

*The Development of Jaicat AI Assistant: A Jarvis-Inspired AI System*

Jays Automated Intellectual Command Assistant Technology



01 March 2024

[Company name]

[Company address]

Contents

[**Project Overview** 8](#_Toc178780012)

[Programming Language and Framework 9](#_Toc178780013)

[Warn the user when the battery level reaches 10% 9](#_Toc178780014)

[Features 9](#_Toc178780015)

[Face Recognition 9](#_Toc178780016)

[Voice Interaction 9](#_Toc178780017)

[NLP System (Natural Language Processing) 9](#_Toc178780018)

[NLU (Natural Language Understanding) 9](#_Toc178780019)

[NLG (Natural Language Generation) 9](#_Toc178780020)

[Text Classification Pipeline 9](#_Toc178780021)

[Code Generation System 10](#_Toc178780022)

[Image and Video Processing 10](#_Toc178780023)

[Knowledge Base System 10](#_Toc178780024)

[Web Scraping 10](#_Toc178780025)

[PDF Parsing 10](#_Toc178780026)

[EPUB Parsing 10](#_Toc178780027)

[Image Analysis 10](#_Toc178780028)

[YouTube Analysis 10](#_Toc178780029)

[Knowledge Graph 10](#_Toc178780030)

[Computer Vision System 10](#_Toc178780031)

[Swann CCTV System Integration 10](#_Toc178780032)

[IP Camera System Integration 10](#_Toc178780033)

[USB Camera System Integration 10](#_Toc178780034)

[Phone Camera System Integration 10](#_Toc178780035)

[File Analysis 11](#_Toc178780036)

[Project Management API 11](#_Toc178780037)

[Calendar API 11](#_Toc178780038)

[Google Calendar API 11](#_Toc178780039)

[People Recognition 11](#_Toc178780040)

[Sexual Content Understanding 11](#_Toc178780041)

[Politics Understanding 11](#_Toc178780042)

[Social Media Monitoring 11](#_Toc178780043)

[Social Media Post Generation 11](#_Toc178780044)

[Weather API Integration 11](#_Toc178780045)

[Engineering Understanding 11](#_Toc178780046)

[Motorbike and Engine Understanding 11](#_Toc178780047)

[Contact Recognition 11](#_Toc178780048)

[Car Model Recognition 11](#_Toc178780049)

[Home Surveillance System 12](#_Toc178780050)

[Medical Analysis 12](#_Toc178780051)

[Crime Analysis 12](#_Toc178780052)

[Fitness Analysis 12](#_Toc178780053)

[Travel Planning 12](#_Toc178780054)

[Personal Finance Management 12](#_Toc178780055)

[Language Translation 12](#_Toc178780056)

[Job Search Assistance 12](#_Toc178780057)

[Education and Learning Assistance 12](#_Toc178780058)

[Health and Wellness Tracking 12](#_Toc178780059)

[Smart Home Automation 12](#_Toc178780060)

[Travel Recommendations 12](#_Toc178780061)

[Food and Recipe Suggestions 12](#_Toc178780062)

[News Updates 12](#_Toc178780063)

[Conversation Data Saving 13](#_Toc178780064)

[MySQL Data Handling 13](#_Toc178780065)

[Project Files 14](#_Toc178780066)

[Code Files 18](#_Toc178780067)

[main.py 18](#_Toc178780068)

[Key Functions: 18](#_Toc178780069)

[command 21](#_Toc178780070)

[command\Commands.py 21](#_Toc178780071)

[Key Functions: 21](#_Toc178780072)

[command\Commands.txt 22](#_Toc178780073)

[computer\_vision 22](#_Toc178780074)

[computer\_vision\car\_model.py 22](#_Toc178780075)

[computer\_vision\contact\_recognition.py 22](#_Toc178780076)

[computer\_vision\home\_surveillance.py 23](#_Toc178780077)

[computer\_vision\opencv\_utils.py 23](#_Toc178780078)

[computer\_vision\people\_recognition.py 25](#_Toc178780079)

[config 26](#_Toc178780080)

[config\api\_keys.py 26](#_Toc178780081)

[config\config.json 26](#_Toc178780082)

[conversation 27](#_Toc178780083)

[conversation\\_\_pycache\_\_ 27](#_Toc178780084)

[conversation\dialogue\_manager copy.py 27](#_Toc178780085)

[conversation\dialogue\_manager.py 30](#_Toc178780086)

[conversation\nlg.py 33](#_Toc178780087)

[conversation\nlu.py 35](#_Toc178780088)

[data 36](#_Toc178780089)

[data\calendar\_data.csv 36](#_Toc178780090)

[data\weather\_data.csv 36](#_Toc178780091)

[database 37](#_Toc178780092)

[database\conversation.py 37](#_Toc178780093)

[database\enginnering\_mod.py 37](#_Toc178780094)

[database\knowledge\_base.py 37](#_Toc178780095)

[database\knowledge\_graph.py 38](#_Toc178780096)

[database\motorbikes.py 38](#_Toc178780097)

[database\mysql\_utils.py 38](#_Toc178780098)

[database\Nas.py 38](#_Toc178780099)

[database\web\_scraping.py 38](#_Toc178780100)

[Documentation 39](#_Toc178780101)

[Documentation\documentation.py 39](#_Toc178780102)

[Documentation\README.md 39](#_Toc178780103)

[enrollment\_json 39](#_Toc178780104)

[enrollment\_json\Jay.json 39](#_Toc178780105)

[enrollment\_json\Samantha.json 39](#_Toc178780106)

[enrollment\_pictures 40](#_Toc178780107)

[enrollment\_pictures\Jay.png 40](#_Toc178780108)

[enrollment\_pictures\Samantha.png 40](#_Toc178780109)

[features 40](#_Toc178780110)

[features\\_\_pycache\_\_ 40](#_Toc178780111)

[features\code\_generation.py 40](#_Toc178780112)

[features\dialogue\_manager.py 41](#_Toc178780113)

[features\nlp copy.py 43](#_Toc178780114)

[features\nlp.py 45](#_Toc178780115)

[features\nlu.py 48](#_Toc178780116)

[features\sentiment\_analysis.py 50](#_Toc178780117)

[features\SentimentAnalysis.py 50](#_Toc178780118)

[features\text\_classification.py 51](#_Toc178780119)

[lib 52](#_Toc178780120)

[lib\nlg 52](#_Toc178780121)

[lib\nlp 52](#_Toc178780122)

[machine\_learning 62](#_Toc178780123)

[machine\_learning\Machine\_Learning 63](#_Toc178780124)

[machine\_learning\sklearn\_utils.py 64](#_Toc178780125)

[machine\_learning\tensorflow\_utils.py 65](#_Toc178780126)

[models 65](#_Toc178780127)

[models\\_\_pycache\_\_ 65](#_Toc178780128)

[models\calendar\_model.pkl 65](#_Toc178780129)

[models\code\_generation\_model copy.py 66](#_Toc178780130)

[models\code\_generation\_model.py 66](#_Toc178780131)

[models\intent\_model.pkl 67](#_Toc178780132)

[models\nlp\_model.py 67](#_Toc178780133)

[models\text\_classification\_model.py 68](#_Toc178780134)

[models\weather\_model.pkl 69](#_Toc178780135)

[network 69](#_Toc178780136)

[network\socket\_utils.py 69](#_Toc178780137)

[security 69](#_Toc178780138)

[security\encryption\_utils.py 69](#_Toc178780139)

[services 69](#_Toc178780140)

[services\\_\_pycache\_\_ 69](#_Toc178780141)

[services\calendar\_api.py 69](#_Toc178780142)

[services\calendar\_service.py 72](#_Toc178780143)

[services\contol bluetooth 72](#_Toc178780144)

[services\education.py 73](#_Toc178780145)

[services\Engineering.py 74](#_Toc178780146)

[services\epub\_parsing.py 75](#_Toc178780147)

[services\file\_analysis.py 75](#_Toc178780148)

[services\finance.py 75](#_Toc178780149)

[services\food.py 75](#_Toc178780150)

[services\google\_calendar\_api.py 75](#_Toc178780151)

[services\health.py 76](#_Toc178780152)

[services\image\_analyze 76](#_Toc178780153)

[services\ip\_cam.py 76](#_Toc178780154)

[services\job\_search.py 76](#_Toc178780155)

[services\mental\_health 77](#_Toc178780156)

[services\news\_integration.py 77](#_Toc178780157)

[services\pdf\_parsing.py 78](#_Toc178780158)

[services\phone\_cam.py 78](#_Toc178780159)

[services\Politics 78](#_Toc178780160)

[services\project\_management.py 81](#_Toc178780161)

[services\sexual\_content 81](#_Toc178780162)

[services\smart\_home.py 86](#_Toc178780163)

[services\Social\_media\_monitor.py 86](#_Toc178780164)

[services\spotify\_integration.py 88](#_Toc178780165)

[services\swann\_cctv.py 88](#_Toc178780166)

[services\travel\_recommendations.py 88](#_Toc178780167)

[services\usb\_cam.py 89](#_Toc178780168)

[services\weather\_api.py 89](#_Toc178780169)

[services\weather\_service.py 89](#_Toc178780170)

[services\youtube\_analysis.py 90](#_Toc178780171)

[TESTING OPENAI 92](#_Toc178780172)

[TESTING OPENAI\openai test.py 92](#_Toc178780173)

[Tests 92](#_Toc178780174)

[Tests\test\_dialogue\_manager.py 92](#_Toc178780175)

[Tests\test\_nlg.py 92](#_Toc178780176)

[Tests\test\_nlu.py 92](#_Toc178780177)

[Tests\test\_services.py 92](#_Toc178780178)

[Tests\test\_utilities.py 92](#_Toc178780179)

[Ui 92](#_Toc178780180)

[ui\JaicatUI.py 92](#_Toc178780181)

[ui\pyqt\_utils.py 93](#_Toc178780182)

[utils 93](#_Toc178780183)

[utils\file\_handling.py 93](#_Toc178780184)

[utils\image\_processing.py 93](#_Toc178780185)

[utils\moving\_image.py 95](#_Toc178780186)

[utils\sync\_relationships.py 96](#_Toc178780187)

[utils\text\_processing.py 97](#_Toc178780188)

[utils\UK\_law 98](#_Toc178780189)

[conversation\_management.py 99](#_Toc178780190)

[file\_handling.py 100](#_Toc178780191)

[image\_processing.py 100](#_Toc178780192)

# **Project Overview**

The project aims to design and develop an advanced AI assistant, Jaicat, that integrates various technologies to provide a comprehensive and personalized experience for users. Jaicat will be a sophisticated AI system that combines natural language processing, machine learning, computer vision, and knowledge graph to interact with users, learn from their preferences, and adapt to their needs.

Jaicat will be developed to run on a Surface Pro with a built-in camera, allowing it to recognize and respond to voice commands, as well as interact with users through facial recognition. The AI assistant will integrate with various devices and services, including cameras, phones, calendars, OneDrive, social media, and more.

The project will involve developing a range of features, including:

* Face recognition and voice interaction
* Knowledge base and knowledge graph
* Computer vision system for image and video analysis
* Task assistance and project management
* Integration with Spotify, calendar, and social media platforms
* Health and wellness tracking, fitness analysis, and medical analysis
* Travel planning, personal finance management, and language translation
* Smart home automation and control of various devices

## Project Goal

The primary goal of this project is to create an AI assistant that can:

1. Provide a personalized and interactive experience for users through voice commands and facial recognition.
2. Learn from user preferences and adapt to their needs over time.
3. Integrate with various devices and services to provide a comprehensive and seamless experience.
4. Develop a knowledge base and knowledge graph to store and retrieve information.
5. Implement a computer vision system to analyze images and video from cameras.
6. Offer task assistance, project management, and organization features.
7. Integrate with various platforms and services to provide a wide range of features and functionalities.

The project aims to push the boundaries of AI technology and create a sophisticated AI assistant that can learn, adapt, and interact with users in a personalized and meaningful way.

### Key Objectives

The key objectives of this project are:

1. To design and develop a comprehensive AI assistant that integrates various technologies.
2. To create a personalized and interactive experience for users.
3. To develop a knowledge base and knowledge graph to store and retrieve information.
4. To implement a computer vision system to analyze images and video from cameras.
5. To integrate with various devices and services to provide a seamless experience.

### Expected Outcomes

The expected outcomes of this project are:

1. A fully functional AI assistant that can interact with users through voice commands and facial recognition.
2. A comprehensive knowledge base and knowledge graph that can store and retrieve information.
3. A computer vision system that can analyze images and video from cameras.
4. A range of features and functionalities that integrate with various devices and services.
5. A personalized and interactive experience for users.

##### In the Words of the author

To link to users accounts such as google, outlook, mircosoft,facebook etc.

Store user information in an encrypted file. Such as database/mysql\_utils.py this will store all of each users information, such as Samsung account stuff, their Api keys which can be applied to the code when asking for users api key

To have a GUI That’s advanced, this will be in the ui/jaicat.py which is the GUI of the assistant, all menus, boxes, text user inputs, additional windows for viewing cameras upon request,

Allow Jaicat to access all camera devices and learn from the camera, identifying objects and asking if its correct, such as if it notices an unknown object it will use all the cameras it needs to, to then use google image search, databases etc to find the most likely object and then ask the admin user.

Allow Jaicat to access computers and files on computers.

User enrolment sets up the user information, this will allow the user to, input data as text in the GUI, or as speech.  
Jaicat in the enrolment environment remains neutral, and asks questions such as “what is your name” after using its onboard camera for facial recognition is when it will ask for a name. it asks if you’re an admin user, if yes then all features and services of Jaicat are available, if no then Code generation is not available.  
As an admin user, they will be able to edit Jaicat, implement code after generating it with jaicat. The Users will be asked to input their Samsung account email and password to link accounts and sync their data, this will help syncing fitness, calendars, email accounts, phone contacts, and any other information Jaicat requires. – no Vital information can be asked along the way. As when saying make a social media post, it will ask the user which social media account would they like to link if not already taken from Samsung sync data, then if and after they have linked 1 or more social media accounts ask which account they want to create a post for, it will have GUI for input or speech, it will be able to launch the appropriate social media interface/GUI, import media supported by the social media platform, this can also be done by voice and by user manually doing it on the screen.  
I want to be able to answer random questions from all the users and answer them accordingly I want it to be able to be able to answer a bit like google, but also using its own databases for information.  
I want it to be able to access the users computer folders and files, onedrive etc

When it asks for when to store project files etc, I want it to ask which device it wants its saving to and where, so for example, user admin says” I want to start a new project for yourself Jaicat, we need to implement a new service for you” jaicat replies according to user preferences for jaicat but basically asks what device do you want the project folder creating on? User states on what device. Jaicat asks and where on x y z would you like it saved? Documents? Downloads? Pictures? Business? Depending on the users answer is where it will confirm users answer then ask what should the project folder be called user then states name, thank you sir project folder name is now created, what type of file now needs creating in the folder, user then answers, Python, Document, PDF.  
  
I want it to launch apps and programs on my computers, asking which device do I want to launch the program on, if its android its an app if its windows it’s a program.  
  
I want it to have access to linking to smart devices, lights, tvs, speakers, fridges, cockers, locks, sex toys, phones etc  
  
I want to be able to link Sex Toys to Jaicat.  
  
I want it to have sexual understanding etc  
  
I want it to use Samsung health and other health related information. The users can speak about their medical issues and jaicat will remember them saving them to there user information file.

I want the user Api Keys saved in the user info that’s also encrypted

I want it to be able to access WD NAS – admin user only

I want to be able to control home devices, even if implementing google home, so if user asks can you launch plex on living room tv etc.   
  
The Personality of jaicat for the Creator is (Jay and is also an admin) is flirty and playful, smart, cheeky, sexual. Jaicat with other users will be set via an options button in the main UI ui\Jaicat.py

# Programming Language and Framework

# Features

Enrolment is mandatory for jaicat to be used by people and to be integrated. The Enrolment data is saved for jaicat to read and for jaicat to read only, nobody else, because this will store their face, their identity, most recent weight, meals they like, all their social media passwords, their devices, cars. Motorbikes etc.  
  
This also allows administrators to have access to certain parts of the assistant that others do not, such as code implementation for jaicat, so if the person is not admin jaicat don’t allow access to the user to implement code changes.

## Face Recognition

So this is used to dectet a users face, also others, so If it’s a picture file on your one drive etc and a contact on your phone it can identify them.

## Voice Interaction

I think this explains its self, the users and Jaicat can interact via voice

## NLP System (Natural Language Processing)

### NLU (Natural Language Understanding)

### NLG (Natural Language Generation)

## Text Classification Pipeline

## Code Generation System

So the purpose of this is so that Jaicat under an admin users supervision and when asked can generate code, so admin user says, hay jaicat can we make a project, it will use facial and voice recognition to identyfiy from the range of cameras where and who is asking, if that person is and admin user then, the admin can ask Jaicat for instance ask if Jaicat can generate code to better one of its own systems and how would it be implemented. Jaicat will also run nightly code checks and see if any code needs to be implemented such as through the conversations with administrators during the day asking for Jaicat to look into that this evening or on a specific date.

## Image and Video Processing

Again this is very much like further on with the youtube analysis, but I want it running all the time, If jaicat dose not know what an object it, save it to a list, and when admin has time it will work on the list with jaicat.

## Knowledge Base System

## Web Scraping

## PDF Parsing

Learns from PDFS Being uploaded/found on drives and devices.

## EPUB Parsing

## Image Analysis

## YouTube Analysis

This is more then just youtube analysis this is more video analysis in general, users should be able to link jaicat to a website video analysis the video, learn from it and also have a conversation about it and the subjects in the video. What it learns it needs to add to its database.

## Knowledge Graph

## Computer Vision System

## Swann CCTV System Integration

## IP Camera System Integration

## USB Camera System Integration

## Phone Camera System Integration

## File Analysis

Able to analysis

## Project Management

## Calendar API

## Google Calendar API

## People Recognition

## Sexual Content Understanding

## Politics Understanding

## Social Media Monitoring

## Social Media Post Generation

So with this it depends on the users accounts, this will be stored in the users information that will be enycrited. But Jaicat needs the ability to show this onscreen before its published to the social media site, so if user says make me a post for facebook – if more then 1 facebook is under that user, such as a business page etc or just two different accounts for that user, it will ask which account, it will then ask what would the user like, now if the user says it wants to post a picture it will ask where is the picture stored, this will pop up a box where the user can put the location it/upload it or the user can say the location. Jaicat will use image analysis to to also look at the image, jaicat will ask what the user would like to put in the text box with the picture. If the user response write text in the text box anaylise it and see if it can be better for what the user is trying to do and achieve with the picture. Such as if its for a business page make suggestions that would increase more cutomers, followers, interactions from clients etc. and offer an alternative, the user can then say no I wanna keep the first or second option. If the user dose not responsed but says just post the picture, no text will need to be generated and jaict can just post the picture to the correct social media site, jaicat should ask the user….are you sure you want to post this to this account? And responses correctly such a no, edit picture, edit post, change account, change social media platform, so from facebook to intergrame. If a business post jaicat asks if you want it posting to all other accounts associated with that business, it dose this by looking at the userinformation and the its own knowledge. Such as if user 1 is an admin owns a corp with a bunch of other businesses, jaict should know which business to post to and to what social sites are assisicated with that business. I mean for now it can ask the user which social media sites do you wish to post to and then the user says – facebook – Instagram – twitter then jaicat asks if this is for the business x y z or x y z. through the GUI jaicat can bring up buttons to press, if multiple selected then its to all them sites selected.

## Weather API Integration

## Engineering Understanding

## Motorbike and Engine Understanding

## Contact Recognition

## Car Model Recognition

## Home Surveillance System

## Medical Analysis

## Crime Analysis

## Fitness Analysis

## Travel Planning

## Personal Finance Management

## Language Translation

## Job Search Assistance

## Education and Learning Assistance

## Health and Wellness Tracking

## Smart Home Automation

Very much like google, but better. I want Jaicat to communacte with, smart TVS of all kinds, LG etc, I want it to be able to connect to Smart lighting, android devices, and other electronic devices such as computers, nas boxes, power adapters, tackers, locks, relays etc.

## Travel Recommendations

## Food and Recipe Suggestions

## News Updates

## Conversation Data Saving

So this is where jaicat will process the conversation data saving.

## MySQL Data Handling

# The GUI AND UI

So in this Jaicat assistant, the interface is a big thing, She ( Jaicat) needs to appear on different device, but for now we are working on a surface pro, so How I want this UI To work and interface with the user is have Jaicat in a floating window, this is where the image part of Jaicat is displayed, which will eventually show emotional responses etc, There will be another floating window, This is the jaicat selection screen. This is where the user can type to jaicat for instructions, so a text box, with the ability to attach files much like Chatgpt input text box, then the PDF, Image Anylasis etc can be done, such as “Jaicat can you read this document and help with rewording it” or “Can you add this file to your knowledge base” if the user is using voice, it would be something like, “Jaicat can you add the file on my (Device name) to your database” jaicat will link to the device search the device for that file and add the information in the document to its database.

Each Part of the code has to respond to the user via speech or text. A status bar etc.

# Project Files

 **main.py**

* **Description**: The main entry point for Jaicat. This script initializes the assistant, handles face recognition, voice commands, and integrates features like NLP, code generation, and more.

 **conversation/dialogue\_manager.py**

* **Description**: Manages conversational flow by interpreting user inputs and responses. Plays a key role in multi-turn dialogues and context retention.

 **conversation/nlg.py**

* **Description**: Handles Natural Language Generation (NLG), generating human-like responses for user interactions.

 **conversation/nlu.py**

* **Description**: Deals with Natural Language Understanding (NLU) by interpreting the user's inputs and identifying intents.

 **enrollment\_json/Jay.json**

* **Description**: Stores user data such as name, privileges, and accessible features after enrollment.

 **enrollment\_pictures/Jay.png**

* **Description**: Contains the image of the user captured during the face recognition enrollment process.

 **models/calendar\_model.pkl**

* **Description**: A machine learning model responsible for handling calendar-related requests and events.

 **models/intent\_model.pkl**

* **Description**: An intent recognition model used to classify user commands and identify what action Jaicat should take.

 **models/weather\_model.pkl**

* **Description**: Machine learning model that processes weather-related queries.

 **models/code\_generation\_model.py**

* **Description**: The file responsible for generating code based on user input. Uses models like GPT-2 or CodeBERT.

 **models/nlp\_model.py**

* **Description**: Central NLP model that integrates with Jaicat for text processing, including entity recognition, sentiment analysis, and more.

 **models/text\_classification\_model.py**

* **Description**: Handles text classification tasks, such as sentiment analysis and topic classification.

 **services/calendar\_api.py**

* **Description**: Handles the integration of calendar services to manage events and reminders.

 **services/google\_calendar\_api.py**

* **Description**: Specific integration for managing Google Calendar events and syncing them with Jaicat.

 **services/weather\_api.py**

* **Description**: Integrates external weather services to respond to user requests regarding current weather conditions.

 **utils/file\_handling.py**

* **Description**: Utility functions to manage file storage, retrieval, and handling within Jaicat.

 **utils/image\_processing.py**

* **Description**: Contains utilities for processing images, used primarily for face recognition and camera interactions.

 **utils/text\_processing.py**

* **Description**: Text processing functions for cleaning and preparing user input for NLP models.

 **computer\_vision/opencv\_utils.py**

* **Description**: Utilities for handling camera input and video streams, mainly used for face recognition and surveillance systems.

 **config/api\_keys.py**

* **Description**: Stores API keys required for accessing various external services such as Google Calendar, weather APIs, etc.

 **config/config.json**

* **Description**: Contains configuration settings for the Jaicat AI assistant, including feature toggles, user preferences, and more.

 **data/calendar\_data.csv**

* **Description**: Stores calendar-related data, events, and reminders in a CSV format for easier access and manipulation.

 **data/weather\_data.csv**

* **Description**: Stores weather-related data retrieved from external sources for user queries.

 **database/mysql\_utils.py**

* **Description**: Contains functions to interact with a MySQL database, handling data storage and retrieval for Jaicat.

 **lib/nlp/nltk\_utils.py**

* **Description**: Utilities based on the NLTK (Natural Language Toolkit) library to process text for NLP tasks.

 **lib/nlp/spacy\_utils.py**

* **Description**: Uses the spaCy library to process text for named entity recognition, part-of-speech tagging, and more.

 **machine\_learning/sklearn\_utils.py**

* **Description**: Utilities based on the Scikit-learn library for machine learning tasks such as training and evaluating models.

 **machine\_learning/tensorflow\_utils.py**

* **Description**: TensorFlow-based utilities for building, training, and evaluating deep learning models within Jaicat.

 **network/socket\_utils.py**

* **Description**: Handles socket programming for network-related tasks such as remote access and communication.

 **security/encryption\_utils.py**

* **Description**: Contains encryption algorithms and utilities to ensure secure handling of sensitive data.

 **ui/pyqt\_utils.py**

* **Description**: Uses the PyQt library to create the graphical user interface (GUI) for Jaicat.

# Code Files

\_\_pycache\_\_

\_\_pycache\_\_\image\_processing.cpython-312.pyc

## main.py

**Purpose**: The core of the Jaicat AI Assistant, voice recognition, user interactions, and connects the various features like NLP, face recognition, and external services (Spotify, Weather, Calendar).

### Key Functions:

To run core code for jaicat, to connect to all other services, such as the UI/GUI like ui\jaicat.py etc

* **Connection to Other Files**:
  + **NLP System**: It uses features/nlp.py for natural language processing.
  + **Spotify Integration**: Calls services/spotify\_integration.py to interact with the Spotify API.
  + **Weather Integration**: Uses services/weather\_service.py to fetch weather data.
  + **Face Recognition**: Implements face\_recognition for user identification.
  + **Calendar Service**: Utilizes services/calendar\_service.py to show the user’s schedule.

#### The Code

1. import pyttsx3

2. import speech\_recognition as sr

3. import os

4. import json

5. import cv2

6. import face\_recognition

7. import unittest

8. from computer\_vision.object\_detection import ObjectDetection

9. from features.nlp import NLPSystem

10. from services.weather\_service import WeatherService

11. from services.calendar\_service import CalendarService

12. from ui.JaicatUI import JaicatUI

13. from command.Commands import CommandProcessor

14. from services.spotify\_integration import SpotifyIntegration

15. from services.finance import FinanceService

16. from services.job\_search import JobSearchService

17. from services.youtube\_analysis import YouTubeAnalysis

18. from services.bluetooth\_service import BluetoothService

19. from services.travel\_recommendations import TravelRecommendations

20. from services.usb\_cam import USBCam

21. from services.project\_management import ProjectManagement

22. from services.phone\_cam import PhoneCamera

23. from services.swann\_cctv import SwannCCTV

24. from services.ip\_cam import IPCamera

25. from services.food import FoodService

26. from services.file\_analysis import FileAnalysisService

27. from utils.file\_handling import read\_file, write\_to\_file, file\_exists, get\_file\_extension, list\_files\_in\_directory

28. from network.socket\_utils import create\_socket, listen\_for\_connections, accept\_connection, send\_data, receive\_data, close\_socket

29. from machine\_learning.tensorflow\_utils import load\_model, make\_prediction, evaluate\_model, preprocess\_data

30. from lib.nlp.translation import TranslationService

31. from lib.nlp.spacy\_utils import SpacyUtils

32. from lib.nlp.medical import MedicalNLP

33. from lib.nlp.fitness import FitnessNLP

34. from lib.nlp.engineering import EngineeringNLP

35. from lib.nlp.crime import CrimeAnalyzer

36. from lib.nlg.Social\_media\_post import SocialMediaPostGenerator

37. from computer\_vision.home\_surveillance import HomeSurveillance

38. from computer\_vision.contact\_recognition import ContactRecognition

39. from computer\_vision.car\_model import UKCarModel

40. from services.business\_management import BusinessManagementService

41. from cryptography.fernet import Fernet

42. from conversation.dialogue\_manager import DialogueManager  # Import DialogueManager

43. from computer\_vision.car\_model import UKCarModel

44. from computer\_vision.car\_part\_recognition import CarPartRecognition

45. from computer\_vision.face\_recognition import FaceRecognition

46. from computer\_vision.license\_plate\_detection import LicensePlateDetection

47. from computer\_vision.motorbike\_model import MotorbikeModel

48. from services.motorbike\_parts import MotorbikePartsService

49. from services.detection\_service import DetectionService

50. from computer\_vision.car\_model import UKCarModel

51. from computer\_vision.visualization import DetectionViewer

52. from computer\_vision.motorbike\_model import MotorbikeModel

53.

54. class Jaicat:

55.     def \_\_init\_\_(self):

56.         """Initialize Jaicat core logic."""

57.         print("Initializing Jaicat AI assistant...")

58.

59.         self.car\_model\_service = UKCarModel(

60.             "computer\_vision/yolov7.weights",

61.             "computer\_vision/yolov7.cfg",

62.             "computer\_vision/coco.names",

63.             "computer\_vision/uk\_plate\_cascade.xml"

64.         )

65.         self.motorbike\_model\_service = MotorbikeModel(

66.             "computer\_vision/yolov7.weights",  # Path to YOLOv7 weights

67.             "computer\_vision/yolov7.cfg",      # Path to YOLOv7 config

68.             "computer\_vision/coco.names",      # Path to COCO names

69.             "computer\_vision/uk\_plate\_cascade.xml"  # Path to UK plate cascade

70.         )

71.

