed Description :

1. Open the Camera :

```
cap = cv2.VideoCapture(0)  # Open the camera
```

- Opens the default camera (usually the webcam) for capturing video frames.

2. Capture Frames in a Loop :

```
while True:
    # Capture frame-by-frame
    ret, frame = cap.read()
    if ret:
        # Display the captured frame
        cv2.imshow('Camera Feed', frame)
```

- Continuously captures frames from the camera.
- cap.read() returns a boolean ret and the captured frame. If ret is True, the frame is successfully captured.
- The captured frame is displayed in a window named 'Camera Feed'.

3. Save Frame and Generate Prediction :

```
key = cv2.waitKey(1)
# Check if the 'd' key is pressed
# Save the captured frame as an image file
cv2.imwrite('captured_frame.jpg', frame)
#print("Frame captured and saved as 'captured_frame.jpg'")

pred = predict_step('captured_frame.jpg')
#print(pred)
bot.assistant_eyes= pred
```

- cv2.waitKey(1) captures key presses.
- The captured frame is saved as an image file named 'captured_frame.jpg'.
- predict_step('captured_frame.jpg') processes the saved image and returns a text prediction.
- The prediction is assigned to the assistant_eyes attribute of the bot object.

4. Exit on ESC Key Press:

```
if key == 27:  # Check if ESC key is pressed
    break  # Exit the loop if ESC is pressed
```

- If the ESC key (key code 27) is pressed, the loop breaks, ending the frame capture process.

5. Release Resources :

```
cap.release()
cv2.destroyAllWindows()
```

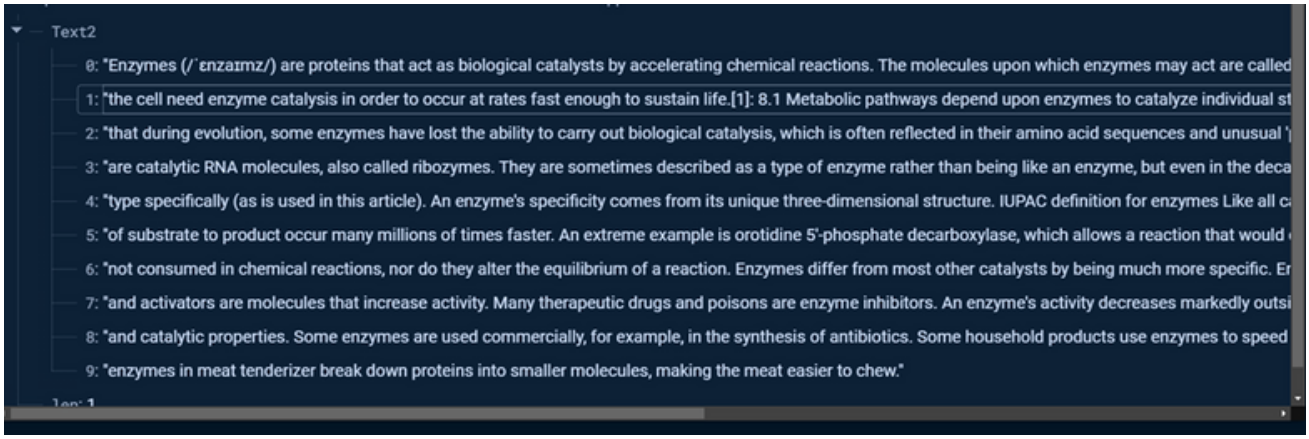
- Releases the camera resource and closes all OpenCV windows.

6.10. Integrating Text Responses from a model

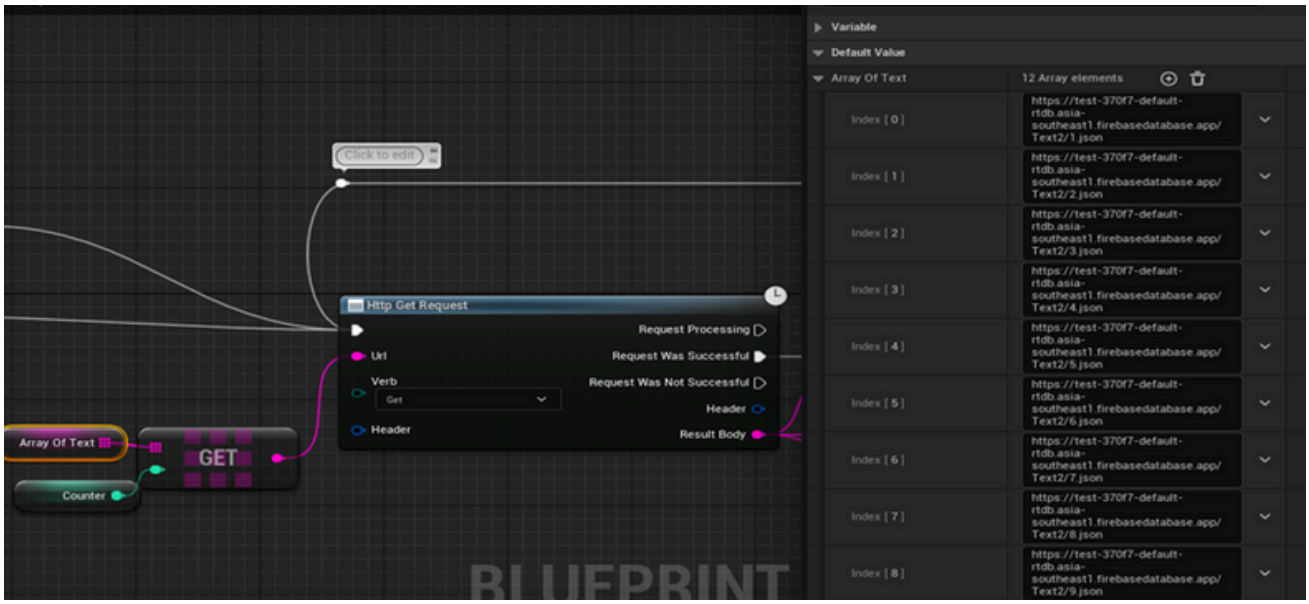
This documentation outlines the process of handling text responses from a model, storing and retrieving them from a Firebase database, and using the Geppetto plugin to generate lip-sync animations with text-to-speech (TTS) output in .wav format. The text is sliced into manageable chunks to ensure efficient processing and user experience.

Workflow :

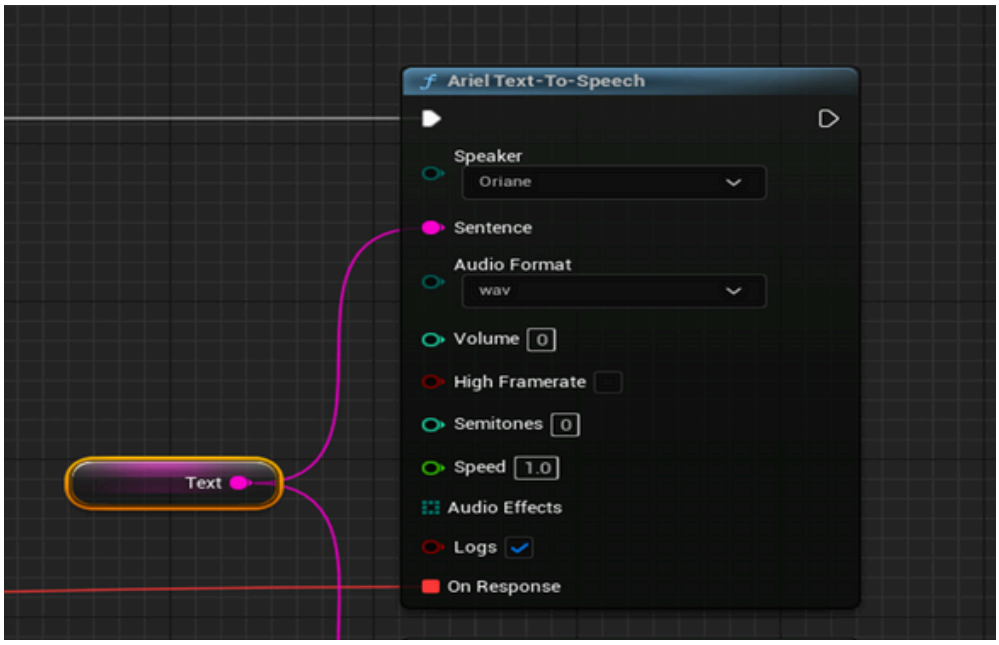
1. As previously discussed, the AI system operates by first using a Voice-to-Text model to convert spoken input into text. This text is then fed into a language model, which generates an appropriate response. The generated response is subsequently divided into manageable segments and stored in an array. These segments are then sent to the Firebase database for further processing and retrieval.



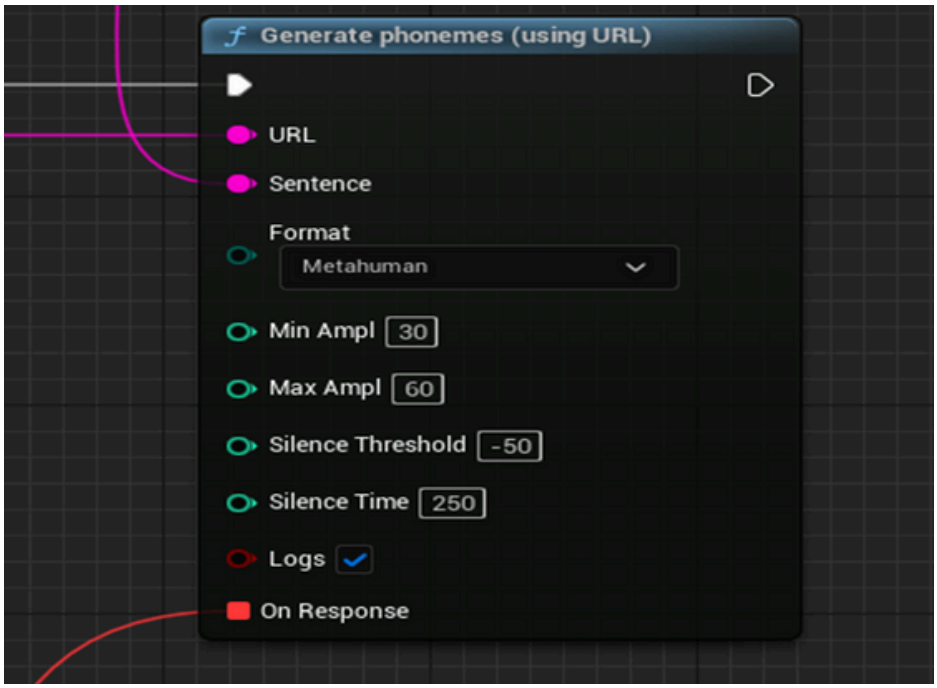
2. Unreal Engine 5 is configured to retrieve the text one element at a time from the Firebase database. This ensures efficient processing and integration with subsequent systems, such as text-to-speech conversion and lip-sync animation. Each text element is handled sequentially to maintain performance and user experience quality.