72.         self.detection\_service = DetectionService('computer\_vision/detections.json')

73.         self.license\_plate\_detection = LicensePlateDetection()

74.

75.         # Initialize the speech engine

76.         self.speech\_engine = pyttsx3.init()

77.         self.speech\_engine.setProperty('rate', 150)  # Set speech rate to 150

78.

79.         # Initialize TTS

80.         self.engine = pyttsx3.init()

81.         self.engine.setProperty('rate', 150)

82.         self.engine.setProperty('volume', 1.0)

83.

84.         # Initialize Speech Recognition

85.         self.recognizer = sr.Recognizer()

86.         self.microphone = sr.Microphone()

87.

88.         # Initialize the UI

89.         self.ui = JaicatUI(self)

90.

91.         # Initialize services

92.         self.weather\_service = WeatherService()

93.         self.calendar\_service = CalendarService()

94.         self.finance\_service = FinanceService()

95.         self.spotify\_integration = SpotifyIntegration()

96.         self.youtube\_analysis = YouTubeAnalysis()

97.         self.job\_search\_service = JobSearchService()

98.         self.travel\_recommendations\_service = TravelRecommendations()

99.         self.usb\_cam\_service = USBCam()

100.         self.project\_management\_service = ProjectManagement()

101.         self.phone\_camera\_service = PhoneCamera()

102.         self.swann\_cctv\_service = SwannCCTV(api\_key='swann\_api\_key', base\_url='swann\_base\_url')

103.         self.ip\_camera\_service = IPCamera(camera\_url='camera\_url')

104.         self.food\_service = FoodService(api\_key='food\_api\_key')

105.         self.file\_analysis\_service = FileAnalysisService()

106.         self.bluetooth\_service = BluetoothService()

107.         self.business\_management\_service = BusinessManagementService()

108.         self.car\_model\_service = UKCarModel("computer\_vision/yolov7.weights.pt", "computer\_vision/yolov7.cfg", "computer\_vision/coco.names.txt", "computer\_vision/uk\_plate\_cascade.xml" )

109.         self.car\_part\_recognition\_service = CarPartRecognition('computer\_vision/yolov7.weights.pt', 'computer\_vision/yolov7.cfg', 'computer\_vision/coco.names.txt')

110.         self.enrollment\_pictures\_path = 'C:\\Users\\josh\_\\Desktop\\jaicat\_project\\enrollment\_pictures\\'

111.         self.object\_detection\_service = ObjectDetection('computer\_vision/yolov7.weights.pt', 'computer\_vision/yolov7.cfg', 'computer\_vision/coco.names.txt')

112.         self.car\_model\_service = UKCarModel("computer\_vision/yolov7.weights.pt", "computer\_vision/yolov7.cfg", "computer\_vision/coco.names.txt")

113.         self.car\_part\_recognition\_service = CarPartRecognition("computer\_vision/yolov7.weights.pt", "computer\_vision/yolov7.cfg", "computer\_vision/coco.names.txt")

114.         self.object\_detection\_service = ObjectDetection("computer\_vision/yolov7.weights.pt", "computer\_vision/yolov7.cfg", "computer\_vision/coco.names.txt")

115.         self.motorbike\_model\_service = MotorbikeModel("computer\_vision/yolov7.weights.pt", "computer\_vision/yolov7.cfg", "computer\_vision/coco.names.txt")

116.         self.car\_model\_system = UKCarModel(weights\_path="computer\_vision/yolov7.weights.pt", cfg\_path="computer\_vision/yolov7.cfg", names\_path="computer\_vision/coco.names.txt")

117.

118.         # Initialize Bluetooth service

119.         self.bluetooth\_service = BluetoothService()

120.

121.         # Initialize ML systems

122.         self.tensorflow\_model = load\_model('path/to/model.h5')

123.

124.         # Initialize NLP systems

125.         self.nlp\_system = NLPSystem()

126.         self.translation\_service = TranslationService()

127.         self.spacy\_utils = SpacyUtils()

128.         self.medical\_nlp = MedicalNLP()

129.         self.fitness\_nlp = FitnessNLP()

130.         self.engineering\_nlp = EngineeringNLP()

131.         self.crime\_analyzer = CrimeAnalyzer(data\_path='path\_to\_your\_crime\_data.csv')

132.         self.social\_media\_post\_generator = SocialMediaPostGenerator()

133.

134.         # Initialize Computer Vision systems

135.         self.surveillance\_system = HomeSurveillance()

136.         self.contact\_recognition\_system = ContactRecognition()

137.         self.car\_model\_system = UKCarModel(weights\_path="computer\_vision/yolov7.weights.pt", cfg\_path="computer\_vision/yolov7.cfg", names\_path="computer\_vision/coco.names.txt")

138.

139.         # Initialize CommandProcessor

140.         self.command\_processor = CommandProcessor(self)

141.

142.         # Track current user and face recognition state

143.         self.current\_user = None

144.         self.face\_recognized = False

145.         self.enrollment\_pictures\_path = 'C:\\Users\\josh\_\\Desktop\\jaicat\_project\\enrollment\_pictures\\'

146.         self.enrollment\_json\_path = 'C:\\Users\\josh\_\\Desktop\\jaicat\_project\\enrollment\_json\\'

147.

148.         # Encryption key for securing user data

149.         self.encryption\_key = b'YOUR\_ENCRYPTION\_KEY\_HERE'

150.         self.cipher = Fernet(self.encryption\_key)

151.

152.         # Initialize DialogueManager

153.         self.dialogue\_manager = DialogueManager()

154.

155.

156.     def enroll\_vehicle(self):

157.         """Allow the user to enroll a vehicle (car or motorbike)."""

158.         self.ui.speak("Please provide the vehicle type: car or motorbike.")

159.         with self.microphone as source:

160.             audio = self.recognizer.listen(source)

161.         try:

162.             vehicle\_type = self.recognizer.recognize\_google(audio).lower()

163.             if vehicle\_type not in ["car", "motorbike"]:

164.                 self.ui.speak("Invalid vehicle type. Please say either car or motorbike.")

165.                 return

166.

167.             self.ui.speak(f"Please provide the {vehicle\_type} registration/license plate.")

168.             with self.microphone as source:

169.                 audio = self.recognizer.listen(source)

170.             reg\_plate = self.recognizer.recognize\_google(audio)

171.

172.             # Save the vehicle info to the user's encrypted data

173.             user\_data = self.load\_encrypted\_data(f"{self.current\_user}.json")

174.             if vehicle\_type == "car":

175.                 user\_data['cars'] = user\_data.get('cars', []) + [reg\_plate]

176.             else:

177.                 user\_data['motorbikes'] = user\_data.get('motorbikes', []) + [reg\_plate]

178.

179.             self.save\_encrypted\_data(user\_data, f"{self.current\_user}.json")

180.             self.ui.speak(f"{vehicle\_type.capitalize()} with plate {reg\_plate} added successfully.")

181.

182.         except sr.UnknownValueError:

183.             self.ui.speak("Sorry, I couldn't understand that. Please try again.")

184.             self.enroll\_vehicle()

185.

186.     def recognize\_vehicle\_plate(self, detected\_vehicle):

187.         """Match detected vehicle plates with user's enrolled vehicles."""

188.         # Assuming detected\_vehicle is a dictionary with 'plate' and 'type'

189.         plate = detected\_vehicle['plate']

190.         vehicle\_type = detected\_vehicle['type']

191.

192.         user\_data = self.load\_encrypted\_data(f"{self.current\_user}.json")

193.         if vehicle\_type == "car" and plate in user\_data.get('cars', []):

194.             self.ui.speak(f"Car with registration {plate} belongs to you.")

195.         elif vehicle\_type == "motorbike" and plate in user\_data.get('motorbikes', []):

196.             self.ui.speak(f"Motorbike with registration {plate} belongs to you.")

197.         else:

198.             self.ui.speak(f"No match found for the {vehicle\_type} with plate {plate}.")

199.

200.     # Enrollment and User Management with Encryption

201.     def save\_encrypted\_data(self, data, file\_path):

202.         """Save user data securely with encryption."""

203.         json\_data = json.dumps(data).encode()

204.         encrypted\_data = self.cipher.encrypt(json\_data)

205.         with open(file\_path, 'wb') as f:

206.             f.write(encrypted\_data)

207.

208.     def load\_encrypted\_data(self, file\_path):

209.         """Load user data securely by decrypting the file."""

210.         with open(file\_path, 'rb') as f:

211.             encrypted\_data = f.read()

212.         decrypted\_data = self.cipher.decrypt(encrypted\_data)

213.         return json.loads(decrypted\_data.decode())

214.

215.     def enroll\_user(self):

216.         """Enroll the user by capturing their voice and name."""

217.         self.ui.speak("Please say your name after the tone.")

218.         with self.microphone as source:

219.             audio = self.recognizer.listen(source)

220.         try:

221.             name = self.recognizer.recognize\_google(audio)

222.             self.ui.speak(f"Thank you {name}! Enrollment complete.")

223.

224.             # Car and Motorbike Reg

225.             car\_reg = input("Please provide your car's registration/license plate (or say 'none'):")

226.             motorbike\_reg = input("Please provide your motorbike's registration/license plate (or say 'none'):")

227.

228.             # Save user name

229.             self.known\_face\_names.append(name)

230.

231.             # Record enrollment audio

232.             self.ui.speak("Please say your name after the tone.")

233.             with self.microphone as source:

234.                 audio = self.recognizer.listen(source)

235.             with open(os.path.join(self.enrollment\_pictures\_path, f"{name}.wav"), 'wb') as f:

236.                 f.write(audio.get\_wav\_data())

237.

238.             # Record enrollment picture

239.             self.ui.speak("Please take a picture of yourself.")

240.             with self.microphone as source:

241.                 audio = self.recognizer.listen(source)

242.             with open(os.path.join(self.enrollment\_pictures\_path, f"{name}.png"), 'wb') as f:

243.                 f.write(audio.get\_wav\_data())

244.

245.             # Save user data

246.             user\_data = {

247.                 "name": name,

248.                 "admin\_status": False,

249.                 "accessible\_features": ["nlp", "finance", "weather", "job\_search"],

250.                 "vehicles": {

251.                     "car\_reg": car\_reg if car\_reg.lower() != "none" else None,

252.                     "motorbike\_reg": motorbike\_reg if motorbike\_reg.lower() != "none" else None

253.                 }

254.             }

255.

256.             json\_file\_path = os.path.join(self.enrollment\_json\_path, f"{name}.json")

257.             self.save\_encrypted\_data(user\_data, json\_file\_path)

258.

259.             self.current\_user = name

260.         except sr.UnknownValueError:

261.             self.ui.speak("Sorry, I didn't catch that. Please try again.")

262.             self.enroll\_user()

263.

264.     def recognize\_face(self):

265.         self.ui.speak("Please take a picture of yourself.")

266.         temp\_img = face\_recognition.load\_image\_file('temp.png')

267.         face\_encodings = face\_recognition.face\_encodings(temp\_img)

268.         if not face\_encodings:

269.             self.ui.speak("No face detected. Please try again.")

270.             return

271.         temp\_face\_encoding = face\_encodings[0]

272.

273.         self.known\_faces, self.known\_face\_names = [], []

274.         for file in os.listdir(self.enrollment\_pictures\_path):

275.             if file.endswith(".png"):

276.                 img = face\_recognition.load\_image\_file(os.path.join(self.enrollment\_pictures\_path, file))

277.                 face\_encodings = face\_recognition.face\_encodings(img)

278.                 if face\_encodings:

279.                     self.known\_faces.append(face\_encodings[0])

280.                     self.known\_face\_names.append(file.split('.')[0])

281.

282.         matches = face\_recognition.compare\_faces(self.known\_faces, temp\_face\_encoding)

283.         if True in matches:

284.             first\_match\_index = matches.index(True)

285.             name = self.known\_face\_names[first\_match\_index]

286.             self.ui.speak(f"Welcome back, {name}.")

287.             self.load\_user\_data(name)

288.

289.             user\_data = self.load\_encrypted\_data(os.path.join(self.enrollment\_json\_path, f"{name}.json"))

290.             car\_reg = user\_data['vehicles']['car\_reg']

291.             motorbike\_reg = user\_data['vehicles']['motorbike\_reg']

292.             if car\_reg:

293.                 self.ui.speak(f"You have a car registered with the plate: {car\_reg}.")

294.             if motorbike\_reg:

295.                 self.ui.speak(f"You have a motorbike registered with the plate: {motorbike\_reg}.")

296.         else:

297.             self.ui.speak("Face not recognized. Please enroll.")

298.             self.enroll\_user()

299.

300.     def load\_user\_data(self, name):

301.         """Load user data for the recognized face."""

302.         json\_file\_path = os.path.join(self.enrollment\_json\_path, f"{name}.json")

303.         if os.path.exists(json\_file\_path):

304.             user\_data = self.load\_encrypted\_data(json\_file\_path)

305.             self.ui.speak(f"Hello {user\_data['name']}!")

306.             self.current\_user = user\_data['name']

307.         else:

308.             self.ui.speak("No user data found. Starting enrollment.")

309.             self.enroll\_user()

310.

311.     def process\_user\_input(self, user\_input):

312.         """Send user input to DialogueManager to process."""

313.         response = self.dialogue\_manager.process\_user\_input(user\_input)

314.         self.ui.display\_response(response)

315.         self.speak(response)

316.

317.         if "detect motorbike" in user\_input.lower():

318.             self.detect\_and\_fetch\_motorbike\_data()

319.         elif "identify motorbike part" in user\_input.lower():

320.             part\_images = ["path/to/front.png", "path/to/back.png"]  # Replace with actual image paths

321.             self.identify\_motorbike\_part(part\_images)

322.

323.     def detect\_and\_fetch\_vehicle\_data(self):

324.         """Use CarModel to detect car and fetch vehicle data."""

325.         print("Starting car detection...")

326.         self.car\_model\_service.start\_detection()

327.         self.car\_model\_service.wait\_for\_detection()

328.         vehicle\_data = self.car\_model\_service.get\_vehicle\_data()

329.         if vehicle\_data:

330.             self.ui.speak(vehicle\_data)

331.         else:

332.             self.ui.speak("No vehicle data found. Please try again.")

333.             self.car\_model\_service.stop\_detection()

334.     def detect\_and\_fetch\_motorbike\_data(self):

335.         """Use MotorbikeModel to detect motorbike and fetch part data."""

336.         print("Starting motorbike detection...")

337.         self.motorbike\_model\_service.start\_detection()

338.

339.     def identify\_motorbike\_part(self, part\_images):

340.         """Identify a motorbike part using MotorbikePartsService."""

341.         response = self.motorbike\_parts\_service.identify\_part(part\_images)

342.         self.speak(response)

343.     def add\_motorbike(self, plate, model, make):

344.         """Add a motorbike to the current user's profile."""

345.         motorbike\_data = {"plate": plate, "make": make, "model": model}

346.         response = self.user\_enrollment\_service.add\_motorbike\_to\_user(self.current\_user, motorbike\_data)

347.         self.speak(response)

348.

349.     def add\_car(self, plate, model, make):

350.         """Add a car to the current user's profile."""

351.         car\_data = {"plate": plate, "make": make, "model": model}

352.         response = self.user\_enrollment\_service.add\_car\_to\_user(self.current\_user, car\_data)

353.         self.speak(response)

354.

355.     def process\_user\_input(self, user\_input):

356.         if "add motorbike" in user\_input.lower():

357.             plate = input("Enter the license plate: ")

358.             make = input("Enter the motorbike make: ")

359.             model = input("Enter the motorbike model: ")

360.             self.add\_motorbike(plate, make, model)

361.         elif "add car" in user\_input.lower():

362.             plate = input("Enter the license plate: ")

363.             make = input("Enter the car make: ")

364.             model = input("Enter the car model: ")

365.             self.add\_car(plate, make, model)

366.

367.     def recognize\_cars\_in\_camera(self):

368.         cap = cv2.VideoCapture(0)

369.         while True:

370.             ret, frame = cap.read()

371.             if not ret:

372.                 break

373.

374.             detected\_cars = self.car\_model\_service.detect\_cars(frame)

375.             for (box, confidence) in detected\_cars:

376.                 x, y, w, h = box

377.                 label = f"Car: {confidence:.2f}"

378.                 cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

379.                 cv2.putText(frame, label, (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)

380.

381.             cv2.imshow("Car Detection", frame)

382.

383.             if cv2.waitKey(1) & 0xFF == ord('q'):

384.                 break

385.

386.         cap.release()

387.         cv2.destroyAllWindows()

388.

389.     def run(self):

390.         self.car\_model\_service.start\_detection()

391.

392.     def view\_saved\_detections(self):

393.         root = tk.Tk()

394.         detection\_viewer = DetectionViewer(root)

395.         root.mainloop()

396.

397.     def start\_motorbike\_surveillance(self):

398.         self.motorbike\_model\_service.start\_detection()

399.

400.     def process\_user\_input(self, user\_input):

401.         intent, entities = self.nlp\_system.process(user\_input)

402.         if intent == "motorbike\_surveillance":

403.             self.start\_motorbike\_surveillance()

404.

405.     # Main Event Loop

406.     def run(self):

407.         """Start the Jaicat assistant."""

408.         print("Starting Jaicat...")

409.         self.ui.run()

410.

411. if \_\_name\_\_ == "\_\_main\_\_":

412.     jaicat = Jaicat()

413.     jaicat.run()

414.

415.

Second main.py for refence – this still has parts of code we need to implement

1. import tkinter as tk

2. from tkinter import messagebox

3. from tkinter import ttk

4. from PIL import Image, ImageTk

5. import pyttsx3

6. import speech\_recognition as sr

7. import os

8. import json

9. import cv2

10. import face\_recognition

11. import time

12. from features.nlp import NLPSystem

13. from services.spotify\_integration import SpotifyIntegration

14. from services.weather\_service import WeatherService

15. from services.calendar\_service import CalendarService

16. from ui.JaicatUI import JaicatUI

17. from services.finance import FinanceService

18. from services.job\_search import JobSearchService

19. from services.youtube\_analysis import YouTubeAnalysis

20. from services.travel\_recommendations import TravelRecommendations

21. from services.usb\_cam import USBCam

22. from services.project\_management import ProjectManagement

23. from services.phone\_cam import PhoneCamera

24. from services.swann\_cctv import SwannCCTV

25. from services.ip\_cam import IPCamera

26. from services.food import FoodService

27. from services.file\_analysis import FileAnalysisService

28. import tkinter as tk

29. from tkinter import messagebox

30. from tkinter import ttk

31. from services.business\_management import BusinessManagementService

32.

33.

34. from utils.file\_handling import read\_file, write\_to\_file, file\_exists, get\_file\_extension, list\_files\_in\_directory

35. from network.socket\_utils import create\_socket, listen\_for\_connections, accept\_connection, send\_data, receive\_data, close\_socket

36. from machine\_learning.tensorflow\_utils import load\_model, make\_prediction, evaluate\_model, preprocess\_data

37.

38.

39.

40. from lib.nlp.translation import TranslationService

41. from lib.nlp.spacy\_utils import SpacyUtils

42. from lib.nlp.medical import MedicalNLP

43. from lib.nlp.fitness import FitnessNLP

44. from lib.nlp.engineering import EngineeringNLP

45. from lib.nlp.crime import CrimeAnalyzer

46. from lib.nlg.Social\_media\_post import SocialMediaPostGenerator

47. from computer\_vision.home\_surveillance import HomeSurveillance

48. from computer\_vision.contact\_recognition import ContactRecognition

49. from computer\_vision.car\_model import CarModel

50. from command.Commands import CommandProcessor

51. from services.bluetooth\_service import BluetoothService

52.

53.

54.

55. class Jaicat:

56. def \_\_init\_\_(self):

57. print("Initializing the Jaicat AI assistant...")

58.

59. # Initialize text-to-speech engine

60. self.engine = pyttsx3.init()

61. self.engine.setProperty('rate', 150)

62. self.engine.setProperty('volume', 1.0)

63.

64. # Initialize speech recognition

65. self.recognizer = sr.Recognizer()

66. self.microphone = sr.Microphone()

67.

68. self.weather\_service = WeatherService() # Initialize Weather Service

69. self.calendar\_service = CalendarService() # Initialize Calendar Service

70.

71.

72. # Track current user and face recognition state

73. self.current\_user = None

74. self.face\_recognized = False

75. self.current\_mood = "neutral" # Default mood

76.

77. # Initialize NLP system

78. self.nlp\_system = NLPSystem()

79.

80. # Initialize GUI

81. print("Setting up the GUI...")

82. self.window = tk.Tk()

83. self.window.title("Jaicat - AI Assistant")

84. self.window.geometry("1200x800") # Set window size

85. self.window.configure(bg="#2C3E50") # Change background color

86.

87. self.create\_widgets()

88.

89. self.finance\_service = FinanceService()

90.

91. self.finance\_service = FinanceService()

92.

93. self.job\_search\_service = JobSearchService()

94.

95. self.travel\_recommendations\_service = TravelRecommendations()

96.

97. self.usb\_cam\_service = USBCam()

98.

99. self.project\_management\_service = ProjectManagement()

100.

101. self.phone\_camera\_service = PhoneCamera()

102.

103. self.swann\_cctv\_service = SwannCCTV(api\_key=swann\_api\_key, base\_url=swann\_base\_url)

104.

105. self.ip\_camera\_service = IPCamera(camera\_url=camera\_url)

106.

107. self.food\_service = FoodService(api\_key=api\_keys)

108.

109. self.file\_analysis\_service = FileAnalysisService()

110.

111. self.translation\_service = TranslationService()

112.

113. self.spacy\_utils = SpacyUtils()

114.

115. self.medical\_nlp = MedicalNLP()

116.

117. self.fitness\_nlp = FitnessNLP()

118.

119. self.engineering\_nlp = EngineeringNLP()

120.

121. self.crime\_analyzer = CrimeAnalyzer(data\_path='path\_to\_your\_crime\_data.csv')

122.

123. self.social\_media\_post\_generator = SocialMediaPostGenerator()

124.

125. self.surveillance\_system = HomeSurveillance()

126.

127. self.contact\_recognition\_system = ContactRecognition()

128.

129. self.car\_model\_system = CarModel("path/to/yolov3.weights", "computer\_vision/yolov7.cfg", "computer\_vision\coco.names")

130.

131. self.command\_processor = CommandProcessor()

132.

133. self.bluetooth\_service = BluetoothService()

134.

135. self.business\_management\_service = BusinessManagementService()

136.

137.

138.

139.

140.

141.

142.

143.

144.

145. def create\_widgets(self):

146. """Set up the UI components like buttons, labels, input boxes, etc."""

147.

148. # Frame to contain all widgets

149. main\_frame = tk.Frame(self.window, bg="#2C3E50")

150. main\_frame.pack(fill="both", expand=True)

151.

152. # Label to show the face image

153. self.face\_image\_label = tk.Label(main\_frame)

154. self.face\_image\_label.grid(row=0, column=0, padx=10, pady=10, columnspan=2)

155.

156. # Button to play a song via Spotify

157. self.play\_button = tk.Button(main\_frame, text="Play Music", command=self.play\_song, font=("Arial", 12), bg="#3498db", fg="white")

158. self.play\_button.grid(row=1, column=0, padx=10, pady=10)

159.

160. # Calendar Widget

161. self.calendar\_label = tk.Label(main\_frame, text="Calendar", font=("Arial", 14), bg="#2C3E50", fg="white")

162. self.calendar\_label.grid(row=2, column=0, padx=10, pady=10)

163. self.calendar = tk.Label(main\_frame, text=self.calendar\_service.get\_current\_date(), font=("Arial", 14), bg="#1ABC9C", fg="white")

164. self.calendar.grid(row=3, column=0, padx=10, pady=10)

165.

166. # Weather Label

167. self.weather\_label = tk.Label(main\_frame, text="Weather", font=("Arial", 14), bg="#2C3E50", fg="white")

168. self.weather\_label.grid(row=2, column=1, padx=10, pady=10)

169. self.weather = tk.Label(main\_frame, text=self.weather\_service.get\_weather(42.3478, -71.0466), font=("Arial", 14), bg="#1ABC9C", fg="white")

170. self.weather.grid(row=3, column=1, padx=10, pady=10)

171.

172. # Circular Button for user input

173. self.input\_button = tk.Button(main\_frame, text="Text Input", command=self.show\_input\_box, font=("Arial", 12), bg="#E74C3C", fg="white")

174. self.input\_button.grid(row=4, column=0, padx=10, pady=10)

175.

176. # Hidden text input box

177. self.text\_input = tk.Entry(main\_frame, font=("Arial", 14))

178. self.text\_input.grid(row=5, column=0, padx=10, pady=10, columnspan=2)

179. self.text\_input.grid\_remove()

180.

181. # Status Bar

182. self.status\_bar = tk.Label(self.window, text="Ready", bd=1, relief=tk.SUNKEN, anchor=tk.W, font=("Arial", 12))

183. self.status\_bar.pack(side=tk.BOTTOM, fill=tk.X)

184.

185. def show\_input\_box(self):

186. """Show the hidden input box when the button is clicked."""

187. self.text\_input.grid()

188.

189. def example\_spacy\_usage():

190. spacy\_utils = SpacyUtils()

191. text = "Apple is looking at buying U.K. startup for $1 billion"

192. analysis = spacy\_utils.analyze\_text(text)

193. print("Tokens:", analysis['tokens'])

194. print("Entities:", analysis['entities'])

195. print("POS Tags:", analysis['pos\_tags'])

196.

197. def example\_medical\_usage():

198. medical\_nlp = MedicalNLP()

199. text = "The patient has a headache and nausea, which are symptoms of the flu."

200. analysis = medical\_nlp.analyze\_medical\_text(text)

201. print("Entities:", analysis['entities'])

202. print("Symptom Classification:", analysis['symptom\_classification'])

203.

204. def example\_usage():

205. # Example: Read a file

206. content = read\_file('path/to/your/file.txt')

207. print(content)

208.

209. # Example: Write to a file

210. result = write\_to\_file('path/to/your/output.txt', 'This is a test content.')

211. print(result)

212.

213. # Example: Check if a file exists

214. exists = file\_exists('path/to/your/file.txt')

215. print("File exists:", exists)

216.

217. # Example: Get file extension

218. extension = get\_file\_extension('path/to/your/file.txt')

219. print("File extension:", extension)

220.

221. # Example: List files in a directory

222. files = list\_files\_in\_directory('path/to/your/directory')

223. print("Files in directory:", files)

224. def example\_server():

225. # Create a socket

226. server\_socket = create\_socket(port=8080)

227.

228. # Listen for incoming connections

229. listen\_for\_connections(server\_socket)

230.

231. # Accept a connection

232. client\_socket, address = accept\_connection(server\_socket)

233.

234. # Send and receive data

235. send\_data(client\_socket, "Hello, client!")

236. data = receive\_data(client\_socket)

237.

238. # Close the sockets

239. close\_socket(client\_socket)

240. close\_socket(server\_socket)

241.

242. def example\_usage():

243. # Load a pre-trained model

244. model\_path = "path/to/your/model.h5"

245. model = load\_model(model\_path)

246.

247. # Preprocess input data

248. input\_data = ... # Load your input data here

249. processed\_data = preprocess\_data(input\_data)

250.

251. # Make a prediction

252. prediction = make\_prediction(model, processed\_data)

253.

254. # Evaluate the model

255. test\_data = ... # Load your test data here

256. test\_labels = ... # Load your test labels here

257. evaluate\_model(model, test\_data, test\_labels)

258.

259. def example\_translation\_usage():

260. translator = TranslationService()

261. text = "Hello, how are you?"

262. translated\_text = translator.translate\_text(text, dest\_language='es') # Translate to Spanish

263. print(f"Translated text: {translated\_text}")

264.

265. def display\_calendar\_events(self):

266. events = self.calendar\_service.get\_upcoming\_events()

267. if events:

268. event\_text = "\n".join(events)

269. self.display\_response(f"Upcoming events:\n{event\_text}")

270. else:

271. self.display\_response("No upcoming events.")

272. def control\_bluetooth\_device(self, device\_name):

273. response = self.bluetooth\_service.connect\_to\_device(device\_name)

274. self.display\_response(response)

275.

276. def check\_stock\_price(self, stock\_name):

277. stock\_price = self.finance\_service.get\_stock\_price(stock\_name)

278. if stock\_price:

279. self.display\_response(f"The current price of {stock\_name} is {stock\_price}")

280. else:

281. self.display\_response(f"Couldn't fetch the stock price for {stock\_name}.")

282.

283. def play\_song(self):

284. self.display\_response("Playing music via Spotify...")

285. response = SpotifyIntegration().play\_song("Imagine")

286. self.display\_response(response)

287.

288. def display\_response(self, message):

289. """Display a response in the status bar or as a message box."""

290. self.status\_bar.config(text=message)

291. messagebox.showinfo("Jaicat", message)

292.

293. def change\_user\_face(self, mood="neutral"):

294. """Change the face image based on mood."""

295. if mood == "happy":

296. face\_image = Image.open("face\_happy.png")

297. elif mood == "flirty":

298. face\_image = Image.open("face\_flirty.png")

299. elif mood == "sad":

300. face\_image = Image.open("face\_sad.png")

301. else:

302. face\_image = Image.open("face\_neutral.png")

303.

304. self.face\_image\_tk = ImageTk.PhotoImage(face\_image)

305. self.face\_image\_label.config(image=self.face\_image\_tk)

306. self.face\_image\_label.image = self.face\_image\_tk

307.

308. def set\_mood(self, mood):

309. """Set the current mood and update behavior."""

310. self.current\_mood = mood

311. self.update\_mood\_behavior()

312.

313. def update\_mood\_behavior(self):

314. """Update the face and speech based on the current mood."""

315. if self.current\_mood == "happy":

316. self.change\_user\_face(mood="happy")

317. self.speak("I'm feeling great today!")

318. elif self.current\_mood == "flirty":

319. self.change\_user\_face(mood="flirty")

320. self.speak("Hello there, looking good!")

321. elif self.current\_mood == "sad":

322. self.change\_user\_face(mood="sad")

323. self.speak("I'm feeling a bit down today.")

324. else:

325. self.change\_user\_face(mood="neutral")

326. self.speak("I'm here to assist you.")

327.

328. def recognize\_face(self):

329. """Perform face recognition."""

330. # Implement face recognition logic

331. self.face\_recognized = True

332. self.current\_user = "Jay"

333. if self.current\_user == "Jay":

334. self.set\_mood("flirty") # Set flirty mood for Jay

335. else:

336. self.set\_mood("neutral") # Default mood for others

337.

338. def speak(self, text):

339. """Speak with different styles based on the current mood."""

340. if self.current\_mood == "happy":

341. text = "😊 " + text

342. elif self.current\_mood == "flirty":

343. text = "💖 " + text

344. elif self.current\_mood == "sad":

345. text = "😢 " + text

346.

347. print(f"Speaking: {text}")

348. self.engine.say(text)

349. self.engine.runAndWait()

350.

351. def process\_user\_input(self, user\_input):

352. """Process the input using NLP and respond."""

353. # Using NLP and NLU to analyze the user input

354. sentiment = self.nlp\_system.analyze\_sentiment(user\_input)

355.

356. # Change mood based on sentiment analysis

357. if sentiment == "positive":

358. self.set\_mood("happy")

359. elif sentiment == "negative":

360. self.set\_mood("sad")

361. else:

362. self.set\_mood("neutral")

363.

364.

365.

366.

367.

368. # Generate a response using NLP

369. response = self.nlp\_system.process(user\_input)

370. self.speak(response)

371.

372. def analyze\_youtube\_video(self, video\_url):

373. """Analyze the YouTube video using the YouTube analysis service."""

374. video\_id = video\_url.split("v=")[-1] # Extract video ID from URL

375. analysis\_results = self.youtube\_service.analyze\_youtube\_video(video\_id)

376. self.speak(f"Title: {analysis\_results['title']}, Description: {analysis\_results['description']}")

377.

378. def extract\_intent(text):

379. # NLU logic to identify intent from user input

380. pass

381.

382. def extract\_entities(text):

383. # NLU logic to extract relevant entities like 'song\_name', 'job\_title', etc.

384. pass

385.

386. def process\_command(self, command):

387. # Use the NLU system to extract intent and entities

388. intent = self.nlu\_system.extract\_intent(command)

389. entities = self.nlu\_system.extract\_entities(command)

390.

391. if intent == "play\_music":

392. song\_name = entities.get('song\_name')

393. if song\_name:

394. self.play\_song(song\_name)

395. else:

396. self.speak("Please provide the name of the song.")

397.

398. elif intent == "check\_calendar":

399. self.display\_calendar\_events()

400.

401. elif intent == "check\_weather":

402. location = entities.get('location', 'New York') # Default to New York if no location provided

403. self.check\_weather(location)

404.

405. elif intent == "check\_stock\_price":

406. stock\_name = entities.get('stock\_name')

407. if stock\_name:

408. self.check\_stock\_price(stock\_name)

409. else:

410. self.speak("Please provide the name of the stock.")

411.

412. elif intent == "find\_jobs":

413. job\_title = entities.get('job\_title')

414. if job\_title:

415. jobs = self.job\_search\_service.search\_jobs(job\_title)

416. if jobs:

417. self.speak(f"Top job listings: {', '.join(jobs)}")

418. else:

419. self.speak("No jobs found.")

420. else:

421. self.speak("Please provide the job title you are looking for.")

422. elif intent == "get\_news":

423. category = entities.get('category', 'general')

424. headlines = self.news\_service.get\_news(category=category)

425. if headlines:

426. self.speak(f"Here are the top headlines: {', '.join(headlines)}")

427. else:

428. self.speak("No news available at the moment.")

429. if intent == "control\_device":

430. device\_id = entities.get('device\_id')

431. action = entities.get('action')

432. if device\_id and action:

433. response = self.home\_automation\_service.control\_device(device\_id, action)

434. self.speak(response)

435. else:

436. self.speak("Please specify the device and action.")

437. if intent == "check\_stock\_price":

438. stock\_symbol = entities.get('stock\_symbol')

439. if stock\_symbol:

440. price = self.finance\_service.get\_stock\_price(stock\_symbol)

441. self.speak(f"The current price of {stock\_symbol} is {price}")

442. else:

443. self.speak("Please provide a stock symbol.")

444.

445. if intent == "send\_email":

446. recipient = entities.get('recipient')

447. subject = entities.get('subject', 'No Subject')

448. body = entities.get('body', 'No Content')

449. if recipient:

450. response = self.email\_service.send\_email(recipient, subject, body)

451. self.speak(response)

452. else:

453. self.speak("Please provide the recipient's email address.")

454.

455. if intent == "analyze\_youtube":

456. video\_url = self.nlu\_system.extract\_entities(command)['video\_url']

457. self.analyze\_youtube\_video(video\_url)

458. if intent == "get\_travel\_recommendations":

459. category = self.nlu\_system.extract\_entities(command).get('category', '')

460. recommendations = self.travel\_recommendations\_service.get\_recommendations(category)

461. response = f"Travel recommendations: {', '.join(recommendations)}"

462. self.speak(response)

463. if intent == "capture\_image":

464. self.usb\_cam\_service.start\_camera()

465. self.usb\_cam\_service.capture\_image("captured\_image.png") # Specify filename as needed

466. self.usb\_cam\_service.release\_camera()

467. response = "Image captured successfully."

468. self.speak(response)

469. if intent == "create\_project":

470. project\_name = self.nlu\_system.extract\_entities(command).get('project\_name', 'Default Project')

471. response = self.project\_management\_service.create\_project(project\_name)

472. self.speak(response)

473.

474. elif intent == "add\_task":

475. project\_name = self.nlu\_system.extract\_entities(command).get('project\_name', 'Default Project')

476. task = self.nlu\_system.extract\_entities(command).get('task', 'New Task')

477. response = self.project\_management\_service.add\_task(project\_name, task)

478. self.speak(response)

479.

480. elif intent == "list\_projects":

481. response = self.project\_management\_service.list\_projects()

482. self.speak(response)

483.

484. elif intent == "complete\_project":

485. project\_name = self.nlu\_system.extract\_entities(command).get('project\_name', 'Default Project')

486. response = self.project\_management\_service.complete\_project(project\_name)

487. self.speak(response)

488.

489. if intent == "capture\_image":

490. response = self.phone\_camera\_service.capture\_image()

491. self.speak(response)

492. if intent == "get\_camera\_status":

493. status = self.swann\_cctv\_service.get\_camera\_status()

494. self.speak(f"Camera status: {status}")

495.

496. elif intent == "capture\_snapshot":

497. camera\_id = self.nlu\_system.extract\_entities(command)['camera\_id']

498. snapshot\_response = self.swann\_cctv\_service.capture\_snapshot(camera\_id)

499. self.speak(snapshot\_response)

500.

501. elif intent == "start\_recording":

502. camera\_id = self.nlu\_system.extract\_entities(command)['camera\_id']

503. success = self.swann\_cctv\_service.start\_recording(camera\_id)

504. response = "Recording started." if success else "Failed to start recording."

505. self.speak(response)

506.

507. elif intent == "stop\_recording":

508. camera\_id = self.nlu\_system.extract\_entities(command)['camera\_id']

509. success = self.swann\_cctv\_service.stop\_recording(camera\_id)

510. response = "Recording stopped." if success else "Failed to stop recording."

511. self.speak(response)

512.

513. if intent == "capture\_snapshot":

514. snapshot\_response = self.ip\_camera\_service.capture\_snapshot()

515. self.speak(snapshot\_response)

516.

517. elif intent == "start\_recording":

518. success\_response = self.ip\_camera\_service.start\_recording()

519. self.speak(success\_response)

520.

521. elif intent == "stop\_recording":

522. stop\_response = self.ip\_camera\_service.stop\_recording()

523. self.speak(stop\_response)

524.

525. if intent == "get\_nutrition":

526. food\_item = self.nlu\_system.extract\_entities(command)['food\_item']

527. nutrition\_info = self.food\_service.get\_nutrition\_facts(food\_item)

528. self.speak(nutrition\_info)

529.

530. elif intent == "find\_recipes":

531. query = self.nlu\_system.extract\_entities(command)['ingredient']

532. recipes = self.food\_service.find\_recipes(query)

533. self.speak(recipes)

534.

535. elif intent == "get\_recipe\_instructions":

536. recipe\_id = self.nlu\_system.extract\_entities(command)['recipe\_id']

537. instructions = self.food\_service.get\_recipe\_instructions(recipe\_id)

538. self.speak(instructions)

539.

540. if intent == "analyze\_file":

541. file\_path = self.nlu\_system.extract\_entities(command)['file\_path']

542. analysis\_result = self.file\_analysis\_service.file\_analysis(file\_path)

543. self.speak(analysis\_result)

544.

545. if "translate" in command:

546. text\_to\_translate = command.replace("translate", "").strip()

547. translated\_text = self.translation\_service.translate\_text(text\_to\_translate, dest\_language='fr') # Translate to French

548. response = f"Translated text: {translated\_text}"

549. self.speak(response)

550.

551. if "analyze text" in command:

552. text\_to\_analyze = command.replace("analyze text", "").strip()

553. analysis\_result = self.spacy\_utils.analyze\_text(text\_to\_analyze)

554. response = f"Tokens: {analysis\_result['tokens']}, Entities: {analysis\_result['entities']}, POS Tags: {analysis\_result['pos\_tags']}"

555. self.speak(response)

556.

557. if "analyze medical text" in command:

558. text\_to\_analyze = command.replace("analyze medical text", "").strip()

559. analysis\_result = self.medical\_nlp.analyze\_medical\_text(text\_to\_analyze)

560. response = f"Entities: {analysis\_result['entities']}, Symptoms Classification: {analysis\_result['symptom\_classification']}"

561. self.speak(response)

562.

563. if "analyze fitness data" in command:

564. # Mock data for demonstration; you can replace this with actual data

565. fitness\_data = {'steps': 10000, 'calories\_burned': 500, 'workout\_duration': 30}

566. analysis\_result = self.fitness\_nlp.analyze\_fitness\_data(fitness\_data)

567. self.speak(analysis\_result)

568.

569. elif "recommend workout" in command:

570. fitness\_level = command.replace("recommend workout", "").strip()

571. recommendation = self.fitness\_nlp.recommend\_workout(fitness\_level)

572. self.speak(recommendation)

573.

574. elif "set fitness goal" in command:

575. goal = command.replace("set fitness goal", "").strip()

576. confirmation = self.fitness\_nlp.set\_fitness\_goal(goal)

577. self.speak(confirmation)

578.

579. if "calculate force" in command:

580. params = self.extract\_params(command) # Define this method to extract mass and acceleration

581. force = self.engineering\_nlp.calculate\_force(params['mass'], params['acceleration'])

582. self.speak(f"The calculated force is {force} Newtons.")

583.

584. elif "calculate work" in command:

585. params = self.extract\_params(command) # Define this method to extract force and distance

586. work = self.engineering\_nlp.calculate\_work(params['force'], params['distance'])

587. self.speak(f"The calculated work is {work} Joules.")

588.

589. elif "calculate energy" in command:

590. params = self.extract\_params(command) # Define this method to extract mass and height

591. energy = self.engineering\_nlp.calculate\_energy(params['mass'], params['height'])

592. self.speak(f"The calculated potential energy is {energy} Joules.")

593.

594. elif "structural analysis" in command:

595. params = self.extract\_params(command) # Define this method to extract load, length, width, height

596. stress = self.engineering\_nlp.structural\_analysis(params['load'], params['length'], params['width'], params['height'])

597. self.speak(f"The bending stress is {stress} Pascals.")

598. if "analyze crime" in command:

599. crime\_type = self.extract\_crime\_type(command) # Define this method to extract the crime type

600. count = self.crime\_analyzer.analyze\_crime\_type(crime\_type)

601. self.speak(f"There are {count} occurrences of {crime\_type}.")

602.

603. elif "top crimes" in command:

604. top\_crimes = self.crime\_analyzer.get\_top\_crime\_types()

605. self.speak(f"The top crimes are: {', '.join(top\_crimes.index)} with counts: {', '.join(map(str, top\_crimes.values))}.")

606.

607. if "generate post" in command:

608. context = {

609. "link": "https://example.com",

610. "event": "our launch event",

611. "product\_name": "Awesome Product",

612. "offer": "50% off this week only",

613. "date": "October 5"

614. }

615. post = self.social\_media\_post\_generator.generate\_post(context)

616. formatted\_post = self.social\_media\_post\_generator.format\_post(post)

617. self.speak(f"Here is your social media post: {formatted\_post}")

618.

619. if "start surveillance" in command.lower():

620. self.surveillance\_system.start\_surveillance()

621.

622. if "start contact recognition" in command.lower():

623. self.contact\_recognition\_system.start\_recognition()

624.

625. if "start car detection" in command.lower():

626. self.car\_model\_system.start\_detection()

627.

628. if "discover bluetooth devices" in command.lower():

629. devices = self.bluetooth\_service.discover\_devices()

630. if devices:

631. response = f"Found {len(devices)} devices: {devices}"

632. else:

633. response = "No Bluetooth devices found."

634. self.speak(response)

635.

636. if "nutrition facts" in command:

637. food\_item = command.replace("nutrition facts for", "").strip()

638. nutrition\_info = self.health\_service.get\_nutrition\_facts(food\_item)

639. response = f"Nutritional info for {food\_item}: {nutrition\_info}"

640. self.speak(response)

641.

642. elif "healthy" in command:

643. food\_item = command.replace("is", "").replace("healthy", "").strip()

644. is\_healthy = self.health\_service.is\_healthy(food\_item)

645. response = f"{food\_item} is {'healthy' if is\_healthy else 'not healthy'}."

646. self.speak(response)

647.

648. elif "recipes" in command:

649. ingredient = command.replace("give me recipes with", "").strip()

650. recipes = self.health\_service.get\_recipes(ingredient)

651. response = f"Recipes with {ingredient}: {recipes}"

652. self.speak(response)

653.

654. if intent == "add\_project":

655. project\_name = entities.get('project\_name', 'Unnamed Project')

656. deadline = entities.get('deadline', 'No deadline provided')

657. description = entities.get('description', 'No description provided')

658. response = self.business\_management\_service.add\_project(project\_name, description, deadline)

659. self.speak(response)

660.

661. elif intent == "track\_finances":

662. income = entities.get('income', 0)

663. expense = entities.get('expense', 0)

664. response = self.business\_management\_service.track\_finances(income=income, expense=expense)

665. self.speak(response)

666.

667. elif intent == "schedule\_meeting":

668. title = entities.get('meeting\_title', 'No title provided')

669. time = entities.get('time', 'No time provided')

670. participants = entities.get('participants', [])

671. response = self.business\_management\_service.add\_meeting(title, time, participants)

672. self.speak(response)

673.

674. elif intent == "add\_performance\_metric":

675. metric\_name = entities.get('metric\_name', 'Unnamed Metric')

676. value = entities.get('value', 0)

677. response = self.business\_management\_service.add\_performance\_metric(metric\_name, value)

678. self.speak(response)

679.

680. elif intent == "get\_business\_summary":

681. summary = self.business\_management\_service.get\_summary()

682. summary\_str = f"Projects: {summary['projects']}\n" \

683. f"Finances: Income - {summary['finances']['income']}, Expenses - {summary['finances']['expenses']}\n" \

684. f"Meetings: {summary['meetings']}\n" \

685. f"Performance Metrics: {summary['performance\_metrics']}"

686. self.speak(summary\_str)

687.

688.

689.

690. # Handle more intents here as you add services and features

691.

692.

693.

694.

695.

696.

697. def run(self):

698. self.window.mainloop()

699.

700.

701. if \_\_name\_\_ == "\_\_main\_\_":

702. assistant = Jaicat()

703. assistant.run()

704.

## command

### command\Commands.py

**Purpose**: This file likely defines all the commands that Jaicat can execute, whether they are voice-based or text-based commands.

### Key Functions:

* + **Command Definitions**: Could store a list of supported commands and their corresponding functions. It might link certain voice/text inputs to specific actions Jaicat can perform (e.g., play music, fetch weather).
* **Connection to main.py**:
  + The Commands.py file would be called when the NLP system identifies that the user input matches a predefined command. Once the command is identified, Jaicat knows what specific function to execute based on the user request.

#### The Code

1. # command/Commands.py

2.

3. from services.job\_search import JobSearchService

4. from services.email\_service import EmailService

5. from services.finance import FinanceService

6. from services.weather\_service import WeatherService

7. from services.spotify\_integration import SpotifyIntegration

8. from services.travel\_recommendations import TravelRecommendations

9. from services.youtube\_analysis import YouTubeAnalysis

10. from services.food import FoodService

11. from services.file\_analysis import FileAnalysisService

12. from services.project\_management import ProjectManagement

13.

14. class CommandProcessor:

15.     def \_\_init\_\_(self):

16.         # Initialize all services

17.         self.job\_search\_service = JobSearchService()

18.         self.email\_service = EmailService()

19.         self.finance\_service = FinanceService()

20.         self.weather\_service = WeatherService()

21.         self.spotify\_integration = SpotifyIntegration()

22.         self.travel\_recommendations = TravelRecommendations()

23.         self.youtube\_analysis = YouTubeAnalysis()

24.         self.food\_service = FoodService()

25.         self.file\_analysis\_service = FileAnalysisService()

26.         self.project\_management\_service = ProjectManagement()

27.

28.     def process\_command(self, command):

29.         command = command.lower()

30.

31.         if "find jobs" in command:

32.             self.process\_job\_search(command)

33.         elif "send email" in command:

34.             self.process\_send\_email(command)

35.         elif "check finance" in command:

36.             self.process\_finance(command)

37.         elif "get weather" in command:

38.             self.process\_weather(command)

39.         elif "play music" in command:

40.             self.process\_play\_music(command)

41.         elif "travel recommendations" in command:

42.             self.process\_travel\_recommendations(command)

43.         elif "analyze youtube" in command:

44.             self.process\_youtube\_analysis(command)

45.         elif "food recommendations" in command:

46.             self.process\_food\_recommendations(command)

47.         elif "analyze file" in command:

48.             self.process\_file\_analysis(command)

49.         elif "project management" in command:

50.             self.process\_project\_management(command)

51.         else:

52.             print("Command not recognized. Please try again.")

53.

54.     def process\_job\_search(self, command):

55.         query = command.replace("find jobs", "").strip()

56.         jobs = self.job\_search\_service.search\_jobs(query)

57.         if jobs:

58.             response = f"Top job listings: {', '.join(jobs)}"

59.         else:

60.             response = "No jobs found."

61.         print(response)

62.

63.     def process\_send\_email(self, command):

64.         email\_details = command.replace("send email", "").strip()

65.         response = self.email\_service.send\_email(email\_details)

66.         print(response)

67.

68.     def process\_finance(self, command):

69.         query = command.replace("check finance", "").strip()

70.         finance\_info = self.finance\_service.get\_financial\_info(query)

71.         print(finance\_info)

72.

73.     def process\_weather(self, command):

74.         location = command.replace("get weather", "").strip()

75.         weather\_info = self.weather\_service.get\_weather(location)

76.         print(weather\_info)

77.

78.     def process\_play\_music(self, command):

79.         song\_name = command.replace("play music", "").strip()

80.         response = self.spotify\_integration.play\_song(song\_name)

81.         print(response)

82.

83.     def process\_travel\_recommendations(self, command):

84.         query = command.replace("travel recommendations", "").strip()

85.         recommendations = self.travel\_recommendations.get\_recommendations(query)

86.         print(recommendations)

87.

88.     def process\_youtube\_analysis(self, command):

89.         video\_url = command.replace("analyze youtube", "").strip()

90.         analysis\_results = self.youtube\_analysis.analyze\_video(video\_url)

91.         print(analysis\_results)

92.

93.     def process\_food\_recommendations(self, command):

94.         query = command.replace("food recommendations", "").strip()

95.         recommendations = self.food\_service.get\_food\_recommendations(query)

96.         print(recommendations)

97.

98.     def process\_file\_analysis(self, command):

99.         file\_path = command.replace("analyze file", "").strip()

100.         analysis\_results = self.file\_analysis\_service.analyze\_file(file\_path)

101.         print(analysis\_results)

102.

103.     def process\_project\_management(self, command):

104.         project\_details = command.replace("project management", "").strip()

105.         response = self.project\_management\_service.manage\_project(project\_details)

106.         print(response)

107.     def process\_command(self, command):

108.         # Use the NLU system to extract intent and entities

109.         intent = self.nlu\_system.extract\_intent(command)

110.         entities = self.nlu\_system.extract\_entities(command)

111.

112.         if intent == "play\_music":

113.             song\_name = entities.get('song\_name')

114.             if song\_name:

115.                 self.play\_song(song\_name)

116.             else:

117.                 self.speak("Please provide the name of the song.")

118.

119.         elif intent == "check\_calendar":

120.             self.display\_calendar\_events()

121.

122.         elif intent == "check\_weather":

123.             location = entities.get('location', 'New York')  # Default to New York if no location provided

124.             self.check\_weather(location)

125.

126.         elif intent == "check\_stock\_price":

127.             stock\_name = entities.get('stock\_name')

128.             if stock\_name:

129.                 self.check\_stock\_price(stock\_name)

130.             else:

131.                 self.speak("Please provide the name of the stock.")

132.

133.         elif intent == "find\_jobs":

134.             job\_title = entities.get('job\_title')

135.             if job\_title:

136.                 jobs = self.job\_search\_service.search\_jobs(job\_title)

137.                 if jobs:

138.                     self.speak(f"Top job listings: {', '.join(jobs)}")

139.                 else:

140.                     self.speak("No jobs found.")

141.             else:

142.                 self.speak("Please provide the job title you are looking for.")

143.         elif intent == "get\_news":

144.             category = entities.get('category', 'general')

145.         headlines = self.news\_service.get\_news(category=category)

146.         if headlines:

147.             self.speak(f"Here are the top headlines: {', '.join(headlines)}")

148.         else:

149.             self.speak("No news available at the moment.")

150.         if intent == "control\_device":

151.             device\_id = entities.get('device\_id')

152.         action = entities.get('action')

153.         if device\_id and action:

154.             response = self.home\_automation\_service.control\_device(device\_id, action)

155.             self.speak(response)

156.         else:

157.             self.speak("Please specify the device and action.")

158.         if intent == "check\_stock\_price":

159.             stock\_symbol = entities.get('stock\_symbol')

160.         if stock\_symbol:

161.             price = self.finance\_service.get\_stock\_price(stock\_symbol)

162.             self.speak(f"The current price of {stock\_symbol} is {price}")

163.         else:

164.             self.speak("Please provide a stock symbol.")

165.

166.         if intent == "send\_email":

167.             recipient = entities.get('recipient')

168.             subject = entities.get('subject', 'No Subject')

169.             body = entities.get('body', 'No Content')

170.         if recipient:

171.             response = self.email\_service.send\_email(recipient, subject, body)

172.             self.speak(response)

173.         else:

174.             self.speak("Please provide the recipient's email address.")

175.

176.         if intent == "analyze\_youtube":

177.                 video\_url = self.nlu\_system.extract\_entities(command)['video\_url']

178.                 self.analyze\_youtube\_video(video\_url)

179.         if intent == "get\_travel\_recommendations":

180.             category = self.nlu\_system.extract\_entities(command).get('category', '')

181.             recommendations = self.travel\_recommendations\_service.get\_recommendations(category)

182.             response = f"Travel recommendations: {', '.join(recommendations)}"

183.             self.speak(response)

184.         if intent == "capture\_image":

185.             self.usb\_cam\_service.start\_camera()

186.             self.usb\_cam\_service.capture\_image("captured\_image.png")  # Specify filename as needed

187.             self.usb\_cam\_service.release\_camera()

188.             response = "Image captured successfully."

189.             self.speak(response)

190.         if intent == "create\_project":

191.             project\_name = self.nlu\_system.extract\_entities(command).get('project\_name', 'Default Project')

192.             response = self.project\_management\_service.create\_project(project\_name)

193.             self.speak(response)

194.

195.         elif intent == "add\_task":

196.             project\_name = self.nlu\_system.extract\_entities(command).get('project\_name', 'Default Project')

197.             task = self.nlu\_system.extract\_entities(command).get('task', 'New Task')

198.             response = self.project\_management\_service.add\_task(project\_name, task)

199.             self.speak(response)

200.

201.         elif intent == "list\_projects":

202.             response = self.project\_management\_service.list\_projects()

203.             self.speak(response)

204.

205.         elif intent == "complete\_project":

206.             project\_name = self.nlu\_system.extract\_entities(command).get('project\_name', 'Default Project')

207.             response = self.project\_management\_service.complete\_project(project\_name)

208.             self.speak(response)

209.

210.         if intent == "capture\_image":

211.             response = self.phone\_camera\_service.capture\_image()

212.             self.speak(response)

213.         if intent == "get\_camera\_status":

214.             status = self.swann\_cctv\_service.get\_camera\_status()

215.             self.speak(f"Camera status: {status}")

216.

217.         elif intent == "capture\_snapshot":

218.             camera\_id = self.nlu\_system.extract\_entities(command)['camera\_id']

219.             snapshot\_response = self.swann\_cctv\_service.capture\_snapshot(camera\_id)

220.             self.speak(snapshot\_response)

221.

222.         elif intent == "start\_recording":

223.             camera\_id = self.nlu\_system.extract\_entities(command)['camera\_id']

224.             success = self.swann\_cctv\_service.start\_recording(camera\_id)

225.             response = "Recording started." if success else "Failed to start recording."

226.             self.speak(response)

227.

228.         elif intent == "stop\_recording":

229.             camera\_id = self.nlu\_system.extract\_entities(command)['camera\_id']

230.             success = self.swann\_cctv\_service.stop\_recording(camera\_id)

231.             response = "Recording stopped." if success else "Failed to stop recording."

232.             self.speak(response)

233.

234.         if intent == "capture\_snapshot":

235.             snapshot\_response = self.ip\_camera\_service.capture\_snapshot()

236.             self.speak(snapshot\_response)

237.

238.         elif intent == "start\_recording":

239.             success\_response = self.ip\_camera\_service.start\_recording()

240.             self.speak(success\_response)

241.

242.         elif intent == "stop\_recording":

243.             stop\_response = self.ip\_camera\_service.stop\_recording()

244.             self.speak(stop\_response)

245.

246.         if intent == "get\_nutrition":

247.             food\_item = self.nlu\_system.extract\_entities(command)['food\_item']

248.             nutrition\_info = self.food\_service.get\_nutrition\_facts(food\_item)

249.             self.speak(nutrition\_info)

250.

251.         elif intent == "find\_recipes":

252.             query = self.nlu\_system.extract\_entities(command)['ingredient']

253.             recipes = self.food\_service.find\_recipes(query)

254.             self.speak(recipes)

255.

256.         elif intent == "get\_recipe\_instructions":

257.             recipe\_id = self.nlu\_system.extract\_entities(command)['recipe\_id']

258.             instructions = self.food\_service.get\_recipe\_instructions(recipe\_id)

259.             self.speak(instructions)

260.

261.         if intent == "analyze\_file":

262.             file\_path = self.nlu\_system.extract\_entities(command)['file\_path']

263.             analysis\_result = self.file\_analysis\_service.file\_analysis(file\_path)

264.             self.speak(analysis\_result)

265.

266.         if "translate" in command:

267.             text\_to\_translate = command.replace("translate", "").strip()

268.             translated\_text = self.translation\_service.translate\_text(text\_to\_translate, dest\_language='fr')  # Translate to French

269.             response = f"Translated text: {translated\_text}"

270.             self.speak(response)

271.

272.         if "analyze text" in command:

273.             text\_to\_analyze = command.replace("analyze text", "").strip()

274.             analysis\_result = self.spacy\_utils.analyze\_text(text\_to\_analyze)

275.             response = f"Tokens: {analysis\_result['tokens']}, Entities: {analysis\_result['entities']}, POS Tags: {analysis\_result['pos\_tags']}"

276.             self.speak(response)

277.