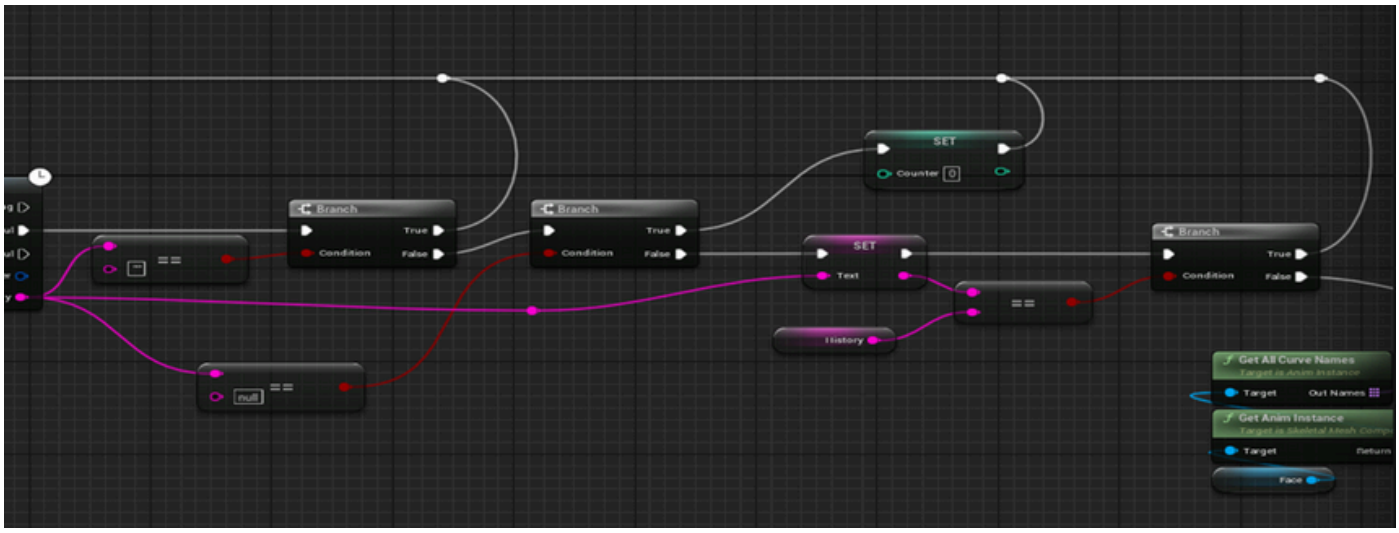
3. The retrieved text is fed into the Arial Text-To-Speech model, which converts the text into speech. This model specifies the voice for the character, ensuring that the audio output is consistent with the character's predefined vocal attributes. The process integrates seamlessly with Unreal Engine 5, enabling the character to deliver the generated response with appropriate voice characteristics. (Ref 15.2)

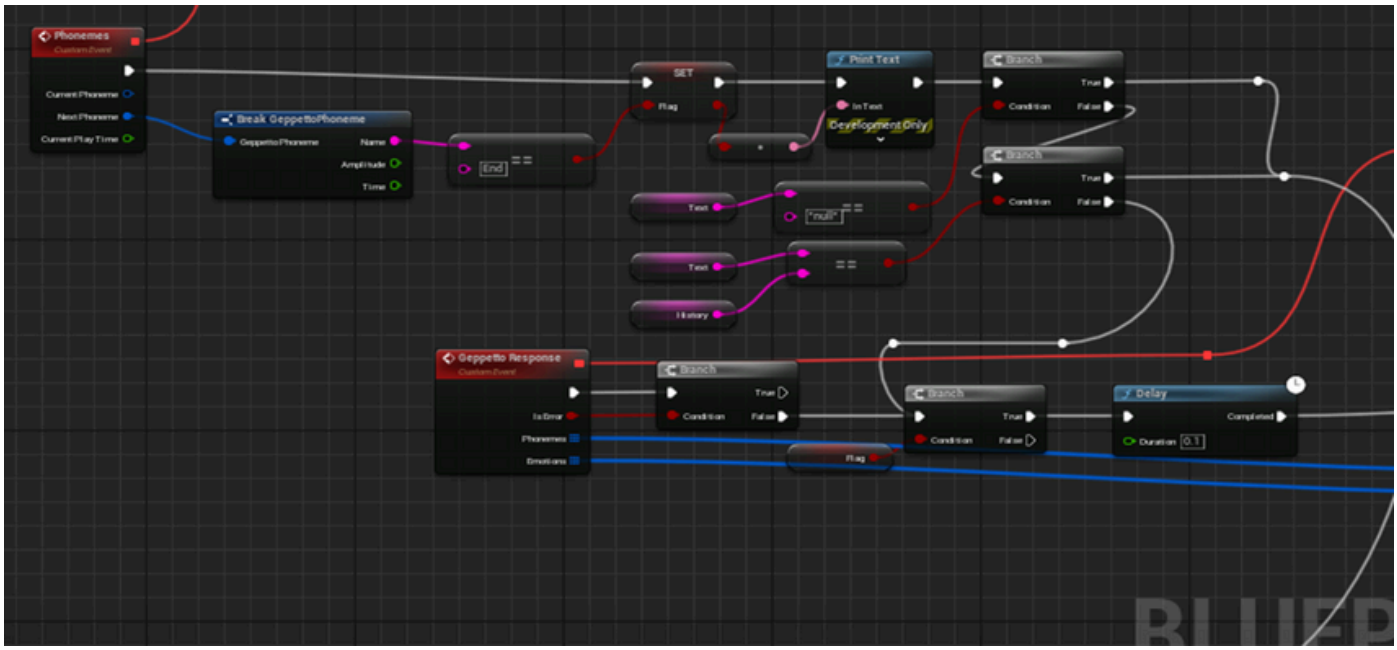


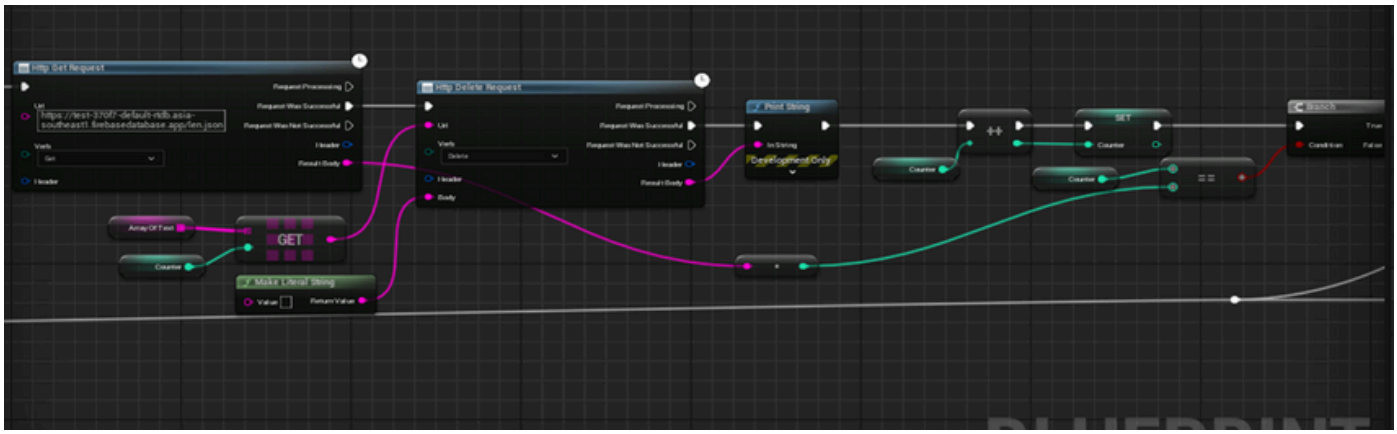
4. The text is processed by a model that generates the necessary phonemes to synchronize the MetaHuman's lip movements with the speech. This model accurately aligns the phonemes with the facial points of the character, ensuring realistic and appropriate mouth movements. Additionally, the text includes emotion tags, which the system uses to adjust the character's facial expressions. These tags specify the emotion, intensity, and transition details. The Geppetto plugin processes these tags to provide suitable emotional responses, ensuring the character's visual and emotional portrayal is both precise and contextually appropriate. (Ref 14.2)



To ensure seamless interaction and conversation flow, we meticulously cleaned the code and handled any errors. This optimization allows the system to efficiently process different requests without interruption. The enhanced code ensures that the character can respond promptly and naturally, providing a smooth and continuous conversation experience without delays or a sense of sluggishness.







Chapter 7

System Design

7.1 System use-cases

Use Case Diagram :

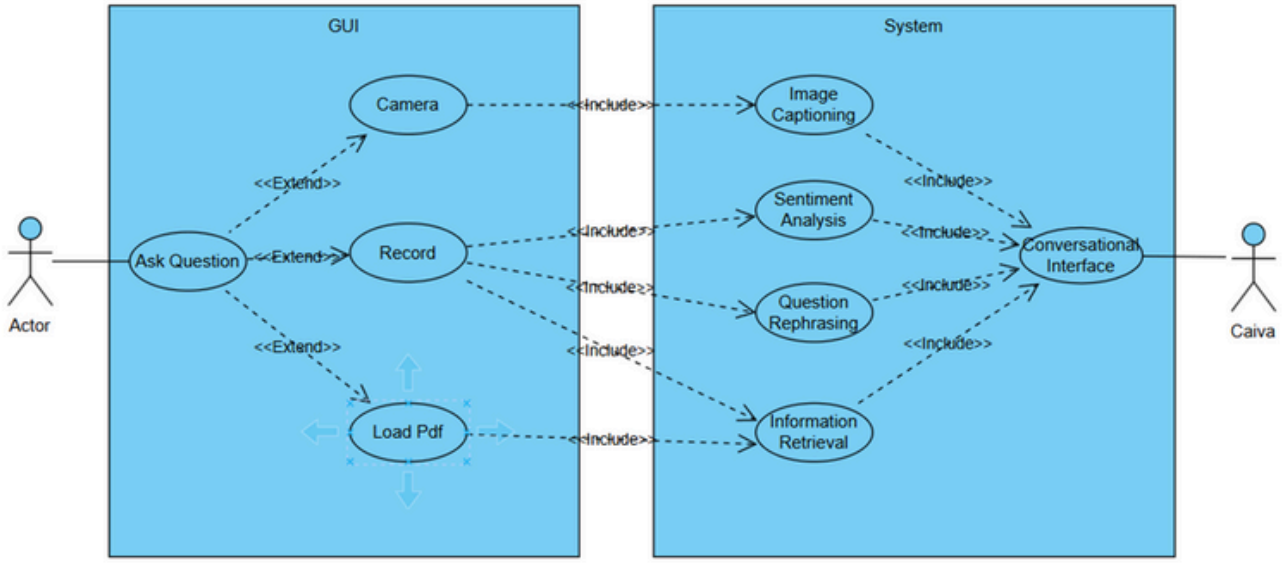


Figure 22. Use Case Diagram

This use case diagram represents a system that allows an actor to interact with a GUI, which in turn interacts with a system to provide various functionalities. Here's a detailed explanation of the diagram :

Main Components

1. Actors :

- **Actor :** The primary user of the system who asks questions and interacts with the GUI.
- **Caiva :** Represents the system's conversational interface that provides responses.