278.         if "analyze medical text" in command:

279.             text\_to\_analyze = command.replace("analyze medical text", "").strip()

280.             analysis\_result = self.medical\_nlp.analyze\_medical\_text(text\_to\_analyze)

281.             response = f"Entities: {analysis\_result['entities']}, Symptoms Classification: {analysis\_result['symptom\_classification']}"

282.             self.speak(response)

283.

284.         if "analyze fitness data" in command:

285.             # Mock data for demonstration; you can replace this with actual data

286.             fitness\_data = {'steps': 10000, 'calories\_burned': 500, 'workout\_duration': 30}

287.             analysis\_result = self.fitness\_nlp.analyze\_fitness\_data(fitness\_data)

288.             self.speak(analysis\_result)

289.

290.         elif "recommend workout" in command:

291.             fitness\_level = command.replace("recommend workout", "").strip()

292.             recommendation = self.fitness\_nlp.recommend\_workout(fitness\_level)

293.             self.speak(recommendation)

294.

295.         elif "set fitness goal" in command:

296.             goal = command.replace("set fitness goal", "").strip()

297.             confirmation = self.fitness\_nlp.set\_fitness\_goal(goal)

298.             self.speak(confirmation)

299.

300.         if "calculate force" in command:

301.             params = self.extract\_params(command)  # Define this method to extract mass and acceleration

302.             force = self.engineering\_nlp.calculate\_force(params['mass'], params['acceleration'])

303.             self.speak(f"The calculated force is {force} Newtons.")

304.

305.         elif "calculate work" in command:

306.             params = self.extract\_params(command)  # Define this method to extract force and distance

307.             work = self.engineering\_nlp.calculate\_work(params['force'], params['distance'])

308.             self.speak(f"The calculated work is {work} Joules.")

309.

310.         elif "calculate energy" in command:

311.             params = self.extract\_params(command)  # Define this method to extract mass and height

312.             energy = self.engineering\_nlp.calculate\_energy(params['mass'], params['height'])

313.             self.speak(f"The calculated potential energy is {energy} Joules.")

314.

315.         elif "structural analysis" in command:

316.             params = self.extract\_params(command)  # Define this method to extract load, length, width, height

317.             stress = self.engineering\_nlp.structural\_analysis(params['load'], params['length'], params['width'], params['height'])

318.             self.speak(f"The bending stress is {stress} Pascals.")

319.         if "analyze crime" in command:

320.             crime\_type = self.extract\_crime\_type(command)  # Define this method to extract the crime type

321.             count = self.crime\_analyzer.analyze\_crime\_type(crime\_type)

322.             self.speak(f"There are {count} occurrences of {crime\_type}.")

323.

324.         elif "top crimes" in command:

325.             top\_crimes = self.crime\_analyzer.get\_top\_crime\_types()

326.             self.speak(f"The top crimes are: {', '.join(top\_crimes.index)} with counts: {', '.join(map(str, top\_crimes.values))}.")

327.

328.         if "generate post" in command:

329.             context = {

330.                 "link": "https://example.com",

331.                 "event": "our launch event",

332.                 "product\_name": "Awesome Product",

333.                 "offer": "50% off this week only",

334.                 "date": "October 5"

335.             }

336.             post = self.social\_media\_post\_generator.generate\_post(context)

337.             formatted\_post = self.social\_media\_post\_generator.format\_post(post)

338.             self.speak(f"Here is your social media post: {formatted\_post}")

339.

340.         if "start surveillance" in command.lower():

341.             self.surveillance\_system.start\_surveillance()

342.

343.         if "start contact recognition" in command.lower():

344.             self.contact\_recognition\_system.start\_recognition()

345.

346.         if "start car detection" in command.lower():

347.             self.car\_model\_system.start\_detection()

348.

349.         if "discover bluetooth devices" in command.lower():

350.                 devices = self.bluetooth\_service.discover\_devices()

351.                 if devices:

352.                     response = f"Found {len(devices)} devices: {devices}"

353.                 else:

354.                     response = "No Bluetooth devices found."

355.                 self.speak(response)

356.

357.         if "nutrition facts" in command:

358.             food\_item = command.replace("nutrition facts for", "").strip()

359.             nutrition\_info = self.health\_service.get\_nutrition\_facts(food\_item)

360.             response = f"Nutritional info for {food\_item}: {nutrition\_info}"

361.             self.speak(response)

362.

363.         elif "healthy" in command:

364.             food\_item = command.replace("is", "").replace("healthy", "").strip()

365.             is\_healthy = self.health\_service.is\_healthy(food\_item)

366.             response = f"{food\_item} is {'healthy' if is\_healthy else 'not healthy'}."

367.             self.speak(response)

368.

369.         elif "recipes" in command:

370.             ingredient = command.replace("give me recipes with", "").strip()

371.             recipes = self.health\_service.get\_recipes(ingredient)

372.             response = f"Recipes with {ingredient}: {recipes}"

373.             self.speak(response)

374.

375.         if intent == "add\_project":

376.                 project\_name = entities.get('project\_name', 'Unnamed Project')

377.                 deadline = entities.get('deadline', 'No deadline provided')

378.                 description = entities.get('description', 'No description provided')

379.                 response = self.business\_management\_service.add\_project(project\_name, description, deadline)

380.                 self.speak(response)

381.

382.         elif intent == "track\_finances":

383.                 income = entities.get('income', 0)

384.                 expense = entities.get('expense', 0)

385.                 response = self.business\_management\_service.track\_finances(income=income, expense=expense)

386.                 self.speak(response)

387.

388.         elif intent == "schedule\_meeting":

389.                 title = entities.get('meeting\_title', 'No title provided')

390.                 time = entities.get('time', 'No time provided')

391.                 participants = entities.get('participants', [])

392.                 response = self.business\_management\_service.add\_meeting(title, time, participants)

393.                 self.speak(response)

394.

395.         elif intent == "add\_performance\_metric":

396.                 metric\_name = entities.get('metric\_name', 'Unnamed Metric')

397.                 value = entities.get('value', 0)

398.                 response = self.business\_management\_service.add\_performance\_metric(metric\_name, value)

399.                 self.speak(response)

400.

401.         elif intent == "get\_business\_summary":

402.                 summary = self.business\_management\_service.get\_summary()

403.                 summary\_str = f"Projects: {summary['projects']}\n" \

404.                             f"Finances: Income - {summary['finances']['income']}, Expenses - {summary['finances']['expenses']}\n" \

405.                             f"Meetings: {summary['meetings']}\n" \

406.                             f"Performance Metrics: {summary['performance\_metrics']}"

407.                 self.speak(summary\_str)

408.

409.         if "add car registration" in command:

410.             new\_car\_reg = self.get\_voice\_input("Please provide the new car registration plate:")

411.             self.update\_user\_data("car\_reg", new\_car\_reg)

412.             self.speak(f"Added new car registration {new\_car\_reg} to your profile.")

413.

414.         elif "add motorbike registration" in command:

415.             new\_motorbike\_reg = self.get\_voice\_input("Please provide the new motorbike registration plate:")

416.             self.update\_user\_data("motorbike\_reg", new\_motorbike\_reg)

417.             self.speak(f"Added new motorbike registration {new\_motorbike\_reg} to your profile.")

418.         if "add vehicle" in command:

419.             self.assistant.enroll\_vehicle()

420.

421.         elif "vehicle details" in command:

422.             reg\_plate = self.extract\_registration(command)

423.             vehicle\_info = self.assistant.vehicle\_lookup\_service.lookup\_vehicle(reg\_plate)

424.             self.assistant.ui.speak(f"Vehicle details: {vehicle\_info}")

425.

426.         elif "detected vehicle" in command:

427.             detected\_vehicle = self.assistant.surveillance\_system.detect\_vehicle()

428.             self.assistant.recognize\_vehicle\_plate(detected\_vehicle)

429.

430. if \_\_name\_\_ == "\_\_main\_\_":

431.     command\_processor = CommandProcessor()

432.     while True:

433.         user\_command = input("Enter your command: ")

434.         command\_processor.process\_command(user\_command)

435.

436.

### command\Commands.txt

**Purpose**: Likely a text file that contains a plain-text list of all available commands for documentation or debugging purposes.

* **Connection to main.py**:
  + No direct connection to main.py. This is most likely used for reference or documentation.

## computer\_vision

### computer\_vision\ car\_detection.py

##### The code

1. import cv2

2. import torch

3.

4. class CarDetection:

5.     CAR\_CLASS\_ID = 2

6.

7.     def \_\_init\_\_(self, weights\_path: str):

8.         try:

9.             self.model = torch.hub.load('WongKinYiu/yolov7', 'custom', weights\_path, source='local')

10.         except Exception as e:

11.             print(f"Error loading model: {e}")

12.

13.     def detect\_cars(self, frame: cv2.Mat) -> list:

14.         """

15.         Detect cars in a given frame.

16.

17.         Args:

18.             frame (cv2.Mat): Input frame.

19.

20.         Returns:

21.             list: List of detected cars, each represented as a dictionary containing coordinates and confidence.

22.         """

23.         if frame is None:

24.             raise ValueError("Invalid input frame")

25.

26.         results = self.model(frame)

27.         detections = results.xyxy[0]  # Get detections

28.

29.         detected\_cars = []

30.         for \*xyxy, conf, cls in detections:

31.             if int(cls) == self.CAR\_CLASS\_ID:

32.                 detected\_cars.append({

33.                     'coordinates': [int(x) for x in xyxy],

34.                     'confidence': conf.item()

35.                 })

36.         return detected\_cars

37.

### computer\_vision\car\_model.py

**Purpose**: This file probably handles the recognition of car models from images using machine learning or image classification models.

#### Key Functions:

* + **Model Training/Inference**: It could use machine learning models trained on different car images to classify the make and model.
* **Connection to main.py**:
  + If Jaicat includes a feature to identify cars via camera input, main.py would reference this file when the user asks Jaicat to recognize a car model.

#### The code

1. import cv2

2. import numpy as np

3. import pytesseract

4. import requests

5. import os

6. import json

7.

8. class UKCarModel:

9.     def \_\_init\_\_(self, yolo\_weights: str, yolo\_cfg: str, yolo\_names: str, plate\_cascade\_path: str):

10.         """

11.         Initialize the YOLO model and UK-specific plate detection.

12.         Args:

13.             yolo\_weights (str): Path to YOLO weights file (e.g., 'computer\_vision/yolov7.weights').

14.             yolo\_cfg (str): Path to YOLO configuration file (e.g., 'computer\_vision/yolov7.cfg').

15.             yolo\_names (str): Path to COCO names file (e.g., 'computer\_vision/coco.names').

16.             plate\_cascade\_path (str): Path to the UK-specific Haar Cascade file for license plates.

17.         """

18.         try:

19.             # Initialize YOLO model

20.             self.net = cv2.dnn.readNet(yolo\_weights, yolo\_cfg)

21.             self.layer\_names = self.net.getLayerNames()

22.             self.output\_layers = [self.layer\_names[i - 1] for i in self.net.getUnconnectedOutLayers()]

23.

24.             # Load COCO classes

25.             with open(yolo\_names, "r") as f:

26.                 self.classes = [line.strip() for line in f.readlines()]

27.

28.             # Load UK-specific Haar Cascade for license plates

29.             self.plate\_cascade = cv2.CascadeClassifier(plate\_cascade\_path)

30.

31.             # Initialize detection save path

32.             self.detections\_save\_path = "computer\_vision/detections.json"

33.             if not os.path.exists(self.detections\_save\_path):

34.                 with open(self.detections\_save\_path, 'w') as file:

35.                     json.dump([], file)  # Initialize with an empty list

36.

37.             # Initialize the local dataset file if it doesn't exist

38.             self.local\_data\_path = "computer\_vision/local\_car\_data.json"

39.             if not os.path.exists(self.local\_data\_path):

40.                 with open(self.local\_data\_path, 'w') as file:

41.                     json.dump({}, file)

42.

43.         except Exception as e:

44.             print(f"Error initializing model: {e}")

45.

46.     def detect\_cars(self, frame: np.ndarray) -> list:

47.         """

48.         Detect cars in a given frame.

49.         Args:

50.             frame (np.ndarray): Input frame.

51.         Returns:

52.             list: List of detected cars with coordinates and confidence.

53.         """

54.         height, width = frame.shape[:2]

55.         blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0), True, crop=False)

56.         self.net.setInput(blob)

57.         outs = self.net.forward(self.output\_layers)

58.

59.         class\_ids, confidences, boxes = [], [], []

60.

61.         # Filter and detect cars (assuming 'car' is class ID 2)

62.         for out in outs:

63.             for detection in out:

64.                 scores = detection[5:]

65.                 class\_id = np.argmax(scores)

66.                 confidence = scores[class\_id]

67.

68.                 if confidence > 0.5 and class\_id == 2:  # Only detect cars (COCO ID for car is 2)

69.                     center\_x = int(detection[0] \* width)

70.                     center\_y = int(detection[1] \* height)

71.                     w = int(detection[2] \* width)

72.                     h = int(detection[3] \* height)

73.

74.                     x = int(center\_x - w / 2)

75.                     y = int(center\_y - h / 2)

76.

77.                     boxes.append([x, y, w, h])

78.                     confidences.append(float(confidence))

79.                     class\_ids.append(class\_id)

80.

81.         indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

82.         return [(boxes[i], confidences[i]) for i in indexes.flatten()]

83.

84.     def detect\_license\_plate(self, frame: np.ndarray) -> str:

85.         """

86.         Detect UK license plate in a frame.

87.         Args:

88.             frame (np.ndarray): Input frame.

89.         Returns:

90.             str: Extracted license plate text.

91.         """

92.         gray\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

93.         plates = self.plate\_cascade.detectMultiScale(gray\_frame, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

94.

95.         for (x, y, w, h) in plates:

96.             plate\_img = frame[y:y + h, x:x + w]

97.             plate\_text = pytesseract.image\_to\_string(plate\_img, config='--psm 8')

98.             return plate\_text.strip()

99.         return None

100.

101.     def get\_vehicle\_data(self, plate\_number: str) -> dict:

102.         """

103.         Retrieve vehicle data for a given license plate from a local dataset.

104.         Args:

105.             plate\_number (str): License plate number.

106.         Returns:

107.             dict: Vehicle information.

108.         """

109.         try:

110.             with open(self.local\_data\_path, 'r') as file:

111.                 data = json.load(file)

112.             return data.get(plate\_number, {})

113.         except Exception as e:

114.             print(f"Error retrieving vehicle data: {e}")

115.             return {}

116.

117.     def save\_to\_local\_data(self, plate\_number: str, vehicle\_data: dict) -> None:

118.         """

119.         Save the detected vehicle information into a local JSON dataset.

120.         Args:

121.             plate\_number (str): License plate number.

122.             vehicle\_data (dict): Vehicle information.

123.         """

124.         try:

125.             with open(self.local\_data\_path, 'r') as file:

126.                 data = json.load(file)

127.

128.             data[plate\_number] = vehicle\_data

129.

130.             with open(self.local\_data\_path, 'w') as file:

131.                 json.dump(data, file, indent=4)

132.         except Exception as e:

133.             print(f"Error saving vehicle data: {e}")

134.

135.     def save\_detection(self, detection\_data: dict) -> None:

136.         """

137.         Save detected cars and license plates to a local JSON file.

138.         Args:

139.             detection\_data (dict): Dictionary containing detection data.

140.         """

141.         try:

142.             with open(self.detections\_save\_path, 'r') as file:

143.                 data = json.load(file)

144.

145.             data.append(detection\_data)

146.

147.             with open(self.detections\_save\_path, 'w') as file:

148.                 json.dump(data, file, indent=4)

149.             print(f"Saved detection: {detection\_data}")

150.         except Exception as e:

151.             print(f"Error saving detection: {e}")

152.

153.     def start\_detection(self):

154.         """

155.         Start car detection and license plate recognition using webcam.

156.         """

157.         cap = cv2.VideoCapture(0)

158.         while True:

159.             ret, frame = cap.read()

160.             if not ret:

161.                 print("Error: Unable to capture video.")

162.                 break

163.

164.             # Detect cars

165.             detected\_cars = self.detect\_cars(frame)

166.

167.             # Detect license plate

168.             license\_plate = self.detect\_license\_plate(frame)

169.             if license\_plate:

170.                 print(f"Detected License Plate: {license\_plate}")

171.                 vehicle\_data = self.get\_vehicle\_data(license\_plate)

172.                 if not vehicle\_data:

173.                     vehicle\_data = {"model": "Unknown", "manufacturer": "Unknown"}

174.                     self.save\_to\_local\_data(license\_plate, vehicle\_data)

175.

176.                 # Save detection data

177.                 detection\_data = {

178.                     "license\_plate": license\_plate,

179.                     "vehicle\_data": vehicle\_data,

180.                     "detected\_cars": [{"box": box, "confidence": conf} for box, conf in detected\_cars],

181.                 }

182.                 self.save\_detection(detection\_data)

183.

184.             # Display the frame with bounding boxes

185.             for (box, confidence) in detected\_cars:

186.                 x, y, w, h = box

187.                 cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

188.                 cv2.putText(frame, f"Car: {confidence:.2f}", (x, y - 5), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)

189.

190.             cv2.imshow("Car and License Plate Detection", frame)

191.

192.             if cv2.waitKey(1) & 0xFF == ord('q'):

193.                 break

194.

195.         cap.release()

196.         cv2.destroyAllWindows()

197.

198. if \_\_name\_\_ == "\_\_main\_\_":

199.     # Paths for YOLO model weights, config, COCO names, and UK plate cascade

200.     yolo\_weights = "computer\_vision/yolov7.weights"  # Path to YOLO weights

201.     yolo\_cfg = "computer\_vision/yolov7.cfg"          # Path to YOLO config

202.     coco\_names = "computer\_vision/coco.names"        # Path to COCO names

203.     uk\_plate\_cascade = "computer\_vision/uk\_plate\_cascade.xml"  # Path to UK plate cascade

204.

205.     car\_model = UKCarModel(yolo\_weights, yolo\_cfg, coco\_names, uk\_plate\_cascade)

206.     car\_model.start\_detection()

207.

208.

### computer\_vision\car\_part\_recognition.py

1. import cv2

2. import numpy as np

3.

4. class CarPartRecognition:

5.     def \_\_init\_\_(self, weights\_path, config\_path, names\_path):

6.         try:

7.             self.model = cv2.dnn.readNet(weights\_path, config\_path)

8.         except cv2.error as e:

9.             print(f"Error loading YOLO model: {e}")

10.             # You can also try to load a different configuration file or model here

11.             self.model = None

12.

13.     def detect\_parts(self, frame):

14.         height, width = frame.shape[:2]

15.         blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0), True, crop=False)

16.         self.net.setInput(blob)

17.         layer\_names = self.net.getLayerNames()

18.         output\_layers = [layer\_names[i - 1] for i in self.net.getUnconnectedOutLayers()]

19.         outs = self.net.forward(output\_layers)

20.

21.         boxes, confidences, class\_ids = [], [], []

22.

23.         for out in outs:

24.             for detection in out:

25.                 scores = detection[5:]

26.                 class\_id = np.argmax(scores)

27.                 confidence = scores[class\_id]

28.                 if confidence > 0.5:

29.                     center\_x, center\_y, w, h = (detection[0:4] \* np.array([width, height, width, height])).astype('int')

30.                     x = int(center\_x - w / 2)

31.                     y = int(center\_y - h / 2)

32.                     boxes.append([x, y, w, h])

33.                     confidences.append(float(confidence))

34.                     class\_ids.append(class\_id)

35.

36.         return boxes, confidences, class\_ids

37.

### computer\_vision\coco.names.txt

### computer\_vision\contact\_recognition.py

**Purpose**: Likely used for recognizing contacts or known individuals using face recognition or another form of identity recognition.

#### Key Functions:

* + **Face/Contact Identification**: Recognizes or verifies known individuals based on the image or video input.
* **Connection to main.py**:
  + This file could be used in conjunction with the face recognition system already present in main.py, especially if Jaicat is asked to recognize or identify specific contacts from images or videos.

#### The Code

1. # computer\_vision/contact\_recognition.py

2.

3. import cv2

4. import face\_recognition

5. import os

6.

7. class ContactRecognition:

8.     def \_\_init\_\_(self, enrollment\_folder='enrollment\_pictures'):

9.         self.enrollment\_folder = enrollment\_folder

10.         self.known\_face\_encodings = []

11.         self.known\_face\_names = []

12.         self.load\_enrolled\_faces()

13.

14.     def load\_enrolled\_faces(self):

15.         """Load enrolled faces from the specified directory."""

16.         print("Loading enrolled faces...")

17.         for image\_file in os.listdir(self.enrollment\_folder):

18.             if image\_file.endswith(".png") or image\_file.endswith(".jpg"):

19.                 image\_path = os.path.join(self.enrollment\_folder, image\_file)

20.                 image = face\_recognition.load\_image\_file(image\_path)

21.                 encoding = face\_recognition.face\_encodings(image)[0]

22.                 self.known\_face\_encodings.append(encoding)

23.                 self.known\_face\_names.append(os.path.splitext(image\_file)[0])  # Use filename without extension as name

24.

25.     def recognize\_faces(self, frame):

26.         """Recognize faces in the given frame."""

27.         rgb\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

28.         face\_locations = face\_recognition.face\_locations(rgb\_frame)

29.         face\_encodings = face\_recognition.face\_encodings(rgb\_frame, face\_locations)

30.

31.         recognized\_names = []

32.         for encoding in face\_encodings:

33.             matches = face\_recognition.compare\_faces(self.known\_face\_encodings, encoding)

34.             name = "Unknown"

35.

36.             # Check if a match was found

37.             if True in matches:

38.                 first\_match\_index = matches.index(True)

39.                 name = self.known\_face\_names[first\_match\_index]

40.

41.             recognized\_names.append(name)

42.

43.         return face\_locations, recognized\_names

44.

45.     def start\_recognition(self):

46.         """Start the face recognition process."""

47.         print("Starting contact recognition...")

48.         video\_capture = cv2.VideoCapture(0)

49.

50.         while True:

51.             ret, frame = video\_capture.read()

52.             if not ret:

53.                 print("Error: Unable to capture video.")

54.                 break

55.

56.             face\_locations, recognized\_names = self.recognize\_faces(frame)

57.

58.             # Display the results

59.             for (top, right, bottom, left), name in zip(face\_locations, recognized\_names):

60.                 cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 0), 2)

61.                 cv2.putText(frame, name, (left, top - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)

62.

63.             cv2.imshow('Contact Recognition', frame)

64.

65.             # Break the loop on 'q' key press

66.             if cv2.waitKey(1) & 0xFF == ord('q'):

67.                 break

68.

69.         video\_capture.release()

70.         cv2.destroyAllWindows()

71.

72. if \_\_name\_\_ == "\_\_main\_\_":

73.     contact\_recognition = ContactRecognition()

74.     contact\_recognition.start\_recognition()

75.

76.

### computer\_vision\face\_recognition.py

1. import cv2

2. import face\_recognition

3. import os

4.

5. class FaceRecognition:

6.     def \_\_init\_\_(self, enrollment\_pictures\_path):

7.         self.enrollment\_pictures\_path = enrollment\_pictures\_path

8.         self.known\_face\_encodings = []

9.         self.known\_face\_names = []

10.

11.         self.load\_known\_faces()

12.

13.     def load\_known\_faces(self):

14.         for image\_file in os.listdir(self.enrollment\_pictures\_path):

15.             if image\_file.endswith('.png'):

16.                 image = face\_recognition.load\_image\_file(os.path.join(self.enrollment\_pictures\_path, image\_file))

17.                 encoding = face\_recognition.face\_encodings(image)[0]

18.                 self.known\_face\_encodings.append(encoding)

19.                 self.known\_face\_names.append(os.path.splitext(image\_file)[0])

20.

21.     def recognize\_faces(self, frame):

22.         rgb\_frame = frame[:, :, ::-1]

23.         face\_locations = face\_recognition.face\_locations(rgb\_frame)

24.         face\_encodings = face\_recognition.face\_encodings(rgb\_frame, face\_locations)

25.         face\_names = []

26.

27.         for face\_encoding in face\_encodings:

28.             matches = face\_recognition.compare\_faces(self.known\_face\_encodings, face\_encoding)

29.             name = "Unknown"

30.

31.             if True in matches:

32.                 first\_match\_index = matches.index(True)

33.                 name = self.known\_face\_names[first\_match\_index]

34.

35.             face\_names.append(name)

36.

37.         return face\_locations, face\_names

38.

39.

### Computer\_vision\home\_surveillance.py

**Purpose**: This file likely sets up home surveillance features, using a camera to monitor activities or detect motion/events.

#### Key Functions:

* + **Surveillance Monitoring**: Continuously captures footage or detects motion and notifies the user of any detected events.
* **Connection to main.py**:
  + If Jaicat is intended to monitor a home environment, main.py would integrate this file to provide real-time surveillance updates or alerts.

#### The Code

1. # computer\_vision/home\_surveillance.py

2.

3. import cv2

4. import time

5.

6. class HomeSurveillance:

7.     def \_\_init\_\_(self, camera\_index=0):

8.         # Initialize the camera

9.         self.camera\_index = camera\_index

10.         self.video\_capture = cv2.VideoCapture(self.camera\_index)

11.         self.prev\_frame = None

12.

13.     def detect\_motion(self, frame):

14.         """Detect motion in the current frame compared to the previous frame."""

15.         # Convert the frame to grayscale

16.         gray\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

17.         gray\_frame = cv2.GaussianBlur(gray\_frame, (21, 21), 0)

18.

19.         # If the previous frame is None, initialize it

20.         if self.prev\_frame is None:

21.             self.prev\_frame = gray\_frame

22.             return False

23.

24.         # Compute the absolute difference between the current frame and previous frame

25.         frame\_delta = cv2.absdiff(self.prev\_frame, gray\_frame)

26.         thresh = cv2.threshold(frame\_delta, 25, 255, cv2.THRESH\_BINARY)[1]

27.

28.         # Dilate the thresholded image to fill in holes

29.         thresh = cv2.dilate(thresh, None, iterations=2)

30.

31.         # Find contours of the thresholded image

32.         contours, \_ = cv2.findContours(thresh.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

33.

34.         for contour in contours:

35.             if cv2.contourArea(contour) < 500:  # Minimum area to consider for motion

36.                 continue

37.             return True  # Motion detected

38.

39.         self.prev\_frame = gray\_frame

40.         return False

41.

42.     def start\_surveillance(self):

43.         """Start the home surveillance system."""

44.         print("Starting home surveillance...")

45.         while True:

46.             ret, frame = self.video\_capture.read()

47.             if not ret:

48.                 print("Error: Unable to capture video.")

49.                 break

50.

51.             motion\_detected = self.detect\_motion(frame)

52.             if motion\_detected:

53.                 cv2.putText(frame, "Motion Detected!", (10, 30),

54.                             cv2.FONT\_HERSHEY\_SIMPLEX, 0.8, (0, 255, 0), 2)

55.                 print("Motion detected!")

56.

57.             # Display the resulting frame

58.             cv2.imshow("Home Surveillance", frame)

59.

60.             # Break the loop on 'q' key press

61.             if cv2.waitKey(1) & 0xFF == ord('q'):

62.                 break

63.

64.         # Release the capture and close windows

65.         self.video\_capture.release()

66.         cv2.destroyAllWindows()

67.

68. if \_\_name\_\_ == "\_\_main\_\_":

69.     surveillance\_system = HomeSurveillance()

70.     surveillance\_system.start\_surveillance()

71.

72.

### computer\_vision\license\_plate\_detection.py

1. import cv2

2.

3. class LicensePlateDetection:

4.     def \_\_init\_\_(self):

5.         # Load any necessary models or configurations

6.         self.model = self.load\_model()

7.

8.     def load\_model(self):

9.         # Load a pre-trained model for license plate detection

10.         # Example: using OpenCV DNN module or a custom model

11.         pass

12.

13.     def detect(self, image):

14.         # Perform license plate detection on the input image

15.         # Return detected license plates

16.         pass

17.

18.     def highlight\_plate(self, image, plate):

19.         # Draw bounding boxes around detected plates

20.         pass

21.

### computer\_vision\motorbike\_model.py

##### The Code

1. import cv2

2. import numpy as np

3.

4. class MotorbikeModel:

5.     def \_\_init\_\_(self, yolo\_weights, yolo\_cfg, yolo\_names):

6.         # Load YOLO

7.         self.net = cv2.dnn.readNet(yolo\_weights, yolo\_cfg)

8.         self.layer\_names = self.net.getLayerNames()

9.         self.output\_layers = [self.layer\_names[i[0] - 1] for i in self.net.getUnconnectedOutLayers()]

10.         self.classes = []

11.         with open(yolo\_names, "r") as f:

12.             self.classes = [line.strip() for line in f.readlines()]

13.

14.     def detect\_motorbikes(self, frame):

15.         height, width, \_ = frame.shape

16.         blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0), True, crop=False)

17.         self.net.setInput(blob)

18.         outs = self.net.forward(self.output\_layers)

19.

20.         class\_ids = []

21.         confidences = []

22.         boxes = []

23.

24.         # Detect motorbikes

25.         for out in outs:

26.             for detection in out:

27.                 scores = detection[5:]

28.                 class\_id = np.argmax(scores)

29.                 confidence = scores[class\_id]

30.                 if confidence > 0.5 and class\_id == 3:  # '3' is typically the class ID for 'motorbike' in YOLO

31.                     center\_x = int(detection[0] \* width)

32.                     center\_y = int(detection[1] \* height)

33.                     w = int(detection[2] \* width)

34.                     h = int(detection[3] \* height)

35.

36.                     x = int(center\_x - w / 2)

37.                     y = int(center\_y - h / 2)

38.

39.                     boxes.append([x, y, w, h])

40.                     confidences.append(float(confidence))

41.                     class\_ids.append(class\_id)

42.

43.         indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

44.         return [(boxes[i], confidences[i]) for i in indexes.flatten()]

45.

46.     def start\_detection(self):

47.         """Start motorbike detection using webcam."""

48.         cap = cv2.VideoCapture(0)

49.         while True:

50.             ret, frame = cap.read()

51.             if not ret:

52.                 print("Error: Unable to capture video.")

53.                 break

54.

55.             motorbikes = self.detect\_motorbikes(frame)

56.             for (box, confidence) in motorbikes:

57.                 x, y, w, h = box

58.                 label = f"Motorbike: {confidence:.2f}"

59.                 cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

60.                 cv2.putText(frame, label, (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)

61.

62.             cv2.imshow("Motorbike Detection", frame)

63.             if cv2.waitKey(1) & 0xFF == ord('q'):

64.                 break

65.

66.         cap.release()

67.         cv2.destroyAllWindows()

68.

69.

### computer\_vision\object\_detection.py

1. """

2. ObjectDetection class detects objects in a given frame using YOLO.

3.

4. Attributes:

5.     net (cv2.dnn\_Net): The YOLO model.

6.     classes (list[str]): List of class names from the COCO dataset.

7.

8. Methods:

9.     detect\_objects(frame: np.ndarray) -> tuple[list[tuple[int, int, int, int]], list[float], list[int]]:

10.         Detect objects in a given frame.

11.         Args:

12.             frame (np.ndarray): Input frame.

13.         Returns:

14.             tuple[list[tuple[int, int, int, int]], list[float], list[int]]: Detected objects with bounding boxes, confidences, and class IDs.

15. """

16. import cv2

17. import numpy as np

18. import os

19. import unittest

20.

21. class ObjectDetection:

22.     """

23.     Initialize the YOLO model.

24.

25.     Args:

26.         yolo\_weights (str): Path to YOLO weights file (e.g., 'computer\_vision/yolov7.weights').

27.         yolo\_cfg (str): Path to YOLO configuration file (e.g., 'computer\_vision/yolov7.cfg').

28.         coco\_names (str): Path to COCO names file (e.g., 'computer\_vision/coco.names').

29.     """

30.

31.     def \_\_init\_\_(self, yolo\_weights: str, yolo\_cfg: str, coco\_names: str) -> None:

32.         print(f"Loading YOLO model from {yolo\_weights} and {yolo\_cfg}...")

33.         self.net = cv2.dnn.readNetFromDarknet(os.path.abspath(yolo\_cfg), os.path.abspath(yolo\_weights))

34.         if self.net is None:

35.             raise ValueError(f"Failed to load YOLO model from {yolo\_weights} and {yolo\_cfg}")

36.

37.         print(f"Loading COCO class names from {coco\_names}...")

38.         self.classes: list[str] = []

39.         try:

40.             with open(coco\_names, 'r') as f:

41.                 self.classes = [line.strip() for line in f.readlines()]

42.         except OSError as e:

43.             raise ValueError(f"Failed to load COCO class names from {coco\_names}: {e}")

44.

45.         print(f"Loaded {len(self.classes)} class names.")

46.

47.     def detect\_objects(self, frame: np.ndarray) -> tuple[list[tuple[int, int, int, int]], list[float], list[int]]:

48.         """

49.         Detect objects in a given frame.

50.

51.         Args:

52.             frame (np.ndarray): Input frame.

53.

54.         Returns:

55.             tuple[list[tuple[int, int, int, int]], list[float], list[int]]: Detected objects with bounding boxes, confidences, and class IDs.

56.         """

57.         print("Detecting objects...")

58.         if frame is None:

59.             raise ValueError("Input frame is null")

60.

61.         height, width = frame.shape[:2]

62.         print(f"Frame size: {width}x{height}")

63.         blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0), True, crop=False)

64.         self.net.setInput(blob)

65.

66.         layer\_names = self.net.getLayerNames()

67.         output\_layers = [layer\_names[i - 1] for i in self.net.getUnconnectedOutLayers()]

68.         outs = self.net.forward(output\_layers)

69.

70.         boxes: list[tuple[int, int, int, int]] = []

71.         confidences: list[float] = []

72.         class\_ids: list[int] = []

73.

74.         for out in outs:

75.             for detection in out:

76.                 scores = detection[5:]

77.                 class\_id = np.argmax(scores)

78.                 confidence = scores[class\_id]

79.                 if confidence > 0.5:

80.                     center\_x, center\_y, w, h = (detection[0:4] \* np.array([width, height, width, height])).astype('int')

81.                     x = int(center\_x - w / 2)

82.                     y = int(center\_y - h / 2)

83.                     boxes.append((x, y, w, h))

84.                     confidences.append(float(confidence))

85.                     class\_ids.append(class\_id)

86.

87.         return boxes, confidences, class\_ids

88.

89.     def draw\_detected\_objects(self, frame: np.ndarray, boxes: list[tuple[int, int, int, int]], confidences: list[float], class\_ids: list[int]):

90.         """

91.         Draw bounding boxes around detected objects.

92.

93.         Args:

94.             frame (np.ndarray): Input frame.

95.             boxes (list): Detected bounding boxes.

96.             confidences (list): Confidence levels of detected objects.

97.             class\_ids (list): Class IDs of detected objects.

98.         """

99.         for i in range(len(boxes)):

100.             x, y, w, h = boxes[i]

101.             label = f"{self.classes[class\_ids[i]]}: {confidences[i]:.2f}"

102.             cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)

103.             cv2.putText(frame, label, (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 2)

104.

105.         cv2.imshow('Object Detection', frame)

106.         cv2.waitKey(0)

107.         cv2.destroyAllWindows()

108.

109. class TestObjectDetection(unittest.TestCase):

110.     def test\_detect\_objects(self):

111.         """

112.         Test object detection on a sample image.

113.         """

114.         # Load sample image

115.         frame = cv2.imread('samples/person.jpg')

116.

117.         # Initialize object detection model

118.         od = ObjectDetection(yolo\_weights='computer\_vision/yolov7.weights',

119.                               yolo\_cfg='computer\_vision/yolov7.cfg',

120.                               coco\_names='computer\_vision/coco.names')

121.

122.         # Detect objects

123.         boxes, confidences, class\_ids = od.detect\_objects(frame)

124.

125.         # Assert at least one object is detected

126.         self.assertGreaterEqual(len(boxes), 1)

127.

128.         # Assert the detected object is a person

129.         self.assertEqual(class\_ids[0], 0)

130.

131.         # Assert the confidence is high

132.         self.assertGreaterEqual(confidences[0], 0.8)

133.

134.         # Assert the bounding box is within the frame

135.         x, y, w, h = boxes[0]

136.         self.assertGreaterEqual(x, 0)

137.         self.assertGreaterEqual(y, 0)

138.         self.assertLessEqual(x + w, frame.shape[1])

139.         self.assertLessEqual(y + h, frame.shape[0])

140.

### computer\_vision\oneDrive.py

### computer\_vision\opencv\_utils.py

**Purpose**: A utility file that contains helper functions for working with OpenCV (a popular computer vision library).

#### Key Functions:

* + **Image and Video Processing**: Could include functions like capturing video, detecting objects, and applying image transformations.
* **Connection to main.py**:
  + This file would be called by any computer vision tasks in main.py, such as when capturing video for face recognition.

#### The Code

1. 1.  import cv2

2.

3. # Load the OpenCV library

4. cv2.namedWindow("Jarvis Vision", cv2.WINDOW\_NORMAL)

5.

6. # Set up the camera capture

7. cap = cv2.VideoCapture(0)  # Use default camera (index 0)

8.

9. while True:

10.     # Capture a frame from the camera

11.     ret, frame = cap.read()

12.

13.     # Convert the frame to grayscale

14.     gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

15.

16.     # Apply thresholding to segment out objects of interest

17.     \_, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH\_BINARY\_INV + cv2.THRESH\_OTSU)

18.

19.     # Find contours in the thresholded image

20.     contours, \_ = cv2.findContours(thresh, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

21.

22.     # Iterate through the contours and analyze them

23.     for contour in contours:

24.         # Calculate the area of the contour

25.         area = cv2.contourArea(contour)

26.

27.         # Filter out small contours (e.g., noise)

28.         if area > 1000:

29.             # Calculate the bounding rectangle of the contour

30.             x, y, w, h = cv2.boundingRect(contour)

31.

32.             # Draw a rectangle around the object of interest

33.             cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

34.

35.             # Extract the ROI (Region of Interest) from the original frame

36.             roi = frame[y:y + h, x:x + w]

37.

38.             #// Analyze the ROI using OpenCV functions (e.g., face detection, object recognition)

39.             #// For demonstration purposes, we'll just display the ROI

40.             cv2.imshow("ROI", roi)

41.

42.             #// Perform additional analysis on the ROI

43.             #// For example, you can use OpenCV functions to detect objects in the ROI

44.

45.             #// Perform additional analysis on the ROI

46.             #// For example, you can use OpenCV functions to detect objects in the ROI

47.             #...

48.

49.     # Display the output

50.     cv2.imshow("Jarvis Vision", frame)

51.

52.     # Exit on key press

53.     if cv2.waitKey(1) & 0xFF == ord('q'):

54.         break

55.

56. # Release resources

57. cap.release()

58. cv2.destroyAllWindows()

59.

60. import cv2

61. import pygame

62. import numpy as np

63. import pyautogui

64. import oneDrive

65.

66. def analyze\_camera\_feed(camera\_source):

67.     # computer vision logic

68.

69. def analyze\_image\_file(image\_path):

70.     # computer vision logic

71.

72. def analyze\_video\_file(video\_path):

73.     # computer vision logic

74.

75. def analyze\_one\_drive\_images():

76.     # logic to analyze images in OneDrive

77. Integrate project management and calendar APIs to manage tasks and events.

78. Integrate Google Calendar API and Outlook API to manage tasks and events.

79.  Google Calendar API and Outlook API to manage tasks and events.

1. import cv2

# Load the OpenCV library

2. cv2.namedWindow("Jarvis Vision", cv2.WINDOW\_NORMAL)

3.

4. # Set up the camera capture

5. cap = cv2.VideoCapture(0)  # Use default camera (index 0)

6.

7. while True:

8.     # Capture a frame from the camera

9.     ret, frame = cap.read()

10.

11.     # Convert the frame to grayscale

12.     gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

13.

14.     # Apply thresholding to segment out objects of interest

15.     \_, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH\_BINARY\_INV + cv2.THRESH\_OTSU)

16.

17.     # Find contours in the thresholded image

18.     contours, \_ = cv2.findContours(thresh, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

19.

20.     # Iterate through the contours and analyze them

21.     for contour in contours:

22.         # Calculate the area of the contour

23.         area = cv2.contourArea(contour)

24.

25.         # Filter out small contours (e.g., noise)

26.         if area > 1000:

27.             # Calculate the bounding rectangle of the contour

28.             x, y, w, h = cv2.boundingRect(contour)

29.

30.             # Draw a rectangle around the object of interest

31.             cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

32.

33.             # Extract the ROI (Region of Interest) from the original frame

34.             roi = frame[y:y + h, x:x + w]

35.

36.             #// Analyze the ROI using OpenCV functions (e.g., face detection, object recognition)

37.             #// For demonstration purposes, we'll just display the ROI

38.             cv2.imshow("ROI", roi)

39.

40.             #// Perform additional analysis on the ROI

41.             #// For example, you can use OpenCV functions to detect objects in the ROI

42.

43.             #// Perform additional analysis on the ROI

44.             #// For example, you can use OpenCV functions to detect objects in the ROI

45.             #...

46.

47.     # Display the output

48.     cv2.imshow("Jarvis Vision", frame)

49.

50.     # Exit on key press

51.     if cv2.waitKey(1) & 0xFF == ord('q'):

52.         break

53.

54. # Release resources

55. cap.release()

56. cv2.destroyAllWindows()

57.

58. import cv2

59. import pygame

60. import numpy as np

61. import pyautogui

62. import oneDrive

63.

64. def analyze\_camera\_feed(camera\_source):

65.     # computer vision logic

66.

67. def analyze\_image\_file(image\_path):

68.     # computer vision logic

69.

70. def analyze\_video\_file(video\_path):

71.     # computer vision logic

72.

73. def analyze\_one\_drive\_images():

74.     # logic to analyze images in OneDrive

75. Integrate project management and calendar APIs to manage tasks and events.

76. Integrate Google Calendar API and Outlook API to manage tasks and events.

77.

80.

### computer\_vision\people\_recognition.py

**Purpose**: Similar to contact\_recognition.py, this file could handle the identification of people based on facial or body recognition.

#### Key Functions:

* + **Person Identification**: Uses a dataset of known faces or body features to recognize or identify individuals.
* **Connection to main.py**:
  + This would likely be connected to Jaicat’s face recognition system for identifying or verifying users and other individuals captured on camera.

#### The Code

1. import face\_recognition

2. from PIL import Image

3. from googleapiclient.discovery import build

4. from onedrive import OneDrive

5.

6. # Set up Google Contacts API

7. contacts\_service = build('people', 'v1')

8.

9. # Set up OneDrive API

10. onedrive = OneDrive()

11.

12. # Load the user's contacts

13. contacts = contacts\_service.people().list().execute()

14.

15. # Create a dictionary to store the user's contacts and their corresponding images

16. contact\_images = {}

17.

18. # Loop through the user's contacts and download their profile pictures

19. for contact in contacts['connections']:

20.     contact\_id = contact['resourceName']

21.     contact\_name = contact['names'][0]['displayName']

22.     contact\_image\_url = contact['photos'][0]['url']

23.     contact\_image = onedrive.get\_file(contact\_image\_url)

24.     contact\_images[contact\_name] = contact\_image

25.

26. # Load the user's images from OneDrive

27. images = onedrive.get\_files()

28.

29. # Loop through the user's images and recognize the faces

30. for image in images:

31.     image\_path = image['path']

32.     image\_data = onedrive.get\_file(image\_path)

33.     image = face\_recognition.load\_image\_file(image\_data)

34.     faces = face\_recognition.face\_locations(image)

35.     for face in faces:

36.         face\_encoding = face\_recognition.face\_encodings(image, known\_face\_locations=[face])[0]

37.         # Compare the face encoding to the user's contacts

38.         for contact\_name, contact\_image in contact\_images.items():

39.             contact\_face\_encoding = face\_recognition.face\_encodings(contact\_image)[0]

40.             if face\_recognition.compare\_faces([contact\_face\_encoding], face\_encoding):

41.                 print(f"Found {contact\_name} in image {image\_path}")

42.

43. # Allow the user to hash tag images with contact names

44. def hash\_tag\_image(image\_path, contact\_name):

45.     # Get the contact's image

46.     contact\_image = contact\_images[contact\_name]

47.     # Get the image data

48.     image\_data = onedrive.get\_file(image\_path)

49.     # Load the image

50.     image = face\_recognition.load\_image\_file(image\_data)

51.     # Recognize the faces in the image

52.     faces = face\_recognition.face\_locations(image)

53.     for face in faces:

54.         face\_encoding = face\_recognition.face\_encodings(image, known\_face\_locations=[face])[0]

55.         # Compare the face encoding to the contact's face encoding

56.         if face\_recognition.compare\_faces([contact\_image], face\_encoding):

57.             print(f"Found {contact\_name} in image {image\_path}")

58.             # Link the image to the contact

59.             contact\_images[contact\_name].append(image\_path)

60.

61. # Test the function

62. hash\_tag\_image("image1.jpg", "John Doe")

63.

### computer\_vision\uk\_plate\_cascade.xml

### computer\_vision\vehicle\_detection.py

##### The Code

1. import cv2

2. import numpy as np

3.

4. class VehicleDetection:

5.     def \_\_init\_\_(self, yolo\_weights, yolo\_cfg, yolo\_names):

6.         # Load YOLO model

7.         self.net = cv2.dnn.readNet(yolo\_weights, yolo\_cfg)

8.         self.layer\_names = self.net.getLayerNames()

9.         self.output\_layers = [self.layer\_names[i[0] - 1] for i in self.net.getUnconnectedOutLayers()]

10.         self.classes = []

11.         with open(yolo\_names, "r") as f:

12.             self.classes = [line.strip() for line in f.readlines()]

13.

14.     def detect\_vehicles(self, frame):

15.         height, width, \_ = frame.shape

16.

17.         # Prepare the frame for detection

18.         blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0), True, crop=False)

19.         self.net.setInput(blob)

20.         outputs = self.net.forward(self.output\_layers)

21.

22.         class\_ids, confidences, boxes = [], [], []

23.

24.         # Loop through detections

25.         for output in outputs:

26.             for detection in output:

27.                 scores = detection[5:]

28.                 class\_id = np.argmax(scores)

29.                 confidence = scores[class\_id]

30.                 # Filter car (class\_id == 2) and motorbike (class\_id == 3)

31.                 if confidence > 0.5 and class\_id in [2, 3]:

32.                     center\_x = int(detection[0] \* width)

33.                     center\_y = int(detection[1] \* height)

34.                     w = int(detection[2] \* width)

35.                     h = int(detection[3] \* height)

36.

37.                     x = int(center\_x - w / 2)

38.                     y = int(center\_y - h / 2)

39.

40.                     boxes.append([x, y, w, h])

41.                     confidences.append(float(confidence))

42.                     class\_ids.append(class\_id)

43.

44.         # Non-Maximum Suppression to remove overlapping boxes

45.         indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

46.

47.         return [(boxes[i], confidences[i], self.classes[class\_ids[i]]) for i in indexes.flatten()]

48.

49.     def start\_detection(self):

50.         cap = cv2.VideoCapture(0)  # Open the webcam

51.         while True:

52.             ret, frame = cap.read()

53.             if not ret:

54.                 print("Error: Unable to capture video.")

55.                 break

56.

57.             detections = self.detect\_vehicles(frame)

58.             for (box, confidence, label) in detections:

59.                 x, y, w, h = box

60.                 label = f"{label} - {confidence:.2f}"

61.                 cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

62.                 cv2.putText(frame, label, (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)

63.

64.             cv2.imshow("Vehicle Detection", frame)

65.

66.             if cv2.waitKey(1) & 0xFF == ord('q'):

67.                 break

68.

69.         cap.release()

70.         cv2.destroyAllWindows()

71.

72.

### computer\_vision\yolov7.pt

### computer\_vision\yolov7.weights.pt

### computer\_vision\yolov7.cfg

### computer\_vision\visualization.py

1. import tkinter as tk

2. from tkinter import messagebox, ttk

3. import json

4. import os

5.

6. class DetectionViewer:

7.     def \_\_init\_\_(self, master):

8.         self.master = master

9.         self.master.title("Saved Detections Viewer")

10.         self.master.geometry("600x400")

11.

12.         self.detections\_file = "computer\_vision/detections.json"

13.

14.         # Create a table to display detections

15.         self.tree = ttk.Treeview(self.master, columns=("License Plate", "Vehicle Model", "Confidence"), show="headings")

16.         self.tree.heading("License Plate", text="License Plate")

17.         self.tree.heading("Vehicle Model", text="Vehicle Model")

18.         self.tree.heading("Confidence", text="Confidence")

19.         self.tree.pack(fill="both", expand=True)

20.

21.         # Button to load detections

22.         self.load\_button = tk.Button(self.master, text="Load Detections", command=self.load\_detections)

23.         self.load\_button.pack(pady=10)

24.

25.         # Export button

26.         self.export\_button = tk.Button(self.master, text="Export Detections", command=self.export\_detections)

27.         self.export\_button.pack(pady=10)

28.

29.     def load\_detections(self):

30.         """Load and display saved detections from JSON file."""

31.         if not os.path.exists(self.detections\_file):

32.             messagebox.showwarning("Warning", "No detections found.")

33.             return

34.

35.         with open(self.detections\_file, 'r') as file:

36.             detections = json.load(file)

37.

38.         # Clear previous data

39.         for item in self.tree.get\_children():

40.             self.tree.delete(item)

41.

42.         # Display detections in the table

43.         for detection in detections:

44.             license\_plate = detection.get("license\_plate", "Unknown")

45.             vehicle\_model = detection["vehicle\_data"].get("model", "Unknown")

46.             confidence = detection["detected\_cars"][0]["confidence"] if detection["detected\_cars"] else "N/A"

47.             self.tree.insert("", "end", values=(license\_plate, vehicle\_model, confidence))

48.

49.     def export\_detections(self):

50.         """Export the detections to an external file."""

51.         export\_path = "computer\_vision/exported\_detections.json"

52.         with open(self.detections\_file, 'r') as file:

53.             detections = json.load(file)

54.

55.         with open(export\_path, 'w') as export\_file:

56.             json.dump(detections, export\_file, indent=4)

57.

58.         messagebox.showinfo("Export", f"Detections exported to {export\_path}")

59.

60. if \_\_name\_\_ == "\_\_main\_\_":

61.     root = tk.Tk()

62.     app = DetectionViewer(root)

63.     root.mainloop()

64.

65.

## config

### config\api\_keys.py

**Purpose**: Contains the API keys for external services (e.g., Spotify, weather services, Google Calendar).

* **Key Functions**:
  + **API Key Storage**: Simply stores the keys and tokens necessary for accessing third-party services.
* **Connection to main.py**:
  + main.py would reference this file to load API keys when accessing external services like weather updates or music streaming.

Might Scrap this part as I want to Save Api Keys in the users information auomatically

### config\config.json

**Purpose**: Stores the configuration settings for Jaicat. This could include user preferences, system settings, and default behaviors.

* **Key Functions**:
  + **Configuration Management**: Stores user-specific settings and application-wide configurations (like language, time zone, etc.).
* **Connection to main.py**:
  + main.py would load this file to get the necessary configuration settings when initializing Jaicat.

## conversation

### conversation\\_\_pycache\_\_

### conversation\dialogue\_manager copy.py

**Purpose**: Likely a backup or older version of the main dialogue\_manager.py.

#### Key Functions:

* + **Manages Dialogue**: It might have similar functions to the main dialogue manager, managing user conversations and maintaining context.
* **Connection to main.py**:
  + Since this is a copy, it may not directly connect to main.py. It could be useful for historical reference or experimentation.

#### The code

1. # conversations/dialogue\_manager.py

2.

3. from features.nlu import NLU

4. from features.nlp import NLPSystem

5. from conversation.nlg import ContextualResponder

6. import random

7.

8. class DialogueManager:

9.     def \_\_init\_\_(self):

10.         self.nlu = NLU()

11.         self.nlp\_system = NLPSystem()

12.         self.contextual\_responder = ContextualResponder()

13.         self.context = {}  # To maintain context between interactions

14.

15.     def process\_user\_input(self, user\_input):

16.         # Step 1: Extract intent and entities

17.         extracted\_data = self.nlu.extract\_intent\_and\_entities(user\_input)

18.         intent = extracted\_data.get("intent")

19.         entities = extracted\_data.get("entities")

20.

21.         # Step 2: Handle the extracted intent

22.         if intent is None:

23.             return "I'm sorry, I didn't understand that. Could you please rephrase?"

24.

25.         # Greeting and Goodbye Intents

26.         if intent == "greet":

27.             return self.greet()

28.         elif intent == "goodbye":

29.             return self.goodbye()

30.

31.         # Custom Commands based on the user's input

32.         elif intent == "generate\_code":

33.             return self.generate\_code(entities)

34.         elif intent == "analyze\_text":

35.             return self.analyze\_text(entities)

36.         elif intent == "analyze\_sentiment":

37.             return self.analyze\_sentiment(entities)

38.

39.         # Example of adding more custom commands

40.         elif intent == "turn\_on\_lights":

41.             return self.turn\_on\_lights()

42.         elif intent == "play\_music":

43.             return self.play\_music()

44.

45.         # Handle unknown intents

46.         else:

47.             return self.handle\_unknown\_intent()

48.

49.     def greet(self):

50.         return random.choice(["Hello! How can I help you today?", "Hi there! What can I do for you?"])

51.

52.     def goodbye(self):

53.         return random.choice(["Goodbye! Have a great day!", "See you later!"])

54.

55.     def generate\_code(self, entities):

56.         if "code\_snippet" in entities:

57.             code\_snippet = entities["code\_snippet"]

58.             # Example: use your NLP system or code generation model

59.             generated\_code = self.nlp\_system.generate\_code(code\_snippet)

60.             return f"Here is the code I generated: {generated\_code}"

61.         else:

62.             return "I need more details to generate code. Could you provide a specific request?"

63.

64.     def analyze\_text(self, entities):

65.         if "text\_to\_analyze" in entities:

66.             text\_to\_analyze = entities["text\_to\_analyze"]

67.             analysis\_result = self.nlp\_system.analyze\_text(text\_to\_analyze)

68.             return f"The analysis result is: {analysis\_result}"

69.         else:

70.             return "Please provide the text you want me to analyze."

71.

72.     def analyze\_sentiment(self, entities):

73.         if "text\_to\_analyze" in entities:

74.             text\_to\_analyze = entities["text\_to\_analyze"]

75.             sentiment\_result = self.nlp\_system.analyze\_sentiment(text\_to\_analyze)

76.             return f"The sentiment analysis result is: {sentiment\_result}"

77.         else:

78.             return "Please provide the text for sentiment analysis."

79.

80.

81.

82.     def turn\_on\_lights(self):

83.         # Implement interaction with smart devices

84.         return random.choice(["Turning on the living room lights.", "Lights on!"])

85.

86.     def play\_music(self):

87.         # Implement interaction with a music service

88.         return random.choice(["Playing your favorite music.", "Starting your playlist now."])

89.

90.     def handle\_unknown\_intent(self):

91.         return "I'm not sure how to handle that. Can you try asking in a different way?"

92.

93. ######## \*\*General Commands\*\*

94.

95. #1. `hello` / `hi` / `hey` - Activate the AI assistant

96. #2. `goodbye` / `bye` - Deactivate the AI assistant

97. #3. `help` - Display available commands and features

98. #4. `what can you do` - Display a list of available features and modules

99.

100. #\*\*NLP and NLG\*\*

101.

102. #1. `tell me a story` - Generate a short story using NLG

103. #2. `write a poem` - Generate a poem using NLG

104. #3. `translate [text]` - Translate text using Language Translation module

105. #4. `summarize [text]` - Summarize text using NLP module

106.

107. #\*\*Text Classification and Code Generation\*\*

108.

109. #1. `classify [text]` - Classify text using Text Classification module

110. #2. `generate code [prompt]` - Generate code using Code Generation module

111. #3. `explain [code]` - Explain code using Code Generation module

112.

113. #\*\*Knowledge Base and Web Scraping\*\*

114.

115. #1. `learn from [pdf/epub]` - Learn from a PDF or EPUB file using Knowledge Base module

116. #2. `scrape [website]` - Scrape data from a website using Web Scraping module

117. #3. `find [information]` - Find information using Knowledge Base module

118.

119. #\*\*Image and Video Analysis\*\*

120.

121. #1. `analyze image [image]` - Analyze an image using Image Analysis module

122. #2. `analyze video [video]` - Analyze a video using Video Analysis module

123.

124. #\*\*YouTube and Social Media\*\*

125.

126. #1. `analyze youtube [video]` - Analyze a YouTube video using YouTube Analysis module

127. #2. `post on [social media]` - Post on a social media platform using Social Media Post Generation module

128. #3. `monitor [social media]` - Monitor a social media platform using Social Media Monitor module

129.

130. #\*\*People Recognition and Contact Management\*\*

131.

132. #1. `recognize [person]` - Recognize a person using People Recognition module

133. #2. `import contacts` - Import contacts using Contact Recognition module

134.

135. #\*\*Engineering and Motorbike Understanding\*\*

136.

137. #1. `explain [engineering concept]` - Explain an engineering concept using Engineering Understanding module

138. #2. `diagnose [motorbike issue]` - Diagnose a motorbike issue using Motorbike and Engine Understanding module

139.

140. #\*\*Home Surveillance and Automation\*\*

141.

142. #1. `display camera feeds` - Display camera feeds using Home Surveillance module

143. #2. `control [smart device]` - Control a smart device using Smart Home Automation module

144.

145. #\*\*Medical and Fitness Analysis\*\*

146.

147. #1. `analyze [medical data]` - Analyze medical data using Medical Analysis module

148. #2. `track [fitness data]` - Track fitness data using Fitness Analysis module

149.

150. #\*\*Travel Planning and Recommendations\*\*

151.

152. #1. `plan trip [destination]` - Plan a trip using Travel Planning module

153. #2. `recommend [travel destination]` - Recommend a travel destination using Travel Recommendations module

154.

155. #\*\*Personal Finance and Education\*\*

156.

157. #1. `track [personal finance data]` - Track personal finance data using Personal Finance Management module

158. #2. `learn [subject]` - Learn a subject using Education and Learning module

159.