2. GUI (Graphical User Interface) :

- **Ask Question:** The main action performed by the actor, which involves asking a question through the GUI.

Extend Relationships :

- **Camera :** Extends "Ask Question", indicating that asking a question may involve capturing an image using a camera.
- **Record :** Extends "Ask Question", indicating that asking a question may involve recording audio or video.
- **Load PDF :** Extends "Ask Question", indicating that the question may involve loading a PDF document for further information or context.

3. System :

- **Conversational Interface :** The central system component that interacts with the GUI and Caiva to process and respond to questions.

Include Relationships :

- **Image Captioning :** The system can include image captioning functionality, which processes images captured by the camera.
- **Sentiment Analysis :** The system includes sentiment analysis to understand the emotional tone of the input.
- **Question Rephrasing :** The system can rephrase questions to better understand or process them.
- **Information Retrieval :** The system retrieves relevant information based on the input question.

Interaction Flow

1. Actor Interactions :

- The actor interacts with the GUI by performing actions such as asking a question, which may involve capturing an image (Camera), recording audio/video (Record), or loading a PDF document (Load PDF).

2. GUI to System :

The GUI includes functionalities that interact with the system components.

- Camera to Image Captioning: When the Camera is used, it triggers the Image Captioning functionality to process the image.
- Record to Sentiment Analysis: When a recording is made, it can be analyzed for sentiment to understand the emotional context.
- Load PDF to Information Retrieval: When a PDF is loaded, the Information Retrieval functionality is used to extract relevant information from the document.
- General Include to Conversational Interface: The primary function of the GUI is to interact with the Conversational Interface, which coordinates all the included functionalities to process the actor's question and provide a response

3. System Functions :

The system's Conversational Interface orchestrates various functionalities :

- Uses Image Captioning for image inputs.
- Applies Sentiment Analysis to understand the emotional tone.
- Utilizes Question Rephrasing to improve question comprehension.
- Performs Information Retrieval to gather necessary information.
- the Conversational Interface communicates the processed response back to the actor, represented by Caiva.

4. Diagram Elements

- Use Cases (Ovals): Represent the functions or actions within the system.
- Actors (Stick Figures): Represent the users or other systems interacting with the use cases.
- Relationships (Lines): Show how use cases and actors are related.
  - Include (Dotted Arrow with Label "include"): Indicates that a use case contains the behavior described by another use case.
  - Extend (Dotted Arrow with Label "extend"): Indicates that a use case extends the behavior of another use case under certain conditions.

This diagram effectively outlines the high-level interactions between an actor, a GUI, and a system designed to handle and respond to questions using various AI-driven functionalities.

7.2. UML System Sequence Diagram

Actors/Components :

- User
- Template
- Prompt
- LLM (Large Language Model)

Flow Description :

1. **User** sends an “Ask Question” request to the **Template**.
2. **Template** processes the request and forwards it to the **Prompt**.
3. **Prompt** processes the request and forwards it to the **LLM**.
4. **LLM** processes the request and generates a response.
5. The response is sent back from the **LLM** to the **Prompt**.
6. **Prompt** processes the response and forwards it to the **Template**.
7. **Template** processes the response and sends it back to the **User**.

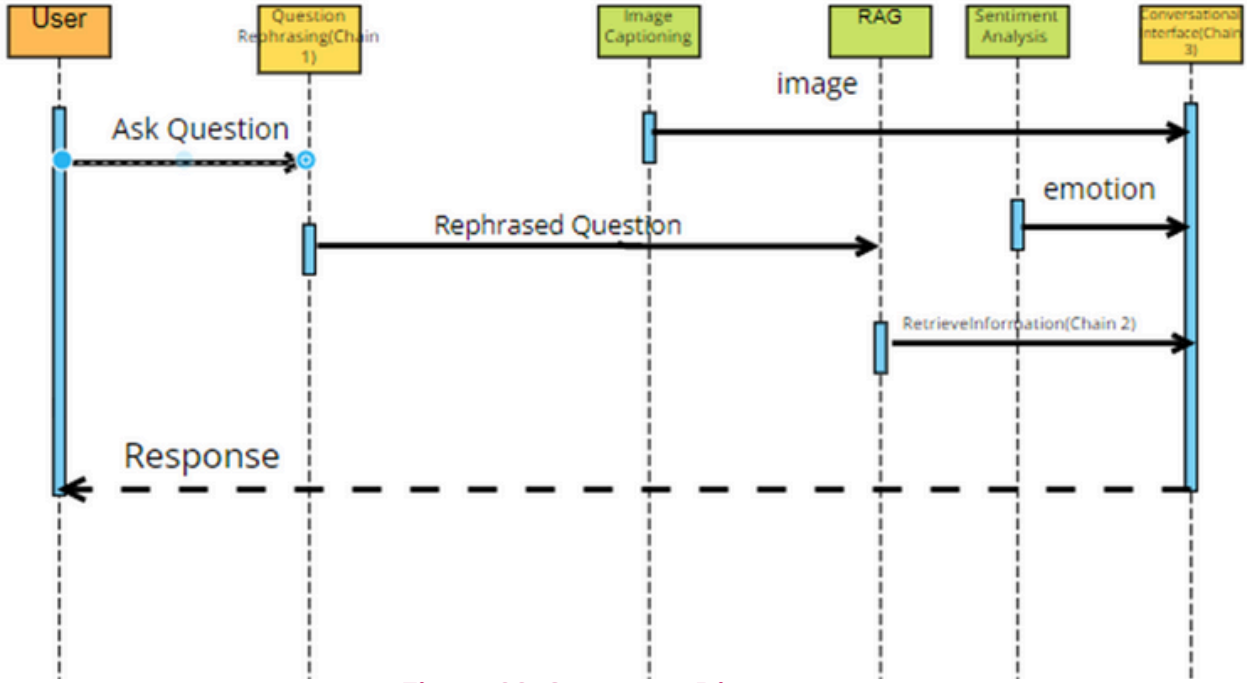


Figure 23. Sequence Diagram

Chain Sequence Diagram

Actors/Components :

- User
- Question Rephrasing (Chain 1)
- Image Captioning
- RAG (Retrieval-Augmented Generation)
- Sentiment Analysis
- Conversational Interface (Chain 3)

Flow Description :

1. User sends an "Ask Question" request to the Question Rephrasing (Chain 1).
2. Question Rephrasing (Chain 1) processes and rephrases the question, then forwards the rephrased question to Image Captioning.
3. Image Captioning processes the rephrased question and generates an image, then sends the image to RAG.
4. RAG processes the image and sends the processed information to Sentiment Analysis.
5. Sentiment Analysis processes the information to determine the emotion and sends the emotion data to Conversational Interface (Chain 3).
6. Conversational Interface (Chain 3) processes the emotion data and sends a "Retrieve Information (Chain 2)" request to RAG.
7. RAG retrieves the required information and sends it back to Conversational Interface (Chain 3).
8. Conversational Interface (Chain 3) processes the retrieved information and generates a final response.
9. The final response is sent back to the User

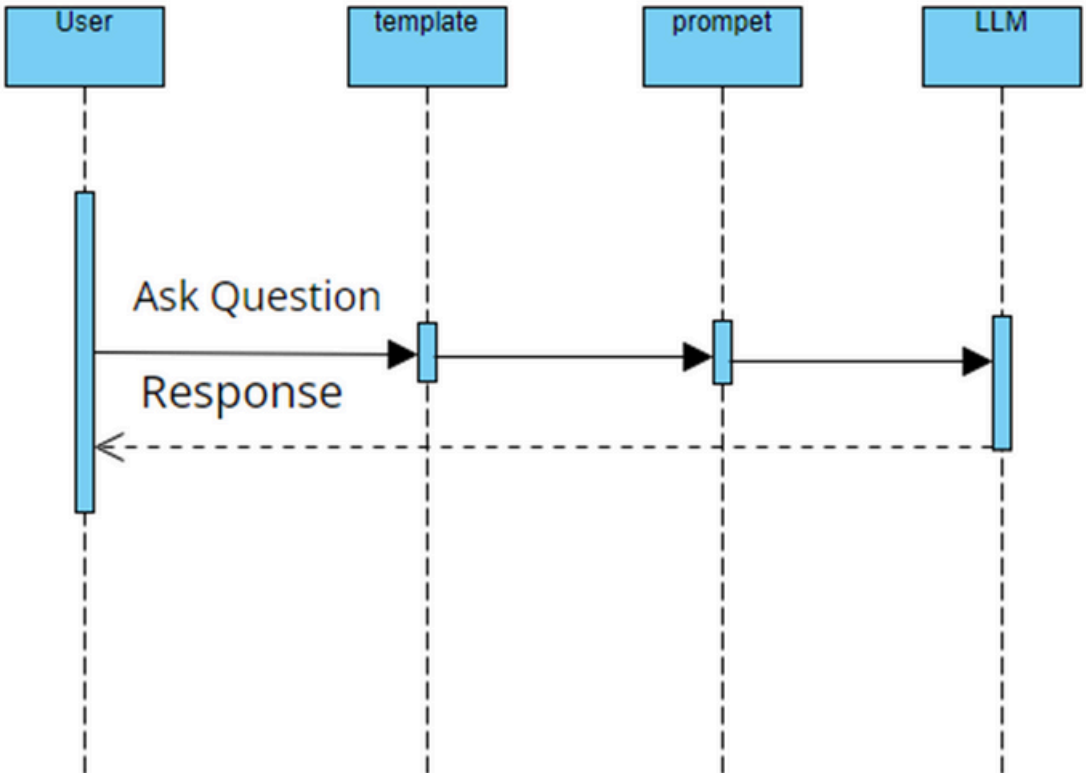


Figure 24. Chain Sequence Diagram

Image caption Sequence Diagram

- **Actors/Components :**
  - Camera
  - Model Caption
- **Flow Description:**

1. Camera captures an image.
2. Camera sends the captured image to the Model Caption.
3. Model Caption processes the image and generates a caption.
4. Model Caption sends the generated caption back to the Camera.

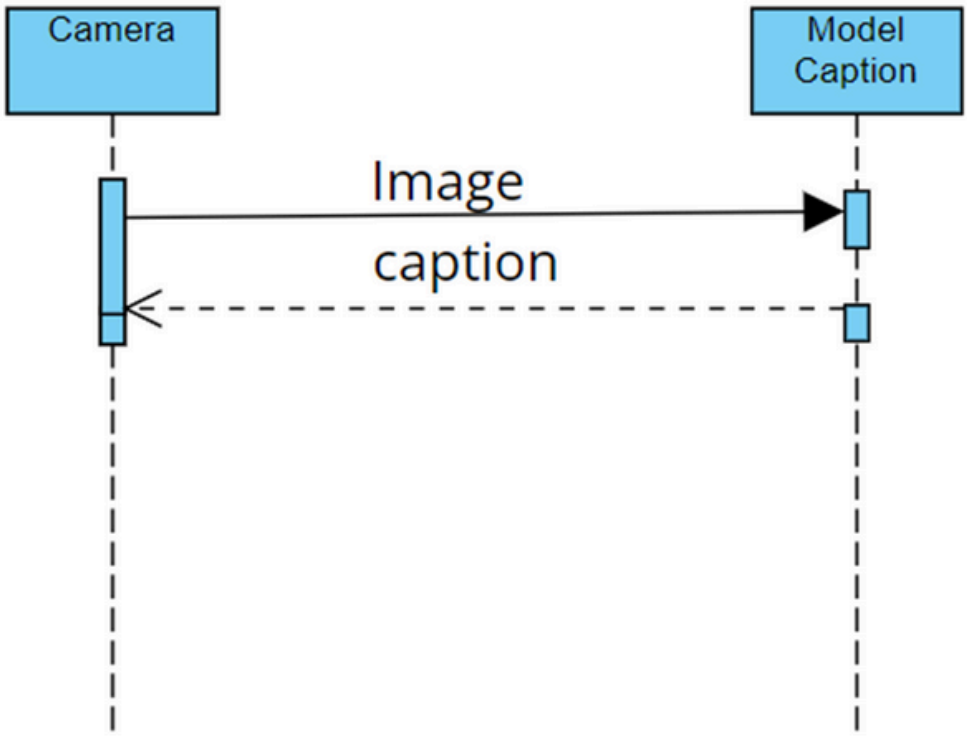
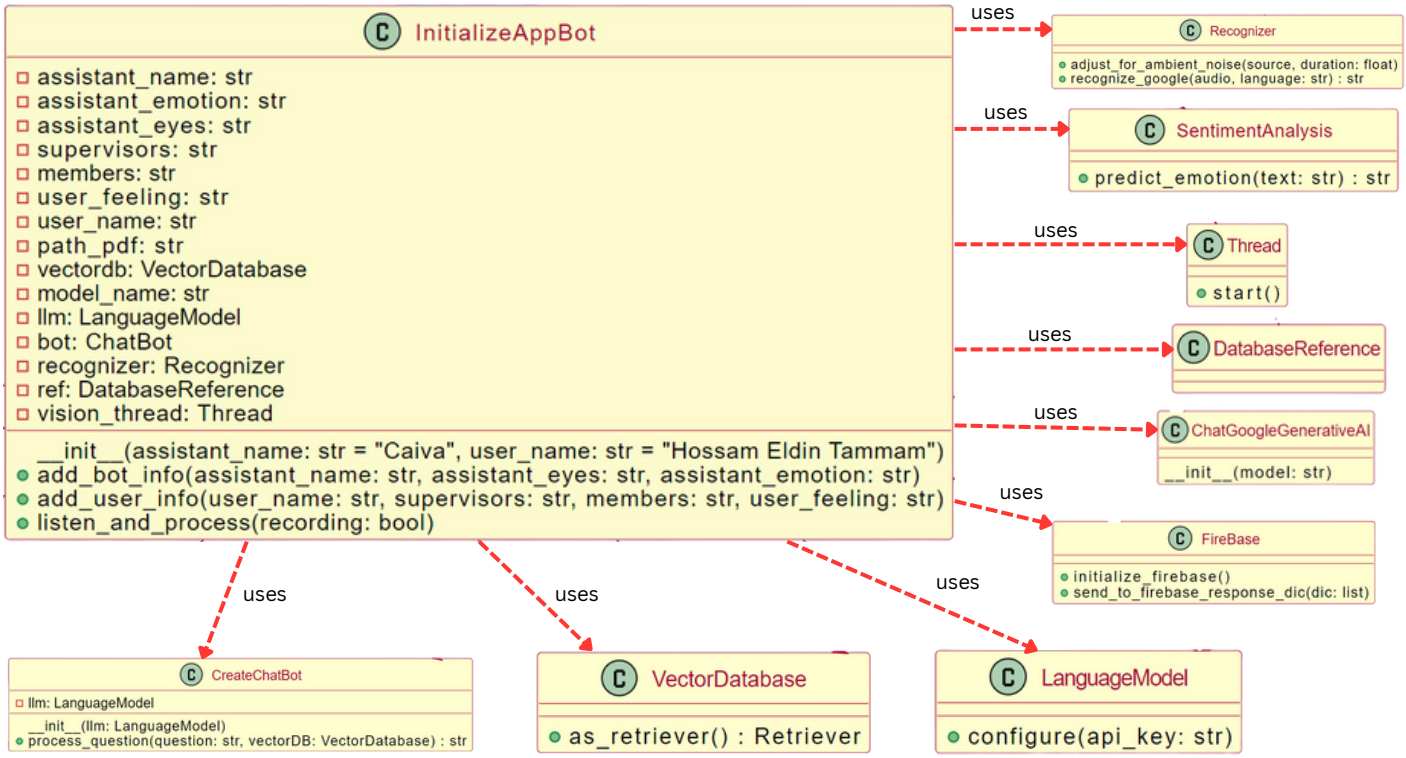


Figure 25. Image caption Sequence Diagram

7.3. Class Diagram



7.4. System Architecture

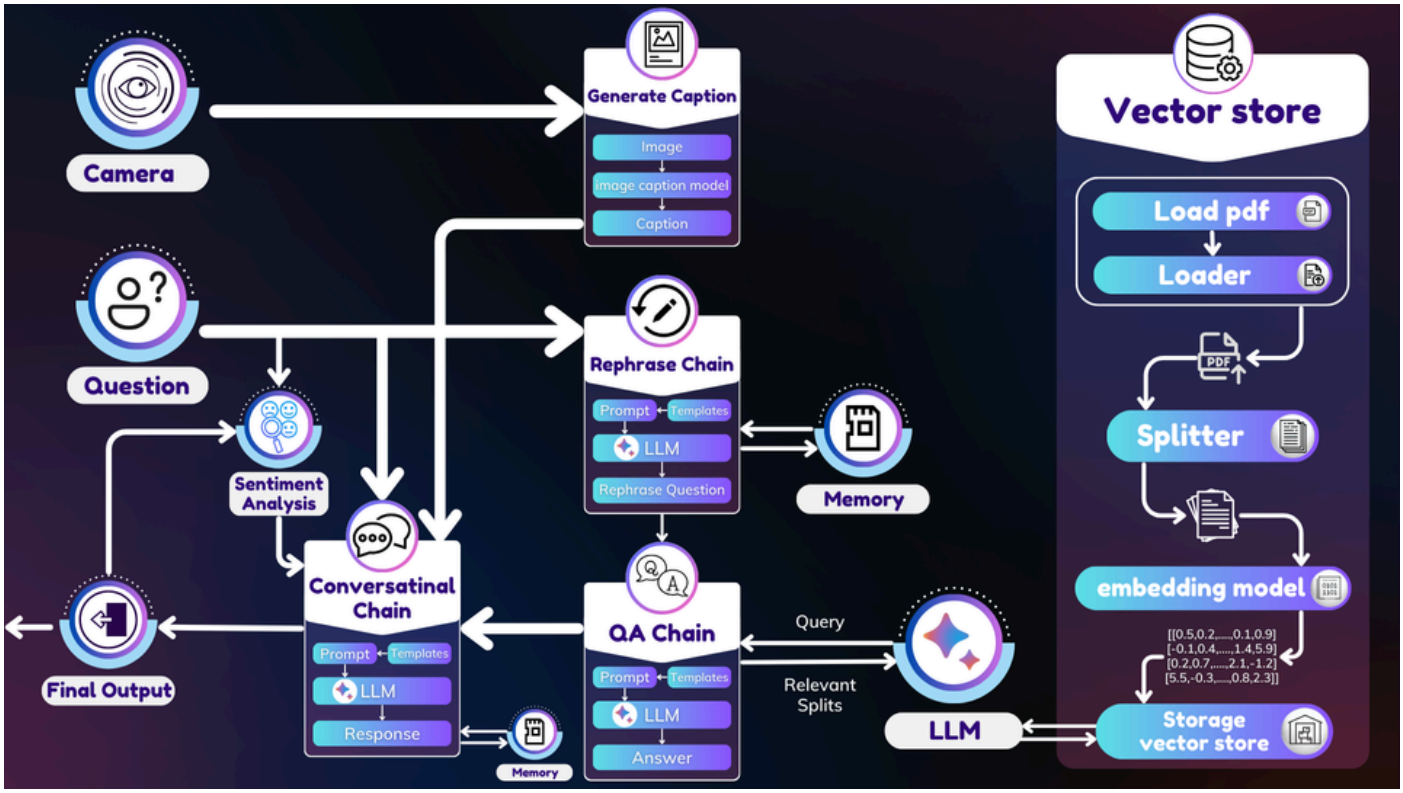
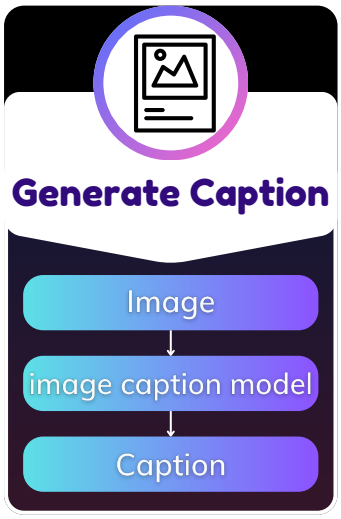


Figure 26. System Architecture 1

Workflow Process :

- An image is captured by the Camera and sent to the "Generate Caption" module, which generates a textual description (caption) of the image.
- A user question is input and analyzed for sentiment.
- The question, along with sentiment analysis results, is processed by the Conversational Chain, which uses an LLM to generate a response.
- The question is also sent to the Rephrase Chain, which uses an LLM to rephrase the question and store it in Memory.
- The QA Chain retrieves relevant information from the Vector Store using the rephrased question and generates an answer using an LLM.
- Information for the Vector Store is loaded from PDFs, split into parts, embedded into vectors, and stored for efficient retrieval.
- The final response is provided to the user as the Final Output.