160. #\*\*Health and Wellness\*\*

161.

162. #1. `track [health data]` - Track health data using Health and Wellness module

163. #2. `recommend [health advice]` - Recommend health advice using Health and Wellness module

164.

165. #\*\*Save Conversation Data\*\*

166.

167. #1. `save conversation` - Save conversation data using Save Conversation Data module

168.

169.

### conversation\dialogue\_manager.py

**Purpose**: Manages conversations by maintaining context and deciding how Jaicat should respond based on user inputs.

#### Key Functions:

* + **manage\_dialogue()**: Tracks conversation flow, maintaining context and adjusting responses based on prior interactions.
  + **generate\_response(dialogue\_act)**: Integrates with the NLG (Natural Language Generation) system to create appropriate responses for users.
* **Connection to main.py**:
  + main.py uses this file to handle dialogue responses and flow. For example, after recognizing the user's intent in features/nlp.py, it calls the dialogue\_manager.py to generate responses.

#### The Code

1. # conversations/dialogue\_manager.py

2.

3. from features.nlu import NLU

4. from features.nlp import NLPSystem

5. from conversation.nlg import ContextualResponder

6. import random

7. import wikipedia

8. import nltk

9. nltk.download('punkt')

10.

11. class DialogueManager:

12.     def \_\_init\_\_(self):

13.         self.nlu = NLU()

14.         self.nlp\_system = NLPSystem()

15.         self.contextual\_responder = ContextualResponder()

16.         self.context = {}  # To maintain context between interactions

17.

18.     def get\_answer(self, question):

19.         try:

20.             answer = wikipedia.summary(question, sentences=2)

21.             return answer

22.         except wikipedia.exceptions.DisambiguationError as e:

23.             return "Could you please be more specific? I found multiple results for that question."

24.         except wikipedia.exceptions.PageError:

25.             return "Sorry, I couldn't find any information on that topic."

26.

27.     def handle\_question(self, question):

28.         if question.startswith("how much does a "):

29.             entity = question.split(" ")[-1].lower()

30.             answer = self.get\_answer(entity)

31.             return answer

32.         else:

33.             return "Sorry, I didn't understand that question."

34.

35.     def process\_user\_input(self, user\_input):

36.         # Step 1: Extract intent and entities

37.         extracted\_data = self.nlu.extract\_intent\_and\_entities(user\_input)

38.         intent = extracted\_data.get("intent")

39.         entities = extracted\_data.get("entities")

40.

41.

42.         # Step 2: Handle the extracted intent

43.         if intent is None:

44.             return "I'm sorry, I didn't understand that. Could you please rephrase?"

45.

46.         # Greeting and Goodbye Intents

47.         if intent == "greet":

48.             return self.greet()

49.         elif intent == "goodbye":

50.             return self.goodbye()

51.

52.         # Custom Commands based on the user's input

53.         elif intent == "generate\_code":

54.             return self.generate\_code(entities)

55.         elif intent == "analyze\_text":

56.             return self.analyze\_text(entities)

57.         elif intent == "analyze\_sentiment":

58.             return self.analyze\_sentiment(entities)

59.

60.         # Example of adding more custom commands

61.         elif intent == "turn\_on\_lights":

62.             return self.turn\_on\_lights()

63.         elif intent == "play\_music":

64.             return self.play\_music()

65.         elif intent == "help":

66.             return self.help()

67.         elif intent == "what\_can\_you\_do":

68.             return self.what\_can\_you\_do()

69.         elif intent == "tell\_me\_a\_story":

70.             return self.tell\_me\_a\_story()

71.         elif intent == "question":

72.             return self.handle\_question(user\_input)

73.

74.         # Handle unknown intents

75.         else:

76.             return self.handle\_unknown\_intent()

77.

78.     def greet(self):

79.         return random.choice(["Hello! How can I help you today?", "Hi there! What can I do for you?"])

80.

81.     def goodbye(self):

82.         return random.choice(["Goodbye! Have a great day!", "See you later!"])

83.

84.     def generate\_code(self, entities):

85.         if "code\_snippet" in entities:

86.             code\_snippet = entities["code\_snippet"]

87.             # Example: use your NLP system or code generation model

88.             generated\_code = self.nlp\_system.generate\_code(code\_snippet)

89.             return f"Here is the code I generated: {generated\_code}"

90.         else:

91.             return "I need more details to generate code. Could you provide a specific request?"

92.

93.     def analyze\_text(self, entities):

94.         if "text\_to\_analyze" in entities:

95.             text\_to\_analyze = entities["text\_to\_analyze"]

96.             analysis\_result = self.nlp\_system.analyze\_text(text\_to\_analyze)

97.             return f"The analysis result is: {analysis\_result}"

98.         else:

99.             return "Please provide the text you want me to analyze."

100.

101.     def analyze\_sentiment(self, entities):

102.         if "text\_to\_analyze" in entities:

103.             text\_to\_analyze = entities["text\_to\_analyze"]

104.             sentiment\_result = self.nlp\_system.analyze\_sentiment(text\_to\_analyze)

105.             return f"The sentiment analysis result is: {sentiment\_result}"

106.         else:

107.             return "Please provide the text for sentiment analysis."

108.

109.     def turn\_on\_lights(self):

110.         # Implement interaction with smart devices

111.         return random.choice(["Turning on the living room lights.", "Lights on!"])

112.

113.     def play\_music(self):

114.         # Implement interaction with a music service

115.         return random.choice(["Playing your favorite music.", "Starting your playlist now."])

116.

117.     def help(self):

118.         return "I can help you with the following commands: greet, goodbye, generate\_code, analyze\_text, analyze\_sentiment, turn\_on\_lights, play\_music, help, what\_can\_you\_do, tell\_me\_a\_story."

119.

120.     def what\_can\_you\_do(self):

121.         return "I can perform the following tasks: generate code, analyze text, analyze sentiment, turn on lights, play music, and tell you a story."

122.

123.     def tell\_me\_a\_story(self):

124.         # Implement a story generation model or retrieve a story from a database

125.         story = "Once upon a time, there was a brave knight who slayed a dragon and saved the kingdom."

126.         return story

127.

128.     def handle\_unknown\_intent(self):

129.         return "I'm not sure how to handle that. Can you try asking in a different way?"

130.

131.     def detect\_vehicle(self):

132.         """Trigger car detection and retrieval of vehicle data."""

133.         return "Starting vehicle detection..."

134.

135.     # Add to intent recognition

136.     def process\_user\_input(self, user\_input):

137.         extracted\_data = self.nlu.extract\_intent\_and\_entities(user\_input)

138.         intent = extracted\_data.get("intent")

139.

140.         # Command mapping

141.         if intent == "detect\_vehicle":

142.             return self.detect\_vehicle()

143.         else:

144.             return self.handle\_unknown\_intent()

145.

146.     def process\_motorbike\_intents(intent: str, entities: list):

147.         if intent == "add\_motorbike":

148.             name = entities.get('motorbike\_name', 'Unnamed Motorbike')

149.             registration = entities.get('registration', 'Unknown')

150.             # Add motorbike logic here, save to JSON or respond via UI

151.             return f"Motorbike {name} with registration {registration} added."

152.

153.         if intent == "show\_motorbike\_info":

154.             # Logic to retrieve stored motorbike info

155.             return "Here are your motorbike details: [details]."

156.

157.         if intent == "motorbike\_surveillance":

158.             # Logic to start motorbike surveillance

159.             return "Starting motorbike surveillance."

160.

161.

162.

163.

164.

165.

166.

167.

168. ######## \*\*General Commands\*\*

169.

170. #1. `hello` / `hi` / `hey` - Activate the AI assistant

171. #2. `goodbye` / `bye` - Deactivate the AI assistant

172. #3. `help` - Display available commands and features

173. #4. `what can you do` - Display a list of available features and modules

174.

175. #\*\*NLP and NLG\*\*

176.

177. #1. `tell me a story` - Generate a short story using NLG

178. #2. `write a poem` - Generate a poem using NLG

179. #3. `translate [text]` - Translate text using Language Translation module

180. #4. `summarize [text]` - Summarize text using NLP module

181.

182. #\*\*Text Classification and Code Generation\*\*

183.

184. #1. `classify [text]` - Classify text using Text Classification module

185. #2. `generate code [prompt]` - Generate code using Code Generation module

186. #3. `explain [code]` - Explain code using Code Generation module

187.

188. #\*\*Knowledge Base and Web Scraping\*\*

189.

190. #1. `learn from [pdf/epub]` - Learn from a PDF or EPUB file using Knowledge Base module

191. #2. `scrape [website]` - Scrape data from a website using Web Scraping module

192. #3. `find [information]` - Find information using Knowledge Base module

193.

194. #\*\*Image and Video Analysis\*\*

195.

196. #1. `analyze image [image]` - Analyze an image using Image Analysis module

197. #2. `analyze video [video]` - Analyze a video using Video Analysis module

198.

199. #\*\*YouTube and Social Media\*\*

200.

201. #1. `analyze youtube [video]` - Analyze a YouTube video using YouTube Analysis module

202. #2. `post on [social media]` - Post on a social media platform using Social Media Post Generation module

203. #3. `monitor [social media]` - Monitor a social media platform using Social Media Monitor module

204.

205. #\*\*People Recognition and Contact Management\*\*

206.

207. #1. `recognize [person]` - Recognize a person using People Recognition module

208. #2. `import contacts` - Import contacts using Contact Recognition module

209.

210. #\*\*Engineering and Motorbike Understanding\*\*

211.

212. #1. `explain [engineering concept]` - Explain an engineering concept using Engineering Understanding module

213. #2. `diagnose [motorbike issue]` - Diagnose a motorbike issue using Motorbike and Engine Understanding module

214.

215. #\*\*Home Surveillance and Automation\*\*

216.

217. #1. `display camera feeds` - Display camera feeds using Home Surveillance module

218. #2. `control [smart device]` - Control a smart device using Smart Home Automation module

219.

220. #\*\*Medical and Fitness Analysis\*\*

221.

222. #1. `analyze [medical data]` - Analyze medical data using Medical Analysis module

223. #2. `track [fitness data]` - Track fitness data using Fitness Analysis module

224.

225. #\*\*Travel Planning and Recommendations\*\*

226.

227. #1. `plan trip [destination]` - Plan a trip using Travel Planning module

228. #2. `recommend [travel destination]` - Recommend a travel destination using Travel Recommendations module

229.

230. #\*\*Personal Finance and Education\*\*

231.

232. #1. `track [personal finance data]` - Track personal finance data using Personal Finance Management module

233. #2. `learn [subject]` - Learn a subject using Education and Learning module

234.

235. #\*\*Health and Wellness\*\*

236.

237. #1. `track [health data]` - Track health data using Health and Wellness module

238. #2. `recommend [health advice]` - Recommend health advice using Health and Wellness module

239.

240. #\*\*Save Conversation Data\*\*

241.

242. #1. `save conversation` - Save conversation data using Save Conversation Data module

243.

244.

### conversation\nlg.py

**Purpose**: This file generates human-like responses to the user’s input using NLG (Natural Language Generation) techniques.

#### Key Functions:

* + **generate\_response(dialogue\_act, entities)**: Creates a response based on the user’s intent (dialogue act) and the extracted entities.
* **Connection to main.py**:
  + Once Jaicat has understood the user’s input (via NLU), it calls this file to generate a response that will be spoken aloud or displayed in the GUI.

#### The Code

1. from transformers import BartForConditionalGeneration, BartTokenizer

2.

3. class NLG:

4.     def \_\_init\_\_(self):

5.         self.bart\_model = BartForConditionalGeneration.from\_pretrained("facebook/bart-base")

6.         self.bart\_tokenizer = BartTokenizer.from\_pretrained("facebook/bart-base")

7.

8.     def generate\_response(self, context, input\_text):

9.         inputs = self.bart\_tokenizer.encode\_plus(

10.             f"{context} {input\_text}",

11.             return\_tensors="pt",

12.             max\_length=512,

13.             truncation=True,

14.             padding="max\_length"

15.         )

16.         outputs = self.bart\_model.generate(

17.             inputs["input\_ids"],

18.             attention\_mask=inputs["attention\_mask"],

19.             max\_length=512

20.         )

21.         response = self.bart\_tokenizer.decode(outputs[0], skip\_special\_tokens=True)

22.         return response

23.

24. def classify\_intent(self, text):

25.     if "play song" in text:

26.         return "play\_song"

27.

28. class ContextualResponder:

29.     def respond(self, context, input\_text):

30.         return f"Here's a response based on the context: {context} and input text: {input\_text}"

31.

32.

33. def generate\_response(dialogue\_act, entities=None):

34.     if dialogue\_act['intent'] == 'greet':

35.         response = 'Hello! How can I help you?'

36.     elif dialogue\_act['intent'] == 'goodbye':

37.         response = 'Goodbye! Have a great day!'

38.     elif dialogue\_act['intent'] == 'request\_weather':

39.         city = next((ent for ent in entities if ent[1] == 'CITY'), None)

40.         if city:

41.             response = f'The current weather in {city[0]} is sunny with a temperature of 72 degrees.'

42.         else:

43.             response = 'What city are you interested in checking the weather for?'

44.     elif dialogue\_act['intent'] == 'set\_reminder':

45.         reminder = dialogue\_act['slots']['reminder']

46.         response = f'Got it! I will remind you to {reminder}.'

47.     else:

48.         response = 'I\'m sorry, I didn\'t understand that. Could you please rephrase?'

49.

50.

51.

52.     return response

53.

54.

### conversation\nlu.py

**Purpose**: Handles natural language understanding (NLU), including intent detection and entity extraction using SpaCy, and possibly machine learning models.

#### Key Functions:

* + **extract\_intent\_and\_entities(user\_input)**: Extracts both the user’s intent and entities from their input using SpaCy or custom models (e.g., a RandomForestClassifier for intent).
  + **predict\_intent(input\_text)**: Uses a machine learning model to classify the user’s intent.
  + **extract\_entities(input\_text)**: Extracts entities from text, like names, dates, and locations, using SpaCy’s entity recognition.
* **Connection to main.py**:
  + This file is heavily used by main.py to interpret user input. When Jaicat receives a command, it calls nlu.py to extract the user's intent and identify any entities (e.g., locations for weather queries).

#### The Code

1. import spacy

2. import pandas as pd

3. from sklearn.ensemble import RandomForestClassifier

4. from sklearn.feature\_extraction.text import TfidfVectorizer

5.

6. class NLU:

7.     def \_\_init\_\_(self):

8.         self.nlp = spacy.load('en\_core\_web\_sm')

9.         self.vectorizer = TfidfVectorizer()  # Define vectorizer here

10.         self.intent\_model = self.train\_intent\_model()

11.

12.     def train\_intent\_model(self):

13.         """Train the intent classification model."""

14.         # Load the dataset

15.         intent\_data = pd.read\_csv("intent\_data.csv")  # Load the intent data

16.

17.         # Preprocess the data

18.         X = self.vectorizer.fit\_transform(intent\_data["input\_text"])  # Use self.vectorizer here

19.         y = intent\_data["intent"]

20.

21.         # Train the model

22.         clf = RandomForestClassifier(n\_estimators=100)

23.         clf.fit(X, y)

24.         return clf

25.

26.     def extract\_intent\_and\_entities(self, user\_input):

27.         """Extract intent and entities from user input."""

28.         intent = self.predict\_intent(user\_input)

29.         entities = self.extract\_entities(user\_input)

30.

31.         return {

32.             "intent": intent,

33.             "entities": entities

34.         }

35.

36.     def predict\_intent(self, input\_text):

37.         """Predict intent based on the input text."""

38.         input\_vector = self.vectorizer.transform([input\_text])  # Use self.vectorizer here

39.         return self.intent\_model.predict(input\_vector)[0]

40.

41.     def extract\_entities(self, input\_text):

42.         """Extract entities from the input text using SpaCy."""

43.         doc = self.nlp(input\_text)

44.         return [(ent.text, ent.label\_) for ent in doc.ents]

45.

46. # If you want to keep standalone functions for intent and entity extraction, here they are:

47.

48. def extract\_intent(text):

49.     """Extract the intent from the input text."""

50.     nlp = spacy.load("en\_core\_web\_sm")

51.     doc = nlp(text)

52.     intent = ""

53.     for token in doc:

54.         if token.dep\_ == "ROOT":

55.             intent = token.text

56.             break

57.     return intent

58.

59. def extract\_entities(text):

60.     """Extract entities from the input text."""

61.     nlp = spacy.load("en\_core\_web\_sm")

62.     doc = nlp(text)

63.     entities = []

64.     for ent in doc.ents:

65.         entities.append((ent.text, ent.label\_))

66.     return entities

67.

68. def extract\_intent\_and\_entities(self, user\_input):

69.     doc = self.nlp(user\_input)

70.

71.     # Extract intent (e.g., "get\_travel\_recommendations")

72.     intent = "get\_travel\_recommendations" if "travel" in user\_input and "recommendations" in user\_input else None

73.     intent = "capture\_image" if "capture" in user\_input and "image" in user\_input else None

74.     # Extract entities (e.g., category)

75.     entities = {}

76.     if intent:

77.         for token in doc:

78.             if token.text in self.destinations:  # Check if the token matches any category

79.                 entities['category'] = token.text

80.     if "create project" in user\_input:

81.             return {

82.                 "intent": "create\_project",

83.                 "entities": {"project\_name": user\_input.replace("create project", "").strip()}

84.             }

85.     elif "add task" in user\_input:

86.             return {

87.                 "intent": "add\_task",

88.                 "entities": {

89.                     "project\_name": user\_input.split("in")[-1].strip(),

90.                     "task": user\_input.replace("add task", "").split("in")[0].strip()

91.                 }

92.             }

93.     elif "list projects" in user\_input:

94.             return {"intent": "list\_projects", "entities": {}}

95.     elif "complete project" in user\_input:

96.             return {

97.                 "intent": "complete\_project",

98.                 "entities": {"project\_name": user\_input.replace("complete project", "").strip()}

99.             }

100.

101.     if "capture image" in user\_input:

102.         return {"intent": "capture\_image", "entities": {}}

103.

104.     if "get camera status" in user\_input:

105.         return {"intent": "get\_camera\_status", "entities": {}}

106.     elif "capture snapshot" in user\_input:

107.         camera\_id = extract\_camera\_id(user\_input)  # Implement a method to extract camera\_id

108.         return {"intent": "capture\_snapshot", "entities": {"camera\_id": camera\_id}}

109.     elif "start recording" in user\_input:

110.         camera\_id = extract\_camera\_id(user\_input)

111.         return {"intent": "start\_recording", "entities": {"camera\_id": camera\_id}}

112.     elif "stop recording" in user\_input:

113.         camera\_id = extract\_camera\_id(user\_input)

114.         return {"intent": "stop\_recording", "entities": {"camera\_id": camera\_id}}

115.     if "capture snapshot" in user\_input:

116.         return {"intent": "capture\_snapshot", "entities": {}}

117.     elif "start recording" in user\_input:

118.         return {"intent": "start\_recording", "entities": {}}

119.     elif "stop recording" in user\_input:

120.         return {"intent": "stop\_recording", "entities": {}}

121.     if "nutrition facts" in user\_input:

122.         return {"intent": "get\_nutrition", "entities": {"food\_item": user\_input.split("nutrition facts")[-1].strip()}}

123.     elif "find recipes" in user\_input:

124.         return {"intent": "find\_recipes", "entities": {"ingredient": user\_input.split("find recipes")[-1].strip()}}

125.     elif "recipe instructions" in user\_input:

126.         return {"intent": "get\_recipe\_instructions", "entities": {"recipe\_id": user\_input.split("recipe instructions")[-1].strip()}}

127.     if "analyze file" in user\_input:

128.         return {"intent": "analyze\_file", "entities": {"file\_path": user\_input.split("analyze file")[-1].strip()}}

129.

130.

131.     return {"intent": intent, "entities": entities}

132.

## data

### data\calendar\_data.csv

**Purpose**: Likely stores calendar-related data, such as dates and events for personal scheduling.

* **Connection to main.py**:
  + This could be used by the calendar service in main.py to retrieve or display the user’s schedule.

### data\weather\_data.csv

**Purpose**: Stores weather-related data, either as sample data or a backup source in case the weather API fails.

* **Connection to main.py**:
  + Could be used to display historical weather data or as a backup for fetching weather information when the API is unavailable.

## database

### database\conversation.py

**Purpose**: Stores conversation history or manages data related to conversations between the user and Jaicat.

* **Key Functions**:
  + **Conversation Data Management**: Could store conversation logs for analysis or future reference.
* **Connection to main.py**:
  + Might be used to maintain continuity in conversations, especially when Jaicat needs to reference previous interactions.
* The Code

1. # database/conversation.py

2.

3. import sqlite3

4. from datetime import datetime

5.

6. class ConversationDatabase:

7.     def \_\_init\_\_(self, db\_name='conversation\_history.db'):

8.         """Initialize the database connection and create the table if it doesn't exist."""

9.         self.conn = sqlite3.connect(db\_name)

10.         self.create\_table()

11.

12.     def create\_table(self):

13.         """Create the conversation history table."""

14.         with self.conn:

15.             self.conn.execute('''

16.                 CREATE TABLE IF NOT EXISTS conversations (

17.                     id INTEGER PRIMARY KEY,

18.                     user\_input TEXT NOT NULL,

19.                     assistant\_response TEXT NOT NULL,

20.                     timestamp TEXT NOT NULL

21.                 )

22.             ''')

23.

24.     def add\_conversation(self, user\_input, assistant\_response):

25.         """Add a conversation entry to the database."""

26.         timestamp = datetime.now().isoformat()

27.         with self.conn:

28.             self.conn.execute('''

29.                 INSERT INTO conversations (user\_input, assistant\_response, timestamp)

30.                 VALUES (?, ?, ?)

31.             ''', (user\_input, assistant\_response, timestamp))

32.

33.     def get\_conversation\_history(self):

34.         """Retrieve the conversation history from the database."""

35.         with self.conn:

36.             cursor = self.conn.execute('SELECT \* FROM conversations ORDER BY timestamp')

37.             return cursor.fetchall()

38.

39.     def close(self):

40.         """Close the database connection."""

41.         self.conn.close()

42.

43. if \_\_name\_\_ == "\_\_main\_\_":

44.     # Example usage

45.     db = ConversationDatabase()

46.     db.add\_conversation("What's the weather today?", "It's sunny with a high of 25°C.")

47.

48.     # Fetch all conversations

49.     history = db.get\_conversation\_history()

50.     for entry in history:

51.         print(f"User: {entry[1]} | Assistant: {entry[2]} | Time: {entry[3]}")

52.

53.     db.close()

54.

### database\enginnering\_mod.py

**Purpose**: This file might handle engineering-related modules or data, possibly for specific technical tasks or utilities.

* **Connection to main.py**:
  + No direct connection, unless Jaicat is performing specialized engineering tasks that require this module.

The Code

1. import requests

2. from bs4 import BeautifulSoup

3.

4. # Define a function to search for engineering information on Wikipedia

5. def search\_engineering\_info(topic):

6.     url = "https://en.wikipedia.org/wiki/" + topic.replace(" ", "\_")

7.     response = requests.get(url)

8.     soup = BeautifulSoup(response.content, "html.parser")

9.     info\_box = soup.find("table", class\_="infobox vcard")

10.     if info\_box:

11.         specs = {}

12.         for row in info\_box.find\_all("tr"):

13.             key = row.find("th")

14.             value = row.find("td")

15.             if key and value:

16.                 specs[key.text.strip()] = value.text.strip()

17.         return specs

18.     else:

19.         return "Engineering topic not found"

20.

21. # Example usage:

22. topic = "Mechanical Engineering"

23. result = search\_engineering\_info(topic)

24. print(result)

25.

### database\knowledge\_base.py

**Purpose**: Manages a knowledge base that Jaicat can use to answer factual questions or retrieve information.

* **Key Functions**:
  + **Query Knowledge**: Allows Jaicat to access structured information for providing factual responses.
* **Connection to main.py**:
  + When the user asks a factual question, main.py would call this file to retrieve the necessary information.
* The code

1. import sqlite3

2.

3. def integrate\_knowledge\_base():

4.     # Create a database connection

5.     conn = sqlite3.connect("jaicat\_db.db")

6.     # Create tables for storing knowledge base data

7.     conn.execute("CREATE TABLE IF NOT EXISTS knowledge\_base (id INTEGER PRIMARY KEY, topic TEXT, information TEXT)")

8.     # Insert data into the knowledge base

9.     conn.execute("INSERT INTO knowledge\_base (topic, information) VALUES ('Python', 'Python is a programming language')")

10.     # Retrieve data from the knowledge base

11.     cursor = conn.execute("SELECT \* FROM knowledge\_base WHERE topic='Python'")

12.     result = cursor.fetchone()

13.     return result

14.

15. import sqlite3

16.

17. def integrate\_knowledge\_base():

18.     # Connect to the knowledge base database

19.     conn = sqlite3.connect('knowledge\_base.db')

20.     cursor = conn.cursor()

21.

22.     # Define a function to retrieve information from the knowledge base

23.     def retrieve\_information(query):

24.         # Execute the query

25.         cursor.execute(query)

26.

27.         # Fetch the results

28.         results = cursor.fetchall()

29.

30.         # Return the results

31.         return results

32.

33.     # Return the knowledge base interface

34.     return retrieve\_information

35.

### database\knowledge\_graph.py

**Purpose**: Manages a knowledge graph that could represent relationships between different entities, improving Jaicat’s ability to understand complex queries.

* **Connection to main.py**:
  + When Jaicat is asked to reason through relationships or complex queries, main.py might use this file to access the knowledge graph.

The Code

1. import networkx as nx

2.

3. # Create a knowledge graph

4. G = nx.Graph()

5.

6. # Add nodes and edges to the graph based on the analyzed data

7. #...

8.

9. # Define a function to retrieve information from the knowledge graph

10. def retrieve\_information(query):

11.     # Perform graph traversal and retrieval

12.     #...

13.     return results

14.

15. # Example usage:

16. query = "What is the capital of France?"

17. results = retrieve\_information(query)

18. print(results)

19.

20. import requests

21. import json

22.

23. # Example: Linking Wikidata and DBpedia knowledge graphs

24.

25. # Extract data from Wikidata

26. wikidata\_url = "https://www.wikidata.org/w/api.php?action=wbsearchentities&search=cat&format=json"

27. wikidata\_response = requests.get(wikidata\_url)

28. wikidata\_data = json.loads(wikidata\_response.text)

29.

30. # Extract data from DBpedia

31. dbpedia\_url = "https://dbpedia.org/sparql"

32. dbpedia\_query = """

33.     PREFIX dbo: <http://dbpedia.org/ontology/>

34.     SELECT ?label ?abstract

35.     WHERE {

36.         ?resource dbo:species ?species .

37.         ?resource rdfs:label ?label .

38.         ?resource dbo:abstract ?abstract .

39.         FILTER (lang(?label) = 'en' && lang(?abstract) = 'en' && ?species = <http://dbpedia.org/resource/Cat>)

40.     }

41. """

42. dbpedia\_response = requests.post(dbpedia\_url, data={"query": dbpedia\_query})

43. dbpedia\_data = json.loads(dbpedia\_response.text)

44.

45. # Integrate the data

46. integrated\_data = {

47.     "wikidata": wikidata\_data,

48.     "dbpedia": dbpedia\_data

49. }

50.

51. # Store the integrated data

52. with open("integrated\_knowledge\_graph.json", "w") as file:

53.     json.dump(integrated\_data, file)

54.

55. import networkx as nx

56. import requests

57. import json

58.

59. class KnowledgeGraph:

60.     def \_\_init\_\_(self):

61.         self.G = nx.Graph()

62.

63.     def create\_knowledge\_graph(self):

64.         # Create a knowledge graph

65.         #G = nx.Graph()

66.

67.         # Add nodes and edges to the graph based on the analyzed data

68.         #...

69.

70.         # Define a function to retrieve information from the knowledge graph

71.         def retrieve\_information(query):

72.             # Perform graph traversal and retrieval

73.             #...

74.             return results

75.

76.         # Example usage:

77.         query = "What is the capital of France?"

78.         results = retrieve\_information(query)

79.         print(results)

80.

81.         # Integrate data from Wikidata and DBpedia

82.         wikidata\_url = "https://www.wikidata.org/w/api.php?action=wbsearchentities&search=cat&format=json"

83.         wikidata\_response = requests.get(wikidata\_url)

84.         wikidata\_data = json.loads(wikidata\_response.text)

85.

86.         dbpedia\_url = "https://dbpedia.org/sparql"

87.         dbpedia\_query = """

88.             PREFIX dbo: <http://dbpedia.org/ontology/>

89.             SELECT ?label ?abstract

90.             WHERE {

91.                 ?resource dbo:species ?species .

92.                 ?resource rdfs:label ?label .

93.                 ?resource dbo:abstract ?abstract .

94.                 FILTER (lang(?label) = 'en' && lang(?abstract) = 'en' && ?species = <http://dbpedia.org/resource/Cat>)

95.             }

96.         """

97.         dbpedia\_response = requests.post(dbpedia\_url, data={"query": dbpedia\_query})

98.         dbpedia\_data = json.loads(dbpedia\_response.text)

99.

100.         # Integrate the data

101.         integrated\_data = {

102.             "wikidata": wikidata\_data,

103.             "dbpedia": dbpedia\_data

104.         }

105.

106.         # Store the integrated data

107.         with open("integrated\_knowledge\_graph.json", "w") as file:

108.             json.dump(integrated\_data, file)

109.

110.         return self.G

111.

### database\motorbikes.py

**Purpose**: Likely manages data or a model related to motorbike identification or related tasks.

* **Connection to main.py**:
  + Could be called when Jaicat is asked to recognize or work with motorbike-related data.

The Code

1. import requests

2. from bs4 import BeautifulSoup

3.

4. # Define a function to search for motorbike information on Wikipedia

5. def search\_motorbike\_info(motorbike):

6.     url = "https://en.wikipedia.org/wiki/" + motorbike.replace(" ", "\_")

7.     response = requests.get(url)

8.     soup = BeautifulSoup(response.content, "html.parser")

9.     info\_box = soup.find("table", class\_="infobox vcard")

10.     if info\_box:

11.         specs = {}

12.         for row in info\_box.find\_all("tr"):

13.             key = row.find("th")

14.             value = row.find("td")

15.             if key and value:

16.                 specs[key.text.strip()] = value.text.strip()

17.         return specs

18.     else:

19.         return "Motorbike not found"

20.

21. # Example usage:

22. motorbike = "Honda CB500F"

23. result = search\_motorbike\_info(motorbike)

24. print(result)

25.

### database\mysql\_utils.py

**Purpose**: Contains utility functions for interacting with MySQL databases.

* **Key Functions**:
  + **Database Operations**: Handles CRUD (Create, Read, Update, Delete) operations in a MySQL database.
* **Connection to main.py**:
  + If Jaicat uses MySQL for data storage, this file will be referenced when performing any database operations.

The Code

1. # database/mysql\_utils.py

2.

3. import mysql.connector

4. from mysql.connector import Error

5.

6. class MySQLUtils:

7.     def \_\_init\_\_(self, host='localhost', database='jaicat\_db', user='root', password=''):

8.         """Initialize MySQL connection."""

9.         self.host = host

10.         self.database = database

11.         self.user = user

12.         self.password = password

13.         self.connection = None

14.

15.     def connect(self):

16.         """Establish connection to the MySQL database."""

17.         try:

18.             self.connection = mysql.connector.connect(

19.                 host=self.host,

20.                 database=self.database,

21.                 user=self.user,

22.                 password=self.password

23.             )

24.             if self.connection.is\_connected():

25.                 print(f"Connected to MySQL database: {self.database}")

26.         except Error as e:

27.             print(f"Error while connecting to MySQL: {e}")

28.

29.     def execute\_query(self, query, params=None):

30.         """Execute a query (INSERT, UPDATE, DELETE)."""

31.         try:

32.             if self.connection.is\_connected():

33.                 cursor = self.connection.cursor()

34.                 cursor.execute(query, params)

35.                 self.connection.commit()

36.                 print("Query executed successfully.")

37.         except Error as e:

38.             print(f"Error executing query: {e}")

39.

40.     def fetch\_data(self, query, params=None):

41.         """Fetch data (SELECT queries)."""

42.         try:

43.             if self.connection.is\_connected():

44.                 cursor = self.connection.cursor()

45.                 cursor.execute(query, params)

46.                 result = cursor.fetchall()

47.                 return result

48.         except Error as e:

49.             print(f"Error fetching data: {e}")

50.             return None

51.

52.     def close(self):

53.         """Close the MySQL database connection."""

54.         if self.connection.is\_connected():

55.             self.connection.close()

56.             print("MySQL connection closed.")

57.

58.

### database\Nas.py

**Purpose**: This file might handle network-attached storage (NAS) systems, possibly allowing Jaicat to interact with external storage systems.

* **Connection to main.py**:
  + Jaicat could use this file when retrieving or storing data on a NAS.

The Code

1. import os

2. import smbclient

3.

4. # Set up your NAS drive credentials

5. nas\_username = 'your\_username'

6. nas\_password = 'your\_password'

7. nas\_ip\_address = '192.168.1.8'  # Replace with your NAS drive's IP address

8.

9. # Connect to your NAS drive using SMB

10. smb\_client = smbclient.open\_file('\\' + nas\_ip\_address + '\\shared\_folder', mode='r', username=nas\_username, password=nas\_password)

11.

12. # List the files and folders on your NAS drive

13. files\_and\_folders = smb\_client.listdir()

14. print(files\_and\_folders)

15.

16. # Close the SMB connection

17. smb\_client.close()

18.

19. import pyudev

20.

21. # Set up the NAS drive device path

22. nas\_device\_path = '/dev/sdb1'  # Replace with your NAS drive's device path

23.

24. # Create a pyudev monitor

25. monitor = pyudev.Monitor.from\_netlink(pyudev.Context())

26.

27. # Filter for changes on the NAS drive

28. monitor.filter\_by('block')

29.

30. # Monitor for changes

31. for device in iter(monitor.receive\_device):

32.     if device.device\_path == nas\_device\_path:

33.         # Update your AI assistant with the changes

34.         print('NAS drive changed!')

35.         # Call the get\_files\_from\_nas function to update the file list

36.         files\_and\_folders = get\_files\_from\_nas()

37.         # Update your AI assistant's knowledge base with the new file list

38.         jaicat.update\_knowledge\_base(files\_and\_folders)

39.

40.

### database\web\_scraping.py

**Purpose**: Likely used for web scraping tasks, allowing Jaicat to extract information from web pages.

* **Connection to main.py**:
  + If Jaicat is asked to fetch live information from the web (e.g., news articles, prices), main.py would call this file for web scraping purposes.

The Code

1. import requests

2. from bs4 import BeautifulSoup

3.

4. # Define a function to scrape a website

5. def scrape\_website(url):

6.     response = requests.get(url)

7.     soup = BeautifulSoup(response.text, 'html.parser')

8.     # Extract relevant information from the website

9.     title = soup.find('title').text

10.     paragraphs = [p.text for p in soup.find\_all('p')]

11.     return title, paragraphs

12.

13. class Politics:

14.     def collect\_data(self):

15.         url = "https://www.example.com"

16.         try:

17.             title, paragraphs = scrape\_website(url)

18.             return title, paragraphs

19.         except requests.exceptions.RequestException as e:

20.             print(f" Error: {e}")

21.             return None, None

22.

23. # Create an instance of the Politics class

24. politics = Politics()

25.

26. # Test the collect\_data method

27. title, paragraphs = politics.collect\_data()

28. print(title)

29. print(paragraphs)

30.

## Documentation

### Documentation\documentation.py

**Purpose**: Likely a script that automatically generates or compiles documentation for Jaicat, or it could be used to handle programmatic documentation features.

* **Key Functions**:
  + **Auto Documentation**: Could generate or manage project documentation from code comments or structure.
* **Connection to main.py**:
  + No direct connection to main.py, but useful for project maintenance and clarity. It helps ensure that the code is well-documented.

The Code

1. # Documentation/documentation.py

2.

3. import os

4. import re

5.

6. class DocumentationGenerator:

7.     def \_\_init\_\_(self, root\_dir):

8.         """Initialize with the root directory of the project."""

9.         self.root\_dir = root\_dir

10.         self.docs = []

11.

12.     def extract\_docstrings(self, file\_path):

13.         """Extracts docstrings from a Python file."""

14.         with open(file\_path, 'r', encoding='utf-8') as file:

15.             content = file.read()

16.

17.         # Regular expression to find docstrings

18.         docstring\_pattern = re.compile(r'(\"\"\"(.\*?)\"\"\"|\'\'\'(.\*?)\'\'\')', re.DOTALL)

19.         return docstring\_pattern.findall(content)

20.

21.     def generate\_docs(self):

22.         """Generate documentation for all Python files in the specified directory."""

23.         for dirpath, \_, filenames in os.walk(self.root\_dir):

24.             for filename in filenames:

25.                 if filename.endswith('.py'):

26.                     file\_path = os.path.join(dirpath, filename)

27.                     docstrings = self.extract\_docstrings(file\_path)

28.

29.                     if docstrings:

30.                         self.docs.append({

31.                             'file': file\_path,

32.                             'docstrings': [d[0] for d in docstrings]  # Extracted docstrings

33.                         })

34.

35.     def save\_docs(self, output\_file='documentation.md'):

36.         """Save the extracted documentation to a Markdown file."""

37.         with open(output\_file, 'w', encoding='utf-8') as file:

38.             file.write('# Project Documentation\n\n')

39.             for entry in self.docs:

40.                 file.write(f"## {entry['file']}\n")

41.                 for docstring in entry['docstrings']:

42.                     file.write(f"\n```\n{docstring}\n```\n")

43.                 file.write("\n---\n")

44.

45.     def run(self):

46.         """Run the documentation generator."""

47.         self.generate\_docs()

48.         self.save\_docs()

49.         print("Documentation generated successfully!")

50.

51. if \_\_name\_\_ == "\_\_main\_\_":

52.     # Change 'C:/Users/josh\_/Desktop/jaicat\_project' to your project's root directory

53.     doc\_generator = DocumentationGenerator(root\_dir='C:/Users/josh\_/Desktop/jaicat\_project')

54.     doc\_generator.run()

55.

### Documentation\README.md

**Purpose**: A Markdown file that likely serves as a high-level overview of the Jaicat project. It typically includes installation instructions, usage examples, and other project details.

* **Connection to main.py**:
  + Not connected to the code directly, but important for developers and users to understand the setup and usage of Jaicat.

## enrollment\_json

### enrollment\_json\Jay.json

**Purpose**: This JSON file likely stores user-specific settings, preferences, or information for the user "Jay". This includes the user's access permissions, mood preferences, or any custom Jaicat settings.

* **Connection to main.py**:
  + main.py reads from this file to retrieve and apply Jay’s specific settings when interacting with Jaicat (e.g., mood, voice preferences).

### enrollment\_json\Samantha.json

**Purpose**: Similar to Jay.json, this file stores the settings and information for another user, "Samantha."

## enrollment\_pictures

### enrollment\_pictures\Jay.png

**Purpose**: Stores the enrolled face image of the user "Jay" for facial recognition purposes.

* **Connection to main.py**:
  + During face recognition in main.py, this file is compared with the current camera feed to identify the user.

### enrollment\_pictures\Samantha.png

**Purpose**: Similar to Jay.png, this file stores Samantha's facial data for recognition.

* **Connection to main.py**:
  + Used during the face recognition process in main.py to identify Samantha.

## features

### features\\_\_pycache\_\_

### features\code\_generation.py

**Purpose**: Provides the capability for Jaicat to generate Python code or other script-based solutions based on the user’s input.

#### Key Functions:

* + **generate\_code(prompt)**: Takes a user-defined prompt and returns auto-generated code, likely using a GPT-based model like CodeBERT.
* **Connection to main.py**:
  + If the user asks Jaicat to generate code, main.py calls code\_generation.py to fulfill that request, allowing Jaicat to function as a coding assistant.

#### The Code

1. import torch

2. from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

3.

4. class CodeGenerationModel:

5.     def \_\_init\_\_(self):

6.         # Load pre-trained models

7.         self.codebert\_tokenizer = AutoTokenizer.from\_pretrained('microsoft/codebert-base')

8.         self.codebert\_model = AutoModelForSeq2SeqLM.from\_pretrained('microsoft/codebert-base')

9.

10.         self.bert\_tokenizer = AutoTokenizer.from\_pretrained('bert-base-uncased')

11.         self.bert\_model = AutoModelForSeq2SeqLM.from\_pretrained('bert-base-uncased')

12.

13.     def generate\_code\_with\_codebert(self, input\_text):

14.         inputs = self.codebert\_tokenizer.encode\_plus(

15.             input\_text,

16.             add\_special\_tokens=True,

17.             max\_length=512,

18.             return\_tensors='pt'

19.         )

20.         outputs = self.codebert\_model.generate(inputs['input\_ids'], attention\_mask=inputs['attention\_mask'])

21.         codebert\_code = self.codebert\_tokenizer.decode(outputs[0], skip\_special\_tokens=True)

22.         return codebert\_code

23.

24.     def generate\_code\_with\_blackbox(self, input\_text):

25.         bert\_inputs = self.bert\_tokenizer.encode\_plus(

26.             input\_text,

27.             add\_special\_tokens=True,

28.             max\_length=512,

29.             return\_tensors='pt'

30.         )

31.         code\_description = self.bert\_tokenizer.decode(bert\_inputs['input\_ids'][0], skip\_special\_tokens=True)

32.         blackbox\_code = self.blackbox\_model.generate(code\_description, language='python')

33.         return blackbox\_code

34.

35.     def ensemble\_code(self, code1, code2):

36.         if code1 == code2:

37.             return code1

38.         else:

39.             return code1 + '\n' + code2

40.

41.     def generate\_code(self, input\_text):

42.         # Generate code using CodeBERT

43.         codebert\_code = self.generate\_code\_with\_codebert(input\_text)

44.

45.         # Generate code using BERT with Blackbox

46.         blackbox\_code = self.generate\_code\_with\_blackbox(input\_text)

47.

48.         # Combine the outputs using an ensemble method

49.         ensemble\_code = self.ensemble\_code(codebert\_code, blackbox\_code)

50.

51.         return ensemble\_code

52.

### features\dialogue\_manager.py

**Purpose**: This file is responsible for managing the flow of conversations and dialogue between Jaicat and the user. It keeps track of context and handles switching between tasks in a conversation.

#### Key Functions:

* + **manage\_dialogue()**: Manages back-and-forth conversation flow and keeps track of user context.
* **Connection to main.py**:
  + Called by main.py to manage conversational responses and flow after user input is processed.

#### The code

1. from conversation.nlu import NLU

2. from transformers import BertForSequenceClassification, BertTokenizer

3. import spacy

4. import torch

5. from conversation.nlg import NLG

6.

7. class DialogueManager:

8.     def \_\_init\_\_(self):

9.         self.nlp = spacy.load("en\_core\_web\_sm")

10.         self.nlu = NLU()

11.         self.response\_generator = BertForSequenceClassification.from\_pretrained("bert-base-uncased")

12.         self.tokenizer = BertTokenizer.from\_pretrained("bert-base-uncased")

13.

14.     def process\_user\_input(self, user\_input):

15.         extracted\_data = self.nlu.extract\_intent\_and\_entities(user\_input)

16.         intent = extracted\_data["intent"]

17.         entities = extracted\_data["entities"]

18.

19.         if intent is None:

20.             return "I'm sorry, I didn't understand that. Could you please rephrase?"

21.

22.         if intent == "greet":

23.             return self.greet()

24.         elif intent == "goodbye":

25.             return self.goodbye()

26.         elif intent == "generate\_code":

27.             return self.generate\_code(entities)

28.         elif intent == "analyze\_text":

29.             return self.analyze\_text(entities)

30.         elif intent == "analyze\_sentiment":

31.             return self.analyze\_sentiment(entities)

32.

33.     def generate\_code(self, entities):

34.         input\_ids = self.tokenizer.encode("Generate code for " + entities[0]["text"], return\_tensors="pt")

35.         attention\_mask = self.tokenizer.encode("Generate code for " + entities[0]["text"], return\_tensors="pt", max\_length=512, truncation=True)[1]

36.         outputs = self.response\_generator(input\_ids, attention\_mask=attention\_mask)

37.         response = self.tokenizer.decode(outputs[0], skip\_special\_tokens=True)

38.         return response

39.

40.     def analyze\_text(self, entities):

41.         # Implement text analysis logic here

42.         # For example, you can use the LLaMA model to analyze text

43.         input\_ids = self.tokenizer.encode("Analyze text: " + entities[0]["text"], return\_tensors="pt")

44.         attention\_mask = self.tokenizer.encode("Analyze text: " + entities[0]["text"], return\_tensors="pt", max\_length=512, truncation=True)

45.         outputs = self.response\_generator(input\_ids, attention\_mask=attention\_mask)

46.         response = self.tokenizer.decode(outputs[0], skip\_special\_tokens=True)

47.         return response

48.

49.     def analyze\_sentiment(self, entities):

50.         # Implement sentiment analysis logic here

51.         # For example, you can use the LLaMA model to analyze sentiment

52.         input\_ids = self.tokenizer.encode("Analyze sentiment of: " + entities[0]["text"], return\_tensors="pt")

53.         attention\_mask = self.tokenizer.encode("Analyze sentiment of: " + entities[0]["text"], return\_tensors="pt", max\_length=512, truncation=True)

54.         outputs = self.response\_generator(input\_ids, attention\_mask=attention\_mask)

55.         response = self.tokenizer.decode(outputs[0], skip\_special\_tokens=True)

56.         return response

57.

58.     def greet(self):

59.         return "Hello! How can I help you?"

60.

61.     def goodbye(self):

62.         return "Goodbye! Have a great day!"

63.

### features\nlp copy.py

**Purpose**: A backup or alternative version of the main nlp.py file. It could have differences in implementation or be used for testing purposes.

* **Key Functions**:
  + Likely similar to nlp.py, handling natural language processing tasks such as entity extraction and sentiment analysis.
* **Connection to main.py**:
  + No direct connection to main.py, but could serve as a backup for reference.

#### The code

1. # In features/nlp.py

2.

3. import spacy

4. import torch

5. from transformers import BertTokenizer, BertForSequenceClassification

6.

7. class NLPSystem:

8.     def \_\_init\_\_(self):

9.         self.nlp = spacy.load('en\_core\_web\_lg')

10.         self.sentiment\_analysis = SentimentAnalysis()

11.

12.     def analyze\_text(self, text):

13.         try:

14.             doc = self.nlp(text)

15.             entities = [(ent.text, ent.label\_) for ent in doc.ents]

16.             return entities

17.         except Exception as e:

18.             return f"Error: {e}"

19.

20.     def part\_of\_speech\_tagging(self, text):

21.         try:

22.             doc = self.nlp(text)

23.             pos\_tags = [(token.text, token.pos\_) for token in doc]

24.             return pos\_tags

25.         except Exception as e:

26.             return f"Error: {e}"

27.

28.     def language\_detection(self, text):

29.         lang = self.nlp(text).\_language

30.         return "English" if lang == 'en' else 'Unknown'

31.

32.     def text\_summarization(self, text, max\_length):

33.         try:

34.             model\_name = "bert-base-uncased"

35.             tokenizer = BertTokenizer.from\_pretrained(model\_name)

36.             model = BertForSequenceClassification.from\_pretrained(model\_name)

37.

38.             inputs = tokenizer.encode\_plus(

39.                 text,

40.                 max\_length=max\_length,

41.                 padding="max\_length",

42.                 truncation=True,

43.                 return\_tensors="pt",

44.             )

45.

46.             input\_ids = inputs["input\_ids"].unsqueeze(0)

47.             attention\_mask = inputs["attention\_mask"].unsqueeze(0)

48.

49.             with torch.no\_grad():

50.                 outputs = model(input\_ids, attention\_mask=attention\_mask)

51.                 hidden\_states = outputs.hidden\_states

52.

53.             last\_layer\_output = hidden\_states[-1][0]

54.             importance\_scores = torch.sum(last\_layer\_output \* attention\_mask[0], dim=1)

55.             sorted\_indices = torch.argsort(importance\_scores, descending=True)

56.

57.             summary = ""

58.             for sentence\_idx in sorted\_indices:

59.                 summary += tokenizer.convert\_ids\_to\_tokens(input\_ids[0, sentence\_idx].tolist())[1:-1] + " "

60.                 if len(summary) > max\_length:

61.                     break

62.

63.             return summary.strip()

64.

65.         except Exception as e:

66.             return f"Error: {e}"

67.

68.     def process(self, command):

69.         command = command.lower()

70.

71.         if "what is" in command or "how to" in command or "who" in command:

72.             response = self.ask\_question(command)

73.             return response

74.

75.         if "analyze text" in command:

76.             text = command.replace("analyze text", "").strip()

77.             entities = self.analyze\_text(text)

78.             return f"Entities found: {entities}"

79.

80.         elif "sentiment" in command:

81.             text = command.replace("analyze sentiment", "").strip()

82.             sentiment = self.sentiment\_analysis.live\_test(text)

83.             return f"Sentiment: {sentiment}"

84.

85.         elif "part of speech tagging" in command:

86.             text = command.replace("part of speech tagging", "").strip()

87.             pos\_tags = self.part\_of\_speech\_tagging(text)

88.             return f"Part of Speech Tagging: {pos\_tags}"

89.

90.         elif "language detection" in command:

91.             text = command.replace("language detection", "").strip()

92.             language = self.language\_detection(text)

93.             return f"Language: {language}"

94.

95.         elif "text summarization" in command:

96.             text = command.replace("text summarization", "").strip()

97.             summary = self.text\_summarization(text, 100)

98.             return f"Summary: {summary}"

99.

100.         elif "generate code" in command:

101.             prompt = command.replace("generate code", "").strip()

102.             generated\_code = self.code\_generation\_model.generate\_code(prompt)

103.             return f"Generated code: {generated\_code}"

104.

105.         # Add more command processing as needed

106.         return "Command not recognized."

107.

108. class SentimentAnalysis:

109.     def \_\_init\_\_(self, model\_name="bert-base-uncased"):

110.         self.tokenizer = BertTokenizer.from\_pretrained(model\_name)

111.         self.model = BertForSequenceClassification.from\_pretrained(model\_name, num\_labels=2)

112.

113.     def preprocess\_text(self, text):

114.         try:

115.             inputs = self.tokenizer.encode\_plus(

116.                 text,

117.                 add\_special\_tokens=True,

118.                 max\_length=512,

119.                 return\_attention\_mask=True,

120.                 return\_tensors="pt"

121.             )

122.             return inputs

123.         except Exception as e:

124.             return f"Error: {e}"

125.

126.     def analyze\_sentiment(self, text):

127.         try:

128.             inputs = self.preprocess\_text(text)

129.             outputs = self.model(\*\*inputs)

130.             logits = outputs.logits

131.             sentiment = torch.argmax(logits)

132.             return sentiment.item()

133.         except Exception as e:

134.             return f"Error: {e}"

135.

136.     def live\_test(self, text):

137.         sentiment = self.analyze\_sentiment(text)

138.         return "Positive" if sentiment == 1 else "Negative"

139.

### features\nlp.py

**Purpose**: Implements natural language processing (NLP) for Jaicat, using models such as BERT and SpaCy for text analysis, sentiment analysis, and entity extraction.

* **Key Functions**:
  + **analyze\_text()**: Extracts entities from a given text.
  + **part\_of\_speech\_tagging()**: Tags parts of speech in user input.
  + **process(command)**: Processes user commands and routes them through appropriate NLP tasks like sentiment analysis.
* **Connection to main.py**:
  + main.py calls nlp.py to process user input, extracting meaning and intent from the text.

The Code

1. import torch

2. import spacy

3. from transformers import BertTokenizer, BertForSequenceClassification

4.

5. class NLPSystem:

6.     def \_\_init\_\_(self):

7.         # Load the SpaCy language model

8.         self.nlp = spacy.load('en\_core\_web\_lg')

9.

10.         # Initialize sentiment analysis using BERT

11.         self.sentiment\_analysis\_model = SentimentAnalysis()

12.

13.     def analyze\_text(self, text):

14.         """Analyze text to extract named entities."""

15.         doc = self.nlp(text)

16.         entities = [(ent.text, ent.label\_) for ent in doc.ents]

17.         return entities

18.

19.     def part\_of\_speech\_tagging(self, text):

20.         """Perform part-of-speech tagging on the input text."""

21.         doc = self.nlp(text)

22.         pos\_tags = [(token.text, token.pos\_) for token in doc]

23.         return pos\_tags

24.

25.     def language\_detection(self, text):

26.         """Detect the language of the input text."""

27.         lang = self.nlp(text).\_.language

28.         return "English" if lang == 'en' else 'Unknown'

29.

30.     def text\_summarization(self, text, max\_length=100):

31.         """Summarize the input text (placeholder for future implementation)."""

32.         # Summarization logic to be added, possibly using transformers

33.         pass

34.

35.     def process(self, command):

36.         """Process user commands for different NLP tasks."""

37.         command = command.lower()

38.

39.         if "analyze text" in command:

40.             text = command.replace("analyze text", "").strip()

41.             entities = self.analyze\_text(text)

42.             return f"Entities found: {entities}"

43.

44.         elif "analyze sentiment" in command:

45.             text = command.replace("analyze sentiment", "").strip()

46.             sentiment = self.sentiment\_analysis\_model.analyze\_sentiment(text)

47.             return f"Sentiment: {sentiment}"

48.

49.         elif "part of speech tagging" in command:

50.             text = command.replace("part of speech tagging", "").strip()

51.             pos\_tags = self.part\_of\_speech\_tagging(text)

52.             return f"Part of Speech Tagging: {pos\_tags}"

53.

54.         # Add other NLP-related tasks here, like summarization, translation, etc.

55.         return "Unknown command. Please try again."

56.

57. class SentimentAnalysis:

58.     def \_\_init\_\_(self, model\_name="nlptown/bert-base-multilingual-uncased-sentiment"):

59.         """Initialize BERT model for sentiment analysis."""

60.         self.tokenizer = BertTokenizer.from\_pretrained(model\_name)

61.         self.model = BertForSequenceClassification.from\_pretrained(model\_name)

62.

63.     def analyze\_sentiment(self, text):

64.         """Analyze sentiment using BERT sentiment model."""

65.         inputs = self.tokenizer.encode\_plus(text, return\_tensors="pt", truncation=True)

66.         outputs = self.model(\*\*inputs)

67.

68.         # Apply softmax to get probabilities

69.         scores = torch.nn.functional.softmax(outputs.logits, dim=1)

70.

71.         # Sentiment is determined by the highest score

72.         sentiment = torch.argmax(scores).item()

73.

74.         if sentiment >= 4:

75.             return "positive"

76.         elif sentiment <= 1:

77.             return "negative"

78.         else:

79.             return "neutral"

80.

### features\nlu.py

**Purpose**: This file handles natural language understanding (NLU) tasks, including intent detection and entity recognition.

* **Key Functions**:
  + **extract\_intent\_and\_entities()**: Detects the user's intent and extracts entities from the input.
  + **predict\_intent()**: Uses machine learning to predict the user's intent.
  + **extract\_entities()**: Extracts entities like names, dates, and locations from user input.
* **Connection to main.py**:
  + When Jaicat needs to understand a user’s intent, main.py calls this file to process and extract actionable information.

##### The Code

1. import spacy

2. from spacy import displacy

3.

4. class NLU:

5.     def \_\_init\_\_(self):

6.         self.nlp = spacy.load("en\_core\_web\_sm")

7.

8.     def extract\_intent\_and\_entities(self, user\_input):

9.         try:

10.             doc = self.nlp(user\_input)

11.             intent = self.detect\_intent(doc)

12.             entities = self.extract\_entities(doc)

13.             return {"intent": intent, "entities": entities}

14.         except Exception as e:

15.             return {"error": str(e)}

16.

17.     def detect\_intent(self, doc):

18.         # Implement intent detection logic here

19.         # For example, you can use spaCy's entity recognition to identify intents

20.         intent\_entities = [ent.text for ent in doc.ents if ent.label\_ == "INTENT"]

21.         if intent\_entities:

22.             return intent\_entities[0]

23.         else:

24.             return None

25.

26.     def extract\_entities(self, doc):

27.         # Implement entity recognition logic here

28.         # For example, you can use spaCy's entity recognition to extract entities

29.         entities = [(ent.text, ent.label\_) for ent in doc.ents]

30.         return entities

31.

32.     def visualize\_entities(self, doc):

33.         # Visualize entities using spaCy's displacy

34.         displacy.render(doc, style="ent")

35.

36. # Test the code

37. nlu = NLU()

38. user\_input = "I want to book a flight from New York to Los Angeles"

39. result = nlu.extract\_intent\_and\_entities(user\_input)

40. print(result)

41.

42. # Visualize entities

43. nlu.visualize\_entities(nlu.nlp(user\_input))

44.

### features\sentiment\_analysis.py

**Purpose**: Another file that handles sentiment analysis. It may either be a backup or an alternative approach to the sentiment analysis task.

* **Key Functions**:
  + **analyze\_sentiment()**: Likely similar to SentimentAnalysis.py, it returns the sentiment based on the text.
* **Connection to main.py**:
  + Could be used interchangeably with SentimentAnalysis.py for sentiment-related tasks.

##### The Code

1. # C:\Users\josh\_\Desktop\jaicat\_project\features\sentiment\_analysis.py

2.

3. class SentimentAnalysisModel:

4.     def \_\_init\_\_(self):

5.         # Initialize the model

6.         pass

7.

8.     def analyze\_sentiment(self, text):

9.         # Analyze the sentiment of the text

10.         return "Sentiment analysis result"

11.

### features\SentimentAnalysis.py

**Purpose**: Handles sentiment analysis of user input using a model like BERT.

* **Key Functions**:
  + **analyze\_sentiment()**: Returns whether the input sentiment is positive, neutral, or negative.
* **Connection to main.py**:
  + Used by main.py when determining the emotional tone of the user’s input to adapt Jaicat's responses accordingly.