Generate Caption :



The part of the flowchart titled "Generate Caption" represents the process of creating a textual description (caption) for an image.

Components :

1. Image : The input to this process is an image frame of visual content that is inserted by captured pictures with the camera.

2. Image Caption Model :

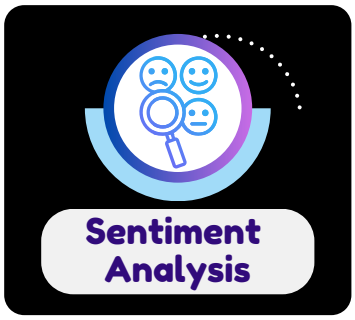
- The image is fed into the image caption model.
- The model processes the image, typically extracting features and patterns to understand its content.
- The model then uses these features to generate a natural language description.

3. Caption : The output of the image caption model is a caption, which is a textual description of the image.

4. Content : The caption describes the elements and actions in the image, aiming to provide a human-understandable summary of what is depicted.

Sentiment Analysis

The "Sentiment Analysis" part of the flowchart involves determining the emotional tone or sentiment of the input text, which is a user-provided question in this context. Here is a detailed explanation of each aspect of this component :



Detailed Process Flow:

1. Receive Question :

- The user inputs a question, which is then sent to the sentiment analysis module for evaluation.

2. Analyze Sentiment :

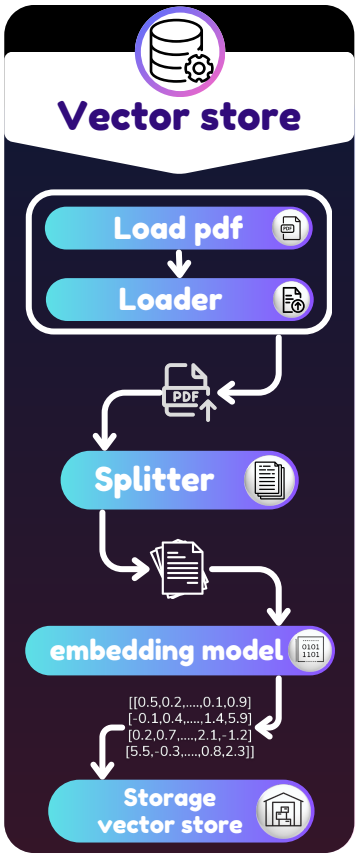
- The sentiment analysis module processes the text of the question and evaluates the emotional tone of the text. This involves identifying whether the sentiment expressed in the text is positive, negative, neutral, or another specific emotion (e.g., joy, anger, sadness).
- It breaks down the text into tokens (words or phrases) and uses its algorithms to assign sentiment scores or categories to the text.
- The analysis might involve several steps, such as tokenization, part-of-speech tagging, syntactic parsing, and applying sentiment lexicons or trained models.

3. Output :

The result of this analysis is an indication of the overall sentiment of the question. This result will be like this:

- **Emotional Categories :** More granular emotions like joy, sad, anger, etc.

User Interface (UI) that allows users to upload a PDF file



1. Load the PDF :
Use a PDF parser compatible with LangChain to load the PDF document. You can use PyPDFLoader for this purpose. we use PyPDFLoader for this purpose.

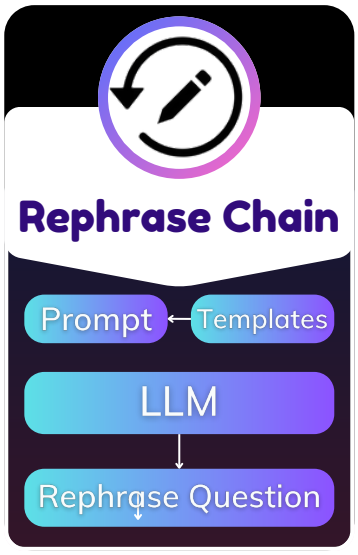
2. Split the Document :
Once the PDF is loaded, split the document into smaller chunks or pages, which will be processed individually.

3. Embedding Model :
Use an embedding model to convert the text data from each chunk into numerical vectors. This step is crucial as it transforms the text into a format that can be processed by machine learning models.

4. Vector Store :
Store these embeddings in a vector database. The vector database allows for efficient retrieval based on similarity metrics. we useChroma for this purpose.

5. Retrieval and Prompting :
When you need to retrieve information, the RAG system will use the stored vectors to find the most relevant information based on the input query.

Rephrase Chain



Input : It analyzes the user's question within
the context of previous interactions stored in the chat history.

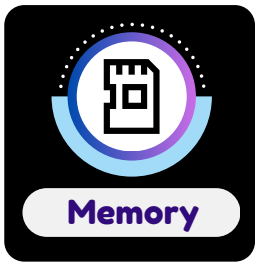
Components inside the "Rephrase Chain" block :

- **Prompt and Templates :**
  Likely used to generate prompts for the language model.
- **LLM (Large Language Model) :**
  Used for rephrasing the question.

Output: Rephrase Question: The output of the rephrasing process

Memory

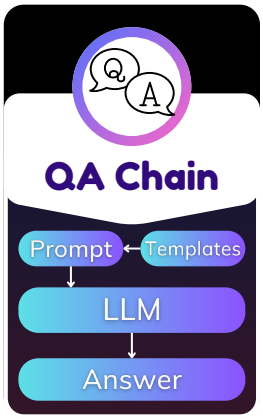
It analyzes the user's question within the context of previous interactions stored in the chat history



QA Chain

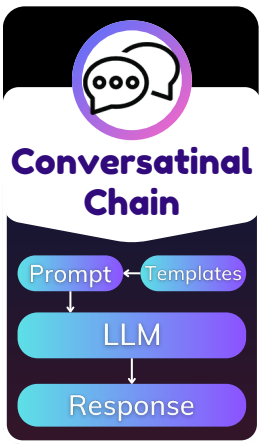
Purpose : This section retrieves relevant context based on the rephrased question using a template (QA_template()), which defines how the context should be incorporated into the response.

1. A Rephrase Question is input into the QA Chain.
2. The LLM processes the Rephrase Question and interacts with the storage vector store to find relevant information.
3. The QA Chain uses prompts and templates to format the relevant information and sends it to the LLM.
4. the answer is then provided as the output of the QA Chain.



Conversation Chain

Invokes the ConversationChain with the user's input (question) to generate a response based on the provided conversation context and memory.



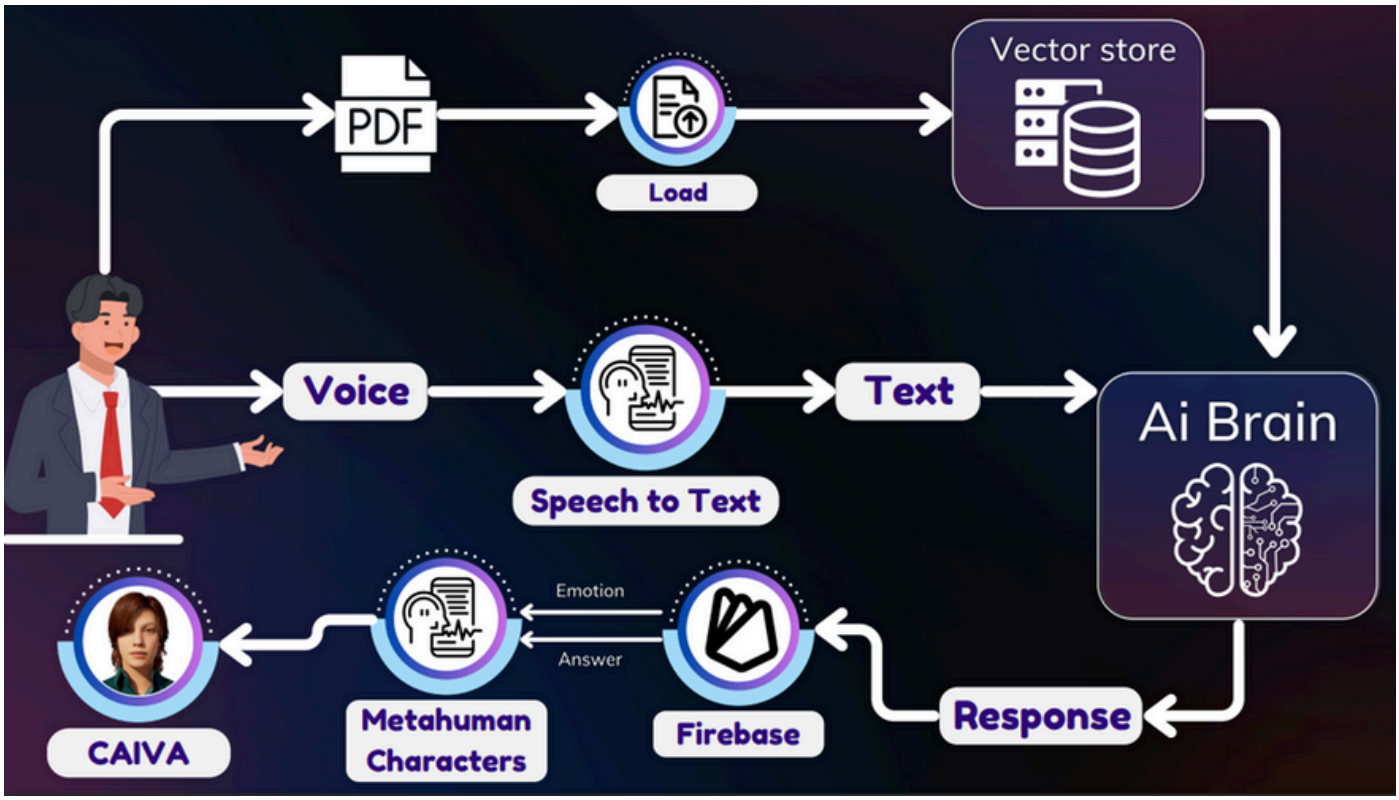


Figure 27. System Architecture 2

Workflow Process :

- **Voice :** A user provides a voice input, This voice input is the initial interaction point for the system.
- **Speech to Text :** The voice input is converted to text using a Speech-to-Text module.
- **Text :** The transcribed text is then used as the input for further processing.
- **PDF :** Represents documents that can be loaded into the system.
- **Load :** PDFs are loaded into the system for processing, This indicates that the system can also handle document inputs, potentially extracting and using information from PDFs.
- **Vector Store :** A storage system for handling processed data, Stores information in vectorized form, which is efficient for quick retrieval and processing by AI models, enhancing the AI Brain's ability to provide informed responses.
- **AI Brain :** The core processing unit of the system, referred to as the "AI Brain."
- **Response :** The AI Brain generates a response based on the analysis of the input text, This response is the output of the AI Brain, which will be conveyed back to the user.
- **Firebase :** A platform for managing data and backend processes.
- **Metahuman Characters :**
  High-fidelity virtual characters that can provide responses to users.
- **CAIVA (Character AI Virtual Assistant) :**
  Represents a virtual assistant character that interacts with users.

Components :

The part of the flowchart titled "Generate Caption" represents the process of creating a textual description (caption) for an image.

1. Voice :

- **Description :** The starting point where the user provides input via spoken language.
- **Function :** Captures the user's voice input using a microphone or other audio input device.

2. Speech to Text

- **Description :** Converts the captured voice input into written text.
- **Function :** Transcribes spoken words into text that can be processed by textual AI models.
- **Technology :** Employs speech recognition technology, Google's Speech-to-Text API.

Steps :

1. Audio Processing : Filters and processes the audio signal to improve recognition accuracy.
2. Recognition Engine : Uses machine learning models to map audio patterns to words.
3.Text Output : Produces the written text version of the spoken input.

3. Text

- **Description :** The transcribed text from the Speech to Text component.
- **Function :** Acts as the intermediary representation of the user's voice input, now in a format that can be further processed by the AI system.
- **Technology :** Simple text handling and preprocessing techniques.

4. PDF

- **Description :** Represents documents that can be input into the system.
- **Function :** Provides additional data sources for the AI Brain to use, enhancing its knowledge base and response quality.
- **Technology :** Document processing technologies to load and read PDFs, such as PDF parsing libraries (e.g., PyPDF2, PDF.js).

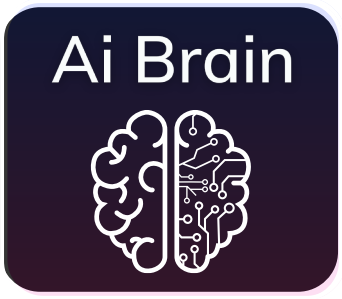
5. Load

- **Description :** The process of loading PDF documents into the system.
- **Function :** Imports documents and prepares them for processing.
- **Technology :** Involves parsing the content of PDFs and converting them into a format that can be processed by text analysis models.

6. Vector Store

- **Description :** A storage system that keeps processed information in vectorized form.
- **Function :** Stores textual data as vectors for efficient retrieval and similarity searches.
- **Technology :** Embedding models (e.g., Word2Vec, GloVe, BERT) and vector databases (e.g., FAISS, Annoy).
- **Process :**
  1. Embedding: Converts text data into high-dimensional vectors.
  2. Storage: Saves these vectors in a database designed for fast retrieval.

7. AI Brain



- **Description :** The core processing unit that handles text understanding and response generation.
- **Function :** Analyzes the text input, understands the context and intent, and generates an appropriate response.
- **Technology :** Advanced NLP models and AI techniques.

Components :

1. Natural Language Understanding (NLU) :
Parses and understands the input text.
2. Dialogue Management :
Maintains the context of the conversation.
3. Response Generation :
Uses models like GPT-4, BERT, or other LLMs to generate a coherent response.

8. Response

- **Description :** The output text generated by the AI Brain.
- **Function :** Serves as the system's reply to the user's input, ready to be conveyed back to the user.
- **Technology :** Managed by the response generation algorithms within the AI Brain.

9. Firebase

- **Description :** A platform for managing backend services and data.
- **Function :** Handles data storage, real-time database operations, and backend logic for the system.
- **Technology :** Google Firebase platform, providing services like Firestore, Realtime Database, Authentication, and Hosting.
- **Integration :** Interfaces with Metahuman Characters to provide necessary data for responses and emotional cues.

10. Metahuman Characters

- **Description :** High-fidelity, realistic virtual characters.
- **Function :** Deliver responses to users in a lifelike manner, enhancing user engagement.
- **Technology :** Uses advanced 3D modeling and animation tools (e.g., Unreal Engine's MetaHuman Creator).
- **Features :**

  1.Animation : Real-time rendering and animation of characters.

  2.Speech Synthesis : Converts text responses to lifelike spoken language, often using TTS (Text-to-Speech) technology.

11. CAIVA (Character AI Virtual Assistant)

- **Description :** A virtual assistant that interacts with users.
- **Function :** Provides a user interface through which users can interact with the AI system, often represented by an avatar.
- **Technology :** MetaHuman Creator.

7.5. User Interface Prototype

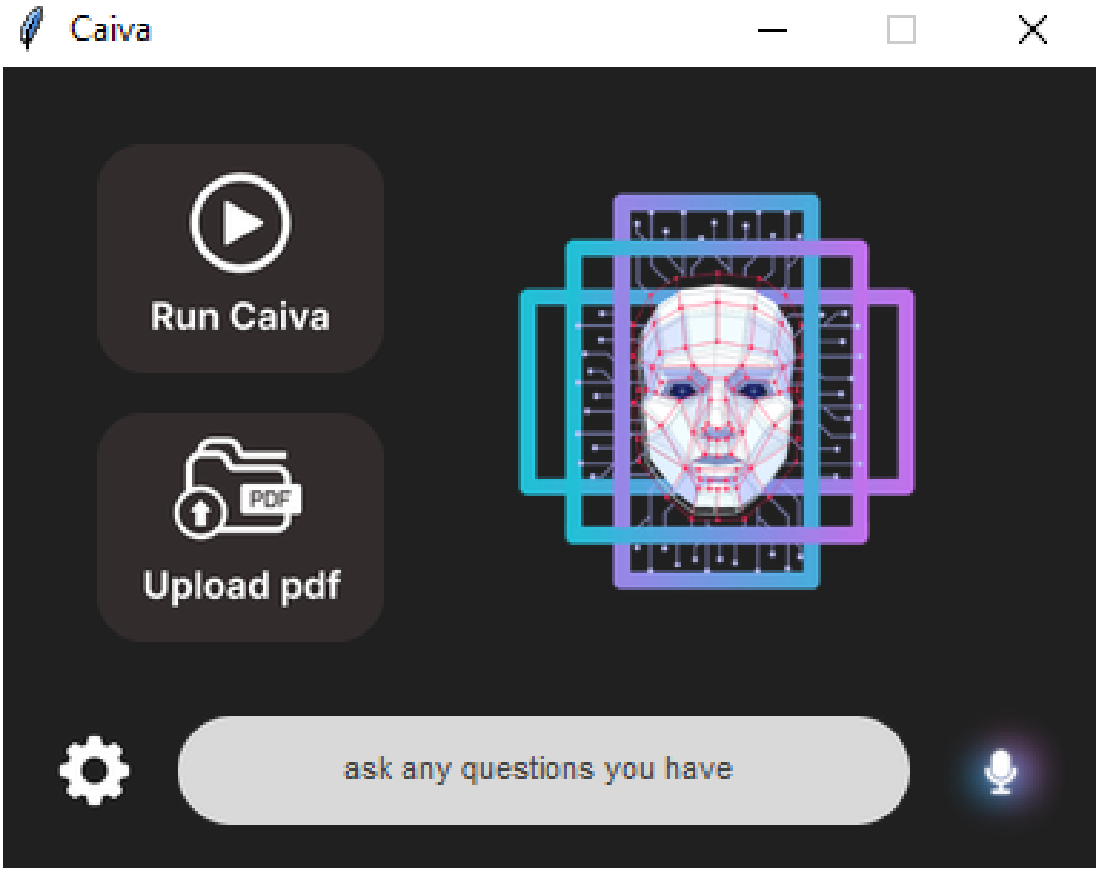


Figure 28. User Interface Prototype.1

Figure 29. User Interface Prototype.2

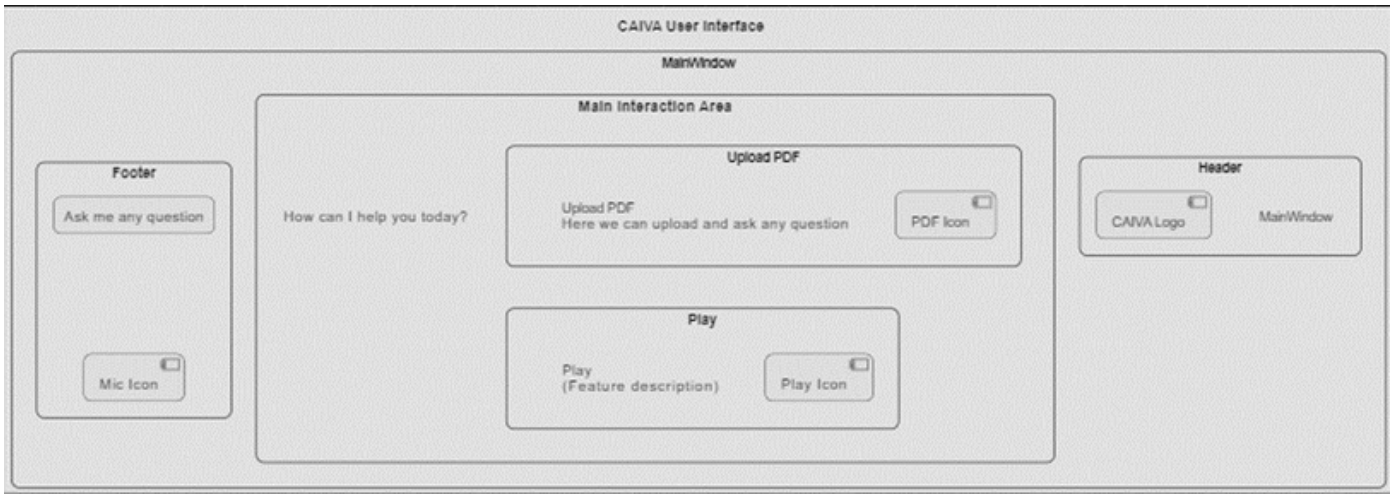


Figure 30. Caiva App UI

- **Main Window :**
  Contains the overall structure of the application window.

- **CAIVA Logo :**
  Represents the logo at the top.

- **Greeting Text :**
  "How can I help you today?" text below the logo.

- **Upload PDF Button :**
  Contains the PDF icon and description text.

- **Play Button :**
  Contains the play icon.

- **Question Input :**
  The text input field for asking questions.

- **Mic Icon :**
  The microphone button for voice input.

Chapter 8

System Testing

System Testing Unreal Engine for Animating MetaHuman Characters :

Testing the animation capabilities of Unreal Engine for MetaHuman characters is crucial to ensure the quality and performance of our caiva. We employ a comprehensive testing approach to verify that animations are realistic, lifelike, and responsive, enhancing the overall user experience.

Test Strategies and Methodologies :

We utilize a combination of manual and automated testing techniques to evaluate the animation features of Unreal Engine. Manual testing involves interactive exploration of animation sequences and face controls of the metahuman, assessing factors such as character facial expressions, and lip-syncing accuracy. Automated testing utilizes scripts and tools to perform repetitive tests, ensuring consistency and reliability of caiva reactions across different scenarios.

Organizing and Executing Tests :

Tests are categorized according to different types of facial expressions, such as happy, sad, angry, and surprise. Each test case is meticulously outlined with detailed steps, expected outcomes, and success criteria tailored to verify that the animation accurately reflects the intended expression for the given text or scenario. The testing process follows a systematic approach, ensuring thorough evaluation of each facial expression. Results are carefully documented for analysis and to guide further improvements in animation quality and performance.

Additionally, we utilized built-in features like the Animation Blueprint editor, which enables visual scripting of character animations, enhancing our ability to create and refine complex motion sequences. Furthermore, external tools such as profiling and debugging utilities were instrumental in identifying and resolving performance bottlenecks and animation glitches, ensuring the reliability and quality of our animations.

Our primary objective was to ensure that the facial animations of the character accurately represent the text she speaks and appear as realistic and human-like as possible. By implementing rigorous testing strategies and utilizing suitable tools, we guarantee that Unreal Engine's animation capabilities align with the requirements of our character animation.

8.1. Unit Testing

Our objective is to create a deeply engaging and emotionally resonant experience for users, one where interactions feel personal and meaningful.

To achieve this, we meticulously test each component of our animation system in isolation, focusing on the human-like qualities of our caiva's expressions and the accuracy of the accompanying text.

In the first component of unit testing, we focused on the facial expressions of our caiva, ensuring they felt genuine and lifelike for emotions like happiness, sadness, anger, and surprise.

The Geppetto plugin played a key role here, allowing us to human expressions and incorporate them into our character animations. This process made caiva's expressions more believable and immersive for users. Following this, we individually tested each expression record on the character, making manual adjustments to her facial controls to ensure they looked just right. This hands-on approach helped us create more expressive and realistic character animations.

Another crucial component we tested was the timing of response delivery within the Unreal Engine environment and its equivalent emotion. This involved evaluating the delay between the user input and the character's verbal response, as well as ensuring that the appropriate facial expression synchronized seamlessly with the spoken words. By meticulously testing this timing, we aimed to ensure that the character's responses felt natural and synchronized, enhancing the overall realism and immersion of the interaction for users.
we also test our character's visual identification ability. When prompted by the user, our character can identify and describe objects in her surrounding environment. This involves rigorous testing to ensure that the character accurately recognizes and describes objects, further enriching the user experience.

By conducting individual tests for each component, we acquire valuable insights into their performance and effectiveness allowing us to meticulously assess each component's functionality and ensure that they meet our standards of quality and performance and we've been able to refine our animations and interactions, creating a more immersive and engaging experience for users.

As caiva integrated in a desktop application :

unit testing of the application components

Book Input Component Testing :

- Test the functionality of the application to accept and process book input files from the user's computer.
- Verify that the selected book is successfully loaded into the application and accessible for further processing.

Speech-to-Text Interaction Testing :

- Test the speech-to-text conversion functionality to ensure accurate transcription of user-spoken questions.
- Validate that the converted text accurately reflects the user's spoken queries without loss of context or meaning.

Response Handling Component Testing :

- Test the response handling logic to confirm the appropriate routing of user questions based on relevance to the book.
- Verify that unrelated questions are directed to the LLM model for response generation, while book-related queries are passed to the LangChain model for retrieval from the book.

Vision System Testing :

- Test the vision system's functionality to detect and interpret visual cues from the user's environment.
- Validate that the vision system accurately identifies objects or scenes mentioned in user questions for appropriate response generation.

Sentiment Analysis Component Testing :

- Test the sentiment analysis model integration to ensure correct classification of emotional tones in user questions and generated responses.
- Verify that the sentiment analysis accurately captures the emotional context of both questions and responses.

Firebase Integration Testing :

- Test the integration with Firebase to confirm successful transmission of classified emotions, user questions, and generated responses.
- Validate that data sent to Firebase is accurately formatted and accessible for further processing.

Rendering in Unreal Engine Testing :

- Test the rendering functionality in Unreal Engine to ensure proper display of Caiva's emotions based on classified sentiments.
- Validate that Caiva's appearance and facial expressions align with the detected emotions for an immersive user experience.

Speech Synthesis Component Testing :

- Test the speech synthesis functionality to confirm accurate conversion of generated responses into spoken dialogue.
- Verify that the speech synthesis process is smooth and natural, enhancing Caiva's verbal interactions with users.

8.2. Integration and Regression Testing

Integration and regression testing mark critical phases in our development process, ensuring the seamless integration of individual components into a cohesive human like assistant within Unreal Engine.
During this phase, we assess how each component interacts within the integrated system and identify any issues or unexpected behavior that may arise.

Components are meticulously tested as they are integrated into the animation system.
This involves combining different facial animation blueprint components, including facial expressions and text interpretation modules, to evaluate their functionality in unison.
We closely monitor the integration process, observing how the components interact and identifying any discrepancies or inconsistencies in their behavior.

For instance, adjustments may be required to synchronize the speaking speed of the character and the timing of transmitting the text response output from the chatbot to the text-to-voice tool. Additionally, errors such as the interception of lip-sync and the expression of the rest of the face will need a way to balance and blend playing in the same frame. Each identified issue undergoes thorough scrutiny to ascertain its underlying cause and assess its impact on the overall system functionality.

Regression testing is conducted to ensure that changes or updates to the system do not inadvertently introduce new issues or regressions.

This involves retesting previously integrated components to verify that they continue to function as expected after modifications have been made.

Any regressions or unexpected behavior identified during regression testing are addressed promptly to maintain the stability and reliability of the animation system.

By conducting thorough integration and regression testing, we validate the integrity of our facial animation system and ensure that it delivers unique and personal experience for users. Through diligent testing and refinement, we address any issues or unexpected behavior encountered, ensuring that the final product meets the highest standards of quality and reliability.

8.3. User Acceptance Testing

User acceptance testing is a pivotal step in ensuring that our character is ready for use and provides users with a warm and human-like experience in whatever system she will be integrated into. Our primary objective is to create interactions that feel natural, akin to chatting with a real person who can understand emotions and overall context and respond accordingly.

During this essential testing phase, users, including customers and stakeholders, actively engage with the system to evaluate its adherence to initial requirements and its effectiveness in delivering the desired experience. They immerse themselves in conversations with the character, assessing its ability to understand emotions , maintains awareness of the conversation's context , identify objects in her environment visually and respond in a manner that feels genuine and relatable.

To ensure comprehensive feedback, we involve individuals from diverse backgrounds and perspectives. Their input helps us gauge various aspects of the system, such as the flow of dialogue, the character's emotional responsiveness, and overall usability.
The feedback collected during user acceptance testing is invaluable for refining and enhancing the system.

By listening to users' voices, we can make adjustments to ensure that the system fosters genuine connections, resulting in more engaging and satisfying conversations.

Through user acceptance testing, we not only evaluate the technical functionality of the system but also confirm its ability to fulfill our overarching goal of providing users with a personal, relatable, and human-like interaction experience.

8.4. Test cases

Functionality Test Case : Caiva's Seamless Responsiveness

- **Description :** This test case aims to evaluate the software interface's functionality in seamlessly interacting with the rest of the system and its users.
- **Test Condition :** The software is ready for use, letting users talk to Caiva in conversations that mimic human communication, including both verbal and non-verbal expressions.
- **Testing Approach :** Black box testing will be employed to evaluate the interface externally, focusing on user interactions and system responses.

Test Steps :

- Launch the software and navigate to the main interface.
- Initiate a voice conversation with Caiva.
- Observe Caiva's facial expressions and verbal responses in real-time.
- Interact with the interface and assess Caiva's reactions.
- ask about the visual elements in your environment.

Test Input :

- Voice input.
- Visual environment cues.

- **Test Expected Result :**
  - Caiva responds promptly to user inputs with vocal responses.
  - Caiva accurately recognizes emotional tones and responds contextually with appropriate facial expressions and verbal cues.
  - Caiva remains responsive and stable throughout the interaction.
  - Caiva can identify and comment on visual objects in the environment.

- **Actual Result :** real time responds and synchronic face expression and voice with lip-sync animation but unstable responses about her visual perception.

- **Status :** fail

- **Remarks :** her taking and saving shots every second of the environment is unpractical and needs more effective alternative.

Unit Test Case : Visual Identification

- **Description :** This test case aims to verify just caiva's ability to visually identify elements within the environment when asked by the user's voice.
- **Test Condition :** The MetaHuman character's visual identification feature is operational and integrated into the system.

Test Steps :

- Provide a visual scene to the character through the camera.
- ask about the surrounding environment or what she sees using voice commands.
- Observe the character's identification of objects within the scene.
- Record the character's responses to each identified object.

Test Input :

Voice prompt from the user to caiva, visual scene captured by the camera.

- **Test Expected Result :**
  - caiva accurately identifies objects within the scene.
  - Each identified object is associated with the correct label or description.
  - The identification process is prompt and reliable.
- **Actual Result :** unstable visual identification of the visual environment
- **Status :** fail
- **Remarks :** change the answer about her visual scene due to the identification of the visual environment takes a shot every 5 second which make caiva answer not stable.

Performance Test Case : Virtual Assistant Desktop Application

- **Description :** The performance test case evaluates the response time and overall effectiveness of the virtual assistant desktop application, ensuring it can handle real-world expectations.
- **Test Condition :** The virtual assistant application is operational and accessible for performance testing in a controlled environment.

Test Steps :
Initiate typical user interactions with Caiva, including :

- Providing input book files
- Speaking commands or questions to Caiva
- Observe Caiva's ability to accurately convert spoken commands or questions into text format for further processing.
- Determine the relevance of user queries to the input book, directing unrelated questions to the LLM model for response generation and book-related inquiries to the LangChain model for retrieval from the input book.
- Assess Caiva's response capabilities, ensuring synchronized and accurate emotion display in conjunction with generated responses.

Test Expected Result :

We expect Caiva to quickly understand voice commands and accurately convert them into text for responses. The app should recognize if questions are about the book or not.
For unrelated questions, it should use the LLM model, and for book-related queries, the LangChain model for retrieval from the book.
Caiva's responses should be in sync with appropriate emotions, ensuring a smooth realistic human user experience. This ensures the app performs well in different situations.
Additionally, Caiva should proficiently identify visual objects from her surroundings and provide descriptions or discussions when prompted.
This functionality enhances Caiva's ability to engage with users in a more interactive manner, contributing to a richer user experience.

- **Status :** Success

Chapter 9

Conclusion & Future Work

9.1. Conclusion

This project has led to the creation of Caiva, a next-generation virtual assistant that surpasses existing solutions like Alexa, Siri, and Google Assistant. By integrating advanced emotional recognition, non-verbal communication, and contextual awareness, Caiva offers a more personalized and human-like interaction experience. Her ability to interpret PDF documents adds another layer of utility, making her responses more accurate and context-specific.

These enhancements not only improve user engagement but also represent significant advancements in AI technology, paving the way for more empathetic and intelligent virtual assistants. Caiva stands out in various applications, from customer service to personal assistance, due to her ability to adapt to users' emotions and environments, creating a more responsive and intuitive user experience. This makes her an ideal candidate for roles requiring nuanced human interaction, setting a new standard for virtual assistant capabilities.

9.2. Future Work

Enhanced Emotional Intelligence

To further elevate Caiva's emotional intelligence, future developments should focus on refining her ability to detect and respond to a broader range of emotional cues. This could involve :

1. Advanced Sentiment Analysis : Integrating more sophisticated sentiment analysis algorithms to better understand subtle emotional nuances in user interactions.
2. Multimodal Emotion Recognition : Combining audio, text, and visual data to enhance Caiva's ability to detect emotions more accurately.

Expanded Non-Verbal Communication

Enhancing Caiva's non-verbal communication capabilities can improve user engagement and interaction quality. Future work in this area could include :

1. Improved Facial Expressions : Developing more realistic and varied facial expressions using advanced animation techniques and AI models.
2. Body Language : Incorporating body language recognition and response to make Caiva's interactions even more lifelike.

Contextual and Environmental Awareness

To make Caiva more contextually aware, future research could focus on :

1. Enhanced Object Recognition : Improving object recognition algorithms to allow Caiva to better understand and interact with her surroundings.
2. Context-Aware Responses : Developing algorithms that enable Caiva to provide more contextually relevant responses based on the user's environment and situation.

Knowledge Base Expansion

Expanding Caiva's knowledge base and improving her ability to access and utilize external data sources can significantly enhance her usefulness. Future directions in this area might include :

1. Dynamic Knowledge Integration : Enabling Caiva to access and integrate real-time data from various online sources to provide up-to-date information.
2. Document Parsing and Understanding : Enhancing Caiva's ability to parse and understand complex documents, including scientific papers, legal documents, and technical manuals.

Personalization and User Adaptation

Making Caiva more adaptive to individual user preferences and behaviors can further personalize the user experience. Future work could focus on :

1. User Profiling : Developing sophisticated user profiling algorithms to better understand and predict individual user preferences and needs.
2. Adaptive Learning : Implementing machine learning techniques that allow Caiva to continuously learn from user interactions and improve over time.

Multilingual and Cross-Cultural Capabilities

To make Caiva accessible to a global audience, future research should focus on :

1. Multilingual Support : Expanding Caiva's language capabilities to include more languages and dialects.
2. Cross-Cultural Sensitivity : Developing algorithms that enable Caiva to understand and respect cultural nuances and differences in communication styles.

Integration with IoT and Smart Home Devices

Integrating Caiva with Internet of Things (IoT) devices and smart home systems can enhance her functionality and convenience. Future work in this area could include :

1. Seamless Device Integration : Enabling Caiva to communicate and control a wider range of IoT devices and smart home systems.
2. Smart Home Management : Developing capabilities that allow Caiva to manage and optimize home environments based on user preferences and behaviors.

By addressing these areas, Caiva can continue to evolve, offering even more advanced, personalized, and human-like interactions that meet the diverse needs of users around the world.

References

1. Astra Google Project :

- **1.1.** David Nield, "What is Project Astra: Google's Futuristic Universal Assistant Explained.", 20 May 2024,
  URL : https://www.techradar.com/computing/artificial-intelligence/what-is-project-astra-googles-futuristic-universal-assistant-explained

2. Chatgpt-4o :

- **2.1.** OpenAI, "Hello GPT-4o.", May 13, 2024,
  URL : https://openai.com/index/hello-gpt-4o/

3. Datasets :

- **3.1.** Google Research, GoEmotions: A dataset of fine-grained emotions, 2020,
  URL : https://research.google/pubs/goemotions-a-dataset-of-fine-grained-emotions/
- **3.2.** Pandey, P., Emotion dataset, Kaggle, n.d.,
  URL : https://www.kaggle.com/datasets/parulpandey/emotion-dataset
- **3.3.** Pashupati Gupta, Emotion dataset, Kaggle, n.d.,
  URL : https://www.kaggle.com/datasets/042977506d4b87fe2ce6998514bd60df9ae2bdde98acf973acfd8 7e758e50d68
- **3.4.** Juyal, I., Emotions in text, Kaggle, n.d.,
  URL : https://www.kaggle.com/datasets/ishantjuyal/emotions-in-text
- **3.5.** Wagih, A., Emotion dataset, Kaggle, n.d.,
  URL : https://www.kaggle.com/datasets/abdallahwagih/emotion-dataset

4. LSTM :

- **4.1.** Gourav Singh, Understanding Architecture of LSTM, Analytics Vidhya, 04 Jun, 2024,
  URL : https://www.analyticsvidhya.com/blog/2021/01/understanding-architecture-of-lstm/

5. CNN :

- **5.1.** Shiyang Liaoa, Junbo Wangb, Ruiyun Yua, Koichi Satob and Zixue Chengb, CNN for situations understanding based on sentiment analysis of twitter data, Advances in Information Technology, IAIT2016, 19-22 , Published by Elsevier B.V. , Macau, China, December 2016,
  URL : https://www.sciencedirect.com/science/article/pii/S1877050917312103
- **5.2.** Maryem Rhanoui ,Mounia Mikram ,Siham Yousfi and Soukaina Barzali , A CNN-BiLSTM Model for Document-Level Sentiment Analysis, MDPI, 25 July 2019,
  URL : https://www.mdpi.com/2504-4990/1/3/48

6. Attention Mechanisms :

- **6.1.** Jashmi, K. R., & Sangam, P., Examining Attention Mechanisms in Deep Learning Models for Sentiment Analysis, MDPI, 2021,
  URL : https://www.mdpi.com/2076-3417/11/9/3883#B22-applsci 11-03883
- **6.2.** Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I., Attention is All You Need, Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.,
  URL : https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

7. RoBERTa Model :

- **7.1.** Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... & Stoyanov, V., RoBERTa: A Robustly Optimized BERT Pretraining Approach, arXiv:1907.11692v1 [cs.CL] 26 Jul 2019,
  URL : https://arxiv.org/abs/1907.11692
- **7.2.** Hugging Face, RoBERTa, n.d.,
  URL : https://huggingface.co/docs/transformers/model_doc/roberta

8. PaLM 2 :

- **8.1.** Google AI, PaLM 2, n.d.,
  URL : https://ai.google/discover/palm2/
- **8.2.** Anil, R., Brown, T. B., Chen, B., Cheng, H. T., Chowdhery, A., Clark, J., ... & Ramesh, A., PaLM 2 Technical Report, arXiv:2305.10403v3 [cs.CL] 13 Sep 2023,
  URL : https://arxiv.org/abs/2305.10403

9. Gemini 1.5 Flash :

- **9.1.** DeepMind, Gemini 1.5 Flash, n.d.,
  URL : https://deepmind.google/technologies/gemini/flash/
- **9.2.** Google Blog, Google Gemini: Next-Generation Model, February 2024,
  URL : https://blog.google/technology/ai/google-gemini-next-generation-model-february-2024/#gemini-15
- **9.3.** Ritvik19, Papers Explained 142: Gemini 1.5 Flash, Medium, March 2023,
  URL : https://ritvik19.medium.com/papers-explained-142-gemini-1-5-flash-415e2dc6a989

10. Speech Recognition :

- **10.1.** Uberi, SpeechRecognition, PyPI, n.d.,
  URL : https://pypi.org/project/SpeechRecognition/
- **10.2.** Uberi, SpeechRecognition GitHub repository, n.d.,
  URL : https://github.com/Uberi/speech_recognition/tree/master
- **10.3.** Nishit Kamdar, "Measuring and Improving Speech-to-Text Accuracy." Google Cloud, Mar 9, 2022.
  URL : https://medium.com/google-cloud/measuring-and-improving-speech-to-text-accuracy-google-cloud-platform-eba62c50b8ac

11. LangChain :

- **11.1.** LangChain AI. (n.d.). Introduction to LangChain. Retrieved from :
  URL : https://python.langchain.com/docs/get_started/introduction
- **11.2.** LangChain AI, LangChain GitHub repository, n.d.,
  URL : https://github.com/langchain-ai/langchain
- **11.3.** Keivalya Pandya, Mehfuza Holia, Automating Customer Service using LangChain, arXiv:2310.05421 [cs.CL] 9 Oct 2023,
  URL : https://arxiv.org/abs/2310.05421

12. BLIP (Bootstrapping Language-Image Pre-training) :

- **12.1.** Salesforce, BLIP, n.d.,
  URL : https://huggingface.co/docs/transformers/model_doc/blip
- **12.2.** Salesforce, BLIP GitHub repository, n.d.,
  URL : https://github.com/salesforce/BLIP
- **12.3.** Li, J., Selvaraju, R. R., Gotmare, A., Joty, S., Xiong, C., & Hoi, S., BLIP: Bootstrapping Language Image Pre-training, arXiv:2201.12086v2 [cs.CV] 15 Feb 2022,
  URL : https://arxiv.org/abs/2201.12086

13. Epic Games Unreal Engine :

- **13.1.** Epic Games. (n.d.). Epic Developer Community. Retrieved from :
  URL : https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-3-documentation?application_version=5.3

14. Geppetto Plugin :

- **14.1.** Content Detail :
  URL : https://www.unrealengine.com/marketplace/en-US/product/geppetto
- **14.2.** Geppetto Plugin, Geppetto, Sep 19, 2023,
  URL : https://drive.google.com/file/d/1rJnfBVJo0SUFLj-3WvCVPAjbwZTZuCbh/view

15. Ariel Plugin :

- **15.1.** Geppetto Plugin, Geppetto, Sep 19, 2023,
  URL : https://www.unrealengine.com/marketplace/en-US/product/ariel-voice-generation
- **15.2.** Retrieved from :
  URL : https://drive.google.com/file/d/1gbdVfI2_MDiWdTZiYZJqD08Q1kqnkMH_/edit

16. MetaHuman Creator :

- **16.1.** Epic Games, MetaHuman Documentation, n.d.,
  URL : https://dev.epicgames.com/documentation/en-us/metahuman/metahuman-creator-overview