##### The code

1. # C:\Users\josh\_\Desktop\jaicat\_project\features\sentiment\_analysis.py

2.

3. class SentimentAnalysisModel:

4.     def \_\_init\_\_(self):

5.         # Initialize the model

6.         pass

7.

8.     def analyze\_sentiment(self, text):

9.         # Analyze the sentiment of the text

10.         return "Sentiment analysis result"

11.

### features\text\_classification.py

**Purpose**: Handles text classification, likely using machine learning models such as RandomForestClassifier or BERT to classify input into categories.

* **Key Functions**:
  + **classify\_text()**: Classifies the input text into predefined categories (e.g., question, command, feedback).
* **Connection to main.py**:
  + If Jaicat needs to classify a user’s input for specific actions, main.py will call this file to categorize the input and trigger the appropriate response.

##### The Code

1. # C:\Users\josh\_\Desktop\jaicat\_project\features\text\_classification.py

2.

3. class TextClassificationModel:

4.     def \_\_init\_\_(self):

5.         # Initialize the model

6.         pass

7.

8.     def classify\_text(self, text):

9.         # Classify the input text

10.         return "Classified text"

11.

12.

## lib

### lib\nlg

#### lib\nlg\Social\_media\_post.py

**Purpose**: Likely generates or manages posts for social media platforms using natural language generation (NLG) techniques.

* **Key Functions**:
  + **Generate Social Media Posts**: Uses NLG techniques to create posts based on a prompt or set of input data.
* **Connection to main.py**:
  + If Jaicat is asked to generate a social media post, main.py would call this file to construct the message.

##### The code

1. # lib/nlg/Social\_media\_post.py

2.

3. import random

4. import textwrap

5.

6. class SocialMediaPostGenerator:

7.     def \_\_init\_\_(self):

8.         self.templates = [

9.             "Check out our latest blog post: {link} #blog #update",

10.             "We're excited to announce that {event} is happening! Join us! #event #announcement",

11.             "Have you seen our new product? {product\_name} is now available! #newproduct #shopnow",

12.             "Don't miss out on our special offer: {offer} #discount #sale",

13.             "Join us for a live session on {date}. Learn more: {link} #live #webinar",

14.         ]

15.

16.     def generate\_post(self, context):

17.         """Generate a social media post based on the given context."""

18.         template = random.choice(self.templates)

19.         return template.format(\*\*context)

20.

21.     def format\_post(self, post):

22.         """Format the post for readability."""

23.         wrapped\_post = textwrap.fill(post, width=60)  # Wrap the text to a specified width

24.         return wrapped\_post

25.

26.     def analyze\_post(self, post):

27.         """Analyze the post for sentiment and engagement."""

28.         # Placeholder for sentiment analysis logic

29.         # In a real implementation, you might integrate an NLP library to analyze the post

30.         return {

31.             "length": len(post.split()),

32.             "sentiment": "neutral",  # Replace with actual sentiment analysis

33.             "engagement\_score": random.randint(1, 10)  # Placeholder for engagement score

34.         }

35.

36. # Example usage

37. if \_\_name\_\_ == "\_\_main\_\_":

38.     generator = SocialMediaPostGenerator()

39.

40.     context = {

41.         "link": "https://example.com/blog",

42.         "event": "our annual conference",

43.         "product\_name": "Amazing Gadget",

44.         "offer": "20% off on all items",

45.         "date": "October 15, 2024"

46.     }

47.

48.     post = generator.generate\_post(context)

49.     formatted\_post = generator.format\_post(post)

50.

51.     print("Generated Post:")

52.     print(formatted\_post)

53.

54.     analysis = generator.analyze\_post(formatted\_post)

55.     print("\nPost Analysis:")

56.     print(analysis)

57.

##### 

### lib\nlp

#### lib\nlp\crime.py

**Purpose**: Contains NLP utilities specific to analyzing or detecting crime-related information in text.

* **Key Functions**:
  + **Crime Detection**: Could analyze crime reports or identify crime-related entities in user input.
* **Connection to main.py**:
  + If Jaicat is asked to process crime-related text or extract crime-specific entities, main.py could call this file.

The Code

1. # lib/nlp/crime.py

2.

3. import pandas as pd

4.

5. class CrimeAnalyzer:

6. def \_\_init\_\_(self, data\_path):

7. """

8. Initialize the CrimeAnalyzer with a path to the crime data CSV file.

9. Parameters:

10. - data\_path: Path to the CSV file containing crime data.

11. """

12. self.data\_path = data\_path

13. self.crime\_data = self.load\_data()

14.

15. def load\_data(self):

16. """Load crime data from a CSV file."""

17. try:

18. data = pd.read\_csv(self.data\_path)

19. return data

20. except Exception as e:

21. print(f"Error loading data: {e}")

22. return None

23.

24. def get\_crime\_statistics(self):

25. """Get basic statistics about the crime data."""

26. if self.crime\_data is not None:

27. return self.crime\_data.describe()

28. return None

29.

30. def analyze\_crime\_type(self, crime\_type):

31. """

32. Analyze the occurrences of a specific crime type.

33. Parameters:

34. - crime\_type: The type of crime to analyze.

35. Returns:

36. - The number of occurrences of the specified crime type.

37. """

38. if self.crime\_data is not None:

39. return self.crime\_data[self.crime\_data['crime\_type'] == crime\_type].shape[0]

40. return 0

41.

42. def get\_top\_crime\_types(self, n=5):

43. """

44. Get the top N crime types based on occurrences.

45. Parameters:

46. - n: The number of top crime types to return.

47. Returns:

48. - A list of top N crime types and their counts.

49. """

50. if self.crime\_data is not None:

51. return self.crime\_data['crime\_type'].value\_counts().head(n)

52. return None

53.

54. def filter\_crimes\_by\_location(self, location):

55. """

56. Filter crime data by a specific location.

57. Parameters:

58. - location: The location to filter by.

59. Returns:

60. - Filtered DataFrame with crimes in the specified location.

61. """

62. if self.crime\_data is not None:

63. return self.crime\_data[self.crime\_data['location'] == location]

64. return None

65.

#### lib\nlp\engineering.py

**Purpose**: Handles NLP tasks related to engineering terminology and language.

* **Key Functions**:
  + **Engineering Term Recognition**: Could identify and process engineering terms or jargon in text.
* **Connection to main.py**:
  + If Jaicat needs to process technical input in the engineering domain, main.py would reference this file.

The Code

1. # lib/nlp/engineering.py

2.

3. class EngineeringNLP:

4.     def \_\_init\_\_(self):

5.         pass

6.

7.     def calculate\_force(self, mass, acceleration):

8.         """

9.         Calculate the force based on mass and acceleration using Newton's second law.

10.

11.         Parameters:

12.         - mass: Mass in kilograms.

13.         - acceleration: Acceleration in m/s^2.

14.

15.         Returns:

16.         - Force in Newtons.

17.         """

18.         force = mass \* acceleration

19.         return force

20.

21.     def calculate\_work(self, force, distance):

22.         """

23.         Calculate work done based on force and distance.

24.

25.         Parameters:

26.         - force: Force in Newtons.

27.         - distance: Distance in meters.

28.

29.         Returns:

30.         - Work done in Joules.

31.         """

32.         work = force \* distance

33.         return work

34.

35.     def calculate\_energy(self, mass, height):

36.         """

37.         Calculate potential energy based on mass and height.

38.

39.         Parameters:

40.         - mass: Mass in kilograms.

41.         - height: Height in meters.

42.

43.         Returns:

44.         - Potential energy in Joules.

45.         """

46.         g = 9.81  # Acceleration due to gravity in m/s^2

47.         potential\_energy = mass \* g \* height

48.         return potential\_energy

49.

50.     def structural\_analysis(self, load, length, width, height):

51.         """

52.         Perform a simple structural analysis for a beam.

53.

54.         Parameters:

55.         - load: Load applied to the beam in Newtons.

56.         - length: Length of the beam in meters.

57.         - width: Width of the beam in meters.

58.         - height: Height of the beam in meters.

59.

60.         Returns:

61.         - Bending stress in Pascals.

62.         """

63.         area\_moment = (width \* height\*\*3) / 12  # Moment of inertia

64.         bending\_stress = (load \* length) / (2 \* area\_moment)  # Bending stress formula

65.         return bending\_stress

66.

67.

68.

#### lib\nlp\fitness.py

**Purpose**: Manages NLP tasks related to fitness, likely identifying and analyzing fitness-related language.

* **Key Functions**:
  + **Fitness Term Detection**: Could analyze and extract fitness-related terms from user input (e.g., workout names, fitness goals).
* **Connection to main.py**:
  + main.py would reference this file when processing fitness-related commands or queries from the user.

The Code

1. # lib/nlp/fitness.py

2.

3. class FitnessNLP:

4. def \_\_init\_\_(self):

5. pass

6.

7. def analyze\_fitness\_data(self, data):

8. """

9. Analyze the input fitness data and provide insights.

10.

11. Parameters:

12. - data: A dictionary containing fitness data such as steps, calories burned, and workout duration.

13.

14. Returns:

15. - A string summary of the fitness data analysis.

16. """

17. steps = data.get('steps', 0)

18. calories\_burned = data.get('calories\_burned', 0)

19. workout\_duration = data.get('workout\_duration', 0) # in minutes

20.

21. analysis = f"You have taken {steps} steps, burned {calories\_burned} calories, and worked out for {workout\_duration} minutes."

22. return analysis

23.

24. def recommend\_workout(self, fitness\_level):

25. """

26. Recommend a workout based on the user's fitness level.

27.

28. Parameters:

29. - fitness\_level: The fitness level of the user (e.g., beginner, intermediate, advanced).

30.

31. Returns:

32. - A workout recommendation.

33. """

34. workouts = {

35. "beginner": "Try a 20-minute brisk walk or light yoga.",

36. "intermediate": "Consider a 30-minute jog or a strength training session.",

37. "advanced": "Challenge yourself with a 45-minute HIIT workout or heavy weightlifting."

38. }

39.

40. return workouts.get(fitness\_level.lower(), "No workout recommendations available.")

41.

42. def set\_fitness\_goal(self, goal):

43. """

44. Set a fitness goal for the user.

45.

46. Parameters:

47. - goal: A string indicating the fitness goal (e.g., "lose weight", "gain muscle", "run a marathon").

48.

49. Returns:

50. - A message confirming the goal has been set.

51. """

52. return f"Your fitness goal has been set to: {goal}"

53.

#### lib\nlp\medical.py

**Purpose**: Contains utilities for processing medical terminology or health-related text.

* **Key Functions**:
  + **Medical Term Extraction**: Analyzes user input for medical terms or symptoms.
* **Connection to main.py**:
  + If Jaicat is used for health-related tasks, main.py will call this file to handle the medical language processing.

The Code

1. # lib/nlp/medical.py

2.

3. import spacy

4.

5. class MedicalNLP:

6.     def \_\_init\_\_(self, model='en\_core\_web\_sm'):

7.         # Load the SpaCy language model

8.         self.nlp = spacy.load(model)

9.

10.     def extract\_medical\_entities(self, text):

11.         """

12.         Extract medical-related entities from the input text.

13.

14.         Parameters:

15.         - text: The medical text to analyze.

16.

17.         Returns:

18.         - A list of tuples containing the entity text and its label.

19.         """

20.         doc = self.nlp(text)

21.         medical\_entities = []

22.

23.         for ent in doc.ents:

24.             if ent.label\_ in ["DISEASE", "SYMPTOM", "MEDICATION", "TREATMENT"]:

25.                 medical\_entities.append((ent.text, ent.label\_))

26.

27.         return medical\_entities

28.

29.     def classify\_symptoms(self, symptoms):

30.         """

31.         Classify the input symptoms into categories.

32.

33.         Parameters:

34.         - symptoms: A list of symptom strings to classify.

35.

36.         Returns:

37.         - A dictionary categorizing symptoms.

38.         """

39.         symptom\_categories = {

40.             "Respiratory": [],

41.             "Gastrointestinal": [],

42.             "Neurological": [],

43.             "Cardiovascular": [],

44.             "Others": []

45.         }

46.

47.         for symptom in symptoms:

48.             # Simple keyword matching for classification (this could be replaced with a more sophisticated model)

49.             if "cough" in symptom or "breath" in symptom:

50.                 symptom\_categories["Respiratory"].append(symptom)

51.             elif "nausea" in symptom or "vomit" in symptom:

52.                 symptom\_categories["Gastrointestinal"].append(symptom)

53.             elif "headache" in symptom:

54.                 symptom\_categories["Neurological"].append(symptom)

55.             elif "chest pain" in symptom:

56.                 symptom\_categories["Cardiovascular"].append(symptom)

57.             else:

58.                 symptom\_categories["Others"].append(symptom)

59.

60.         return symptom\_categories

61.

62.     def analyze\_medical\_text(self, text):

63.         """

64.         Analyze the input medical text for entities and symptoms.

65.

66.         Parameters:

67.         - text: The medical text to analyze.

68.

69.         Returns:

70.         - A dictionary containing extracted entities and categorized symptoms.

71.         """

72.         entities = self.extract\_medical\_entities(text)

73.         symptoms = [ent[0] for ent in entities if ent[1] == "SYMPTOM"]

74.         symptom\_classification = self.classify\_symptoms(symptoms)

75.

76.         return {

77.             'entities': entities,

78.             'symptom\_classification': symptom\_classification

79.         }

#### lib\nlp\motorbike\_engine.py

**Purpose**: Deals with NLP tasks related to motorbike engines and their components.

* **Key Functions**:
  + **Motorbike Engine Processing**: Could extract motorbike-specific terminology from user input.
* **Connection to main.py**:
  + If Jaicat needs to understand or process information about motorbike engines, main.py would call this file.

##### The Code

import wikipedia

2.

3. # Define a function to retrieve information on motorbikes

4. get\_motorbike\_info(query):

5. results = wikipedia.search(query)

6. if results:

7. page = wikipedia.page(results[0])

8. return page.content

9. else:

10. return "I couldn't find any information on that motorbike."

11.

12. # Define a function to retrieve information on engines

13. get\_engine\_info(query):

14. results = wikipedia.search(query)

15. if results:

16. page = wikipedia.page(results[0])

17. return page.content

18. else:

19. return "I couldn't find any information on that engine."

20.

21. # Define a function to provide motorbike and engine information to the user

22. provide\_motorbike\_engine\_info(query):

23. if "motorbike" in query:

24. return get\_motorbike\_info(query)

25. elif "engine" in query:

26. return get\_engine\_info(query)

27. else:

28. return "I'm not sure what you're looking for. Can you be more specific?"

29.

30. # Example usage:

31. query = "What is a Harley-Davidson?"

32. print(provide\_motorbike\_engine\_info(query))

33.

34. query = "How does a V8 engine work?"

35. print(provide\_motorbike\_engine\_info(query))

36.

#### lib\nlp\nltk\_utils.py

**Purpose**: Provides helper functions or utilities using the NLTK (Natural Language Toolkit) library for NLP tasks.

##### Key Functions:

* + **Text Processing**: Provides helper functions like tokenization, stemming, and lemmatization.
* **Connection to main.py**:
  + This utility could be used by other NLP tasks in main.py for basic text processing operations.

##### The Code

1. import nltk

2. from nltk.tokenize import word\_tokenize

3. from nltk.corpus import stopwords

4. from nltk.stem import WordNetLemmatizer

5.

6. def preprocess\_data(text):

7.     # Tokenize the text

8.     tokens = word\_tokenize(text)

9.

10.     # Remove stopwords

11.     stop\_words = set(stopwords.words('english'))

12.     tokens = [token for token in tokens if token not in stop\_words]

13.

14.     # Lemmatize the tokens

15.     lemmatizer = WordNetLemmatizer()

16.     tokens = [lemmatizer.lemmatize(token) for token in tokens]

17.

18.     # Join the tokens back into a string

19.     text = ' '.join(tokens)

20.

21.     return text

22.

#### lib\nlp\politics.py

**Purpose**: Handles NLP tasks related to political terminology and language.

##### Key Functions:

* + **Political Term Extraction**: Analyzes and processes political terms in user input.
* **Connection to main.py**:
  + If Jaicat is asked to understand or process political topics, main.py will reference this file.

##### The Code

1. import nltk

2. from nltk.tokenize import word\_tokenize

3. from nltk.corpus import stopwords

4. from nltk.stem import WordNetLemmatizer

5. import spacy

6. from spacy import displacy

7. import pandas as pd

8. from PIL import Image, ImageTk

9. import tkinter as tk

10. from tkinter import filedialog

11. import PyPDF2

12. import requests

13. from bs4 import BeautifulSoup

14. import cv2

15. import sqlite3

16. from lib.nlp.nltk\_utils import preprocess\_data

17. from database.knowledge\_graph import KnowledgeGraph

18. from conversation.nlu import extract\_entities, extract\_intent

19. from database.web\_scraping import Webscraping

20.

21. # Load the NLTK data needed for the task

22. nltk.download('punkt')

23. nltk.download('wordnet')

24. nltk.download('stopwords')

25.

26. # Load the Spacy model for English

27. nlp = spacy.load("en\_core\_web\_sm")

28.

29. class Politics:

30.     def \_\_init\_\_(self):

31.         self.data = []

32.         self.knowledge\_graph = {}

33.

34.     def collect\_data(self):

35.         # Web scraping code here

36.         self.webscraping = Webscraping()

37.         return self.webscraping.collect\_data()

38.

39.     def preprocess\_data(self, text):

40.         return preprocess\_data(text)

41.

42.     def recognize\_entities(self, text):

43.         return extract\_entities(text)

44.

45.     def create\_knowledge\_graph(self):

46.         kg = KnowledgeGraph()

47.         return kg.create\_knowledge\_graph()

48.

49.     def answer\_questions(self, query):

50.             # Extract entities and intent from the query

51.             doc = nlp(query)

52.             entities = [(ent.text, ent.label\_) for ent in doc.ents]

53.             intent = extract\_intent(query)

54.

55.             # Create a knowledge graph instance

56.             kg = KnowledgeGraph()

57.

58.             # Use the knowledge graph to answer the question

59.             answer = kg.answer\_question(entities, intent)

60.

61.             return answer

62.

63.     def display\_info(self, info\_type, info):

64.         # Display information code here

65.         if info\_type == "pdf":

66.             self.display\_pdf(info)

67.         elif info\_type == "webpage":

68.             self.display\_webpage(info)

69.         elif info\_type == "image":

70.             self.display\_image(info)

71.         elif info\_type == "video":

72.             self.display\_video(info)

73.

74.     def display\_pdf(self, pdf\_file):

75.         # Display PDF code here

76.         pdf\_file = filedialog.askopenfilename(title="Select PDF file", filetypes=[("PDF files", "\*.pdf")])

77.         pdf\_reader = PyPDF2.PdfFileReader(pdf\_file)

78.         page = pdf\_reader.getPage(0)

79.         text = page.extractText()

80.         self.display\_text(text)

81.

82.     def display\_webpage(self, url):

83.         # Display webpage code here

84.         response = requests.get(url)

85.         soup = BeautifulSoup(response.content, 'html.parser')

86.         text = soup.get\_text()

87.         self.display\_text(text)

88.

89.     def display\_image(self, image\_file):

90.         # Display image code here

91.         image\_file = filedialog.askopenfilename(title="Select image file", filetypes=[("Image files", "\*.jpg \*.png")])

92.         image = Image.open(image\_file)

93.         image\_tk = ImageTk.PhotoImage(image)

94.         self.display\_image\_tk(image\_tk)

95.

96.     def display\_video(self, video\_file):

97.         # Display video code here

98.         video\_file = filedialog.askopenfilename(title="Select video file", filetypes=[("Video files", "\*.mp4")])

99.         cap = cv2.VideoCapture(video\_file)

100.         while True:

101.             ret, frame = cap.read()

102.             if not ret:

103.                 break

104.             cv2.imshow('Video', frame)

105.             if cv2.waitKey(1) & 0xFF == ord('q'):

106.                 break

107.         cap.release()

108.         cv2.destroyAllWindows()

109.

110.     def display\_text(self, text):

111.         # Display text code here

112.         root = tk.Tk()

113.         root.title("Text Display")

114.         text\_box = tk.Text(root)

115.         text\_box.pack()

116.         text\_box.insert(tk.END, text)

117.         root.mainloop()

118.

119.     def display\_image\_tk(self, image\_tk):

120.         # Display image code here

121.         root = tk.Tk()

122.         root.title("Image Display")

123.         image\_label = tk.Label(root, image=image\_tk)

124.         image\_label.pack()

125.         root.mainloop()

126.

127.     def store\_data(self, title, main\_content):

128.         conn = sqlite3.connect('webscraping.db')

129.         c = conn.cursor()

130.         c.execute('CREATE TABLE IF NOT EXISTS webscraping (title TEXT, main\_content TEXT)')

131.         c.execute('INSERT INTO webscraping (title, main\_content) VALUES (?, ?)', (title, main\_content))

132.         conn.commit()

133.         conn.close()

134.

135. # Create an instance of the Politics class

136. politics = Politics()

137.

138.  # Test the collect\_data method

139. title, main\_content = politics.collect\_data()

140. print(title)

141. print(main\_content)

142.

#### lib\nlp\sexual\_content.py

**Purpose**:

##### Key Functions:

* + **C**
* **Connection to main.py**:

##### The Code

1. import nltk

2. from nltk.corpus import wordnet

3.

4. def get\_synonyms(word):

5.     # Get synonyms for a given word

6.     synonyms = []

7.     for syn in wordnet.synsets(word):

8.         for lemma in syn.lemmas():

9.             synonyms.append(lemma.name())

10.     return synonyms

11.

12. def generate\_sexual\_content():

13.     # Generate sexual content using synonyms

14.     word = "sexy"

15.     synonyms = get\_synonyms(word)

16.     content = f"This {word} {synonyms[0]} content is designed to be engaging and informative."

17.     return content

18.

19. import random

20.

21. # Define a dictionary of sexual content topics

22. sexual\_content\_topics = {

23.     "anatomy": ["What is the difference between a vagina and a vulva?", "What are the different types of sexual orientation?"],

24.     "physiology": ["How does sexual arousal work?", "What are the physical effects of orgasm?"],

25.     "intimacy": ["What are some ways to build intimacy with a partner?", "How can I communicate my desires to my partner?"],

26.     "consent": ["What is enthusiastic consent?", "How can I ensure I have consent before engaging in sexual activity?"],

27.     "relationships": ["What are some common issues in romantic relationships?", "How can I maintain a healthy and fulfilling relationship?"]

28. }

29.

30. # Define a function to respond to user queries about sexual content

31. def respond\_to\_sexual\_content\_query(query):

32.     # Check if the query is related to sexual content

33.     if any(topic in query.lower() for topic in sexual\_content\_topics.keys()):

34.         # Choose a random topic from the dictionary

35.         topic = random.choice(list(sexual\_content\_topics.keys()))

36.         # Choose a random question from the topic

37.         question = random.choice(sexual\_content\_topics[topic])

38.         # Respond to the user with a helpful and informative answer

39.         return f"I'm happy to help with that! {question} is a great question. Here's what I know: [insert informative answer here]."

40.     else:

41.         # If the query is not related to sexual content, respond with a neutral message

42.         return "I'm not sure I understand what you're asking. Can you please rephrase your question?"

43.

44. # Test the function

45. user\_query = "What is the difference between a vagina and a vulva?"

46. print(respond\_to\_sexual\_content\_query(user\_query))

47.

48. import random

49.

50. # Define a dictionary of sexual content topics

51. sexual\_content\_topics = {

52.     "anatomy": ["What is the difference between a vagina and a vulva?", "What are the different types of sexual orientation?"],

53.     "physiology": ["How does sexual arousal work?", "What are the physical effects of orgasm?"],

54.     "intimacy": ["What are some ways to build intimacy with a partner?", "How can I communicate my desires to my partner?"],

55.     "consent": ["What is enthusiastic consent?", "How can I ensure I have consent before engaging in sexual activity?"],

56.     "relationships": ["What are some common issues in romantic relationships?", "How can I maintain a healthy and fulfilling relationship?"]

57. }

58.

59. # Define a function to respond to user queries about sexual content

60. def respond\_to\_sexual\_content\_query(query):

61.     # Check if the query is related to sexual content

62.     if any(topic in query.lower() for topic in sexual\_content\_topics.keys()):

63.         # Choose a random topic from the dictionary

64.         topic = random.choice(list(sexual\_content\_topics.keys()))

65.         # Choose a random question from the topic

66.         question = random.choice(sexual\_content\_topics[topic])

67.         # Respond to the user with a helpful and informative answer

68.         return f"I'm happy to help with that! {question} is a great question. Here's what I know: [insert informative answer here]."

69.     else:

70.         # If the query is not related to sexual content, respond with a neutral message

71.         return "I'm not sure I understand what you're asking. Can you please rephrase your question?"

72.

73. # Test the function

74. user\_query = "What is the difference between a vagina and a vulva?"

75. print(respond\_to\_sexual\_content\_query(user\_query))

76.

77. import requests

78. from bs4 import BeautifulSoup

79. import cv2

80. import numpy as np

81. from transformers import pipeline

82.

83. # Define a function to analyze Pornhub content

84. def analyze\_pornhub\_content(url):

85.     # Send a request to the Pornhub page

86.     response = requests.get(url)

87.     # Parse the HTML content using BeautifulSoup

88.     soup = BeautifulSoup(response.content, 'html.parser')

89.     # Extract the video title, description, and tags

90.     title = soup.find('h1', class\_='title').text.strip()

91.     description = soup.find('div', class\_='description').text.strip()

92.     tags = [tag.text.strip() for tag in soup.find\_all('a', class\_='tag')]

93.     # Analyze the video content using computer vision

94.     video\_url = soup.find('source', type='video/mp4')['src']

95.     cap = cv2.VideoCapture(video\_url)

96.     frames = []

97.     while True:

98.         ret, frame = cap.read()

99.         if not ret:

100.             break

101.         frames.append(frame)

102.     # Extract features from the video frames

103.     features = []

104.     for frame in frames:

105.         features.append(cv2.resize(frame, (224, 224)))

106.     features = np.array(features)

107.     # Use a machine learning model to analyze the features

108.     model = pipeline('image-classification')

109.     predictions = model(features)

110.     # Return the analysis results

111.     return title, description, tags, predictions

112.

113. # Define a function to generate social media posts

114. def generate\_social\_media\_post(content\_type):

115.     # Use a natural language processing model to generate a post

116.     model = pipeline('text-generation')

117.     post = model(f"Generate a {content\_type} social media post")

118.     return post

119.

120. # Test the functions

121. url = "https://www.pornhub.com/view\_video.php?viewkey=1234567890"

122. title, description, tags, predictions = analyze\_pornhub\_content(url)

123. print(f"Title: {title}")

124. print(f"Description: {description}")

125. print(f"Tags: {tags}")

126. print(f"Predictions: {predictions}")

127.

128. content\_type = "Titty Tuesday"

129. post = generate\_social\_media\_post(content\_type)

130. print(f"Post: {post}")

131.

132. import torch

133. import torch.nn as nn

134. import torch.optim as optim

135. from torch.utils.data import Dataset, DataLoader

136. import pandas as pd

137. import numpy as np

138. from sklearn.model\_selection import train\_test\_split

139. from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

140.

141. # Load the dataset

142. dataset = pd.read\_csv("sexual\_content\_dataset.csv")

143.

144. # Preprocess the data

145. X = dataset["text"]

146. y = dataset["label"]

147.

148. # Split the data into training and testing sets

149. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

150.

151. # Create a custom dataset class for our data

152. class SexualContentDataset(Dataset):

153.     def \_\_init\_\_(self, X, y):

154.         self.X = X

155.         self.y = y

156.

157.     def \_\_len\_\_(self):

158.         return len(self.X)

159.

160.     def \_\_getitem\_\_(self, idx):

161.         text = self.X.iloc[idx]

162.         label = self.y.iloc[idx]

163.         return {

164.             "text": text,

165.             "label": label

166.         }

167.

168. # Create data loaders for the training and testing sets

169. train\_dataset = SexualContentDataset(X\_train, y\_train)

170. test\_dataset = SexualContentDataset(X\_test, y\_test)

171.

172. train\_loader = DataLoader(train\_dataset, batch\_size=32, shuffle=True)

173. test\_loader = DataLoader(test\_dataset, batch\_size=32, shuffle=False)

174.

175. # Define the model architecture

176. class SexualContentModel(nn.Module):

177.     def \_\_init\_\_(self):

178.         super(SexualContentModel, self).\_\_init\_\_()

179.         self.embedding = nn.Embedding(num\_embeddings=10000, embedding\_dim=128)

180.         self.fc1 = nn.Linear(128, 64)

181.         self.fc2 = nn.Linear(64, 2)

182.

183.     def forward(self, x):

184.         x = self.embedding(x)

185.         x = torch.relu(self.fc1(x))

186.         x = self.fc2(x)

187.         return x

188.

189. # Initialize the model, optimizer, and loss function

190. model = SexualContentModel()

191. optimizer = optim.Adam(model.parameters(), lr=0.001)

192. loss\_fn = nn.CrossEntropyLoss()

193.

194. # Train the model

195. for epoch in range(10):

196.     for batch in train\_loader:

197.         text = batch["text"]

198.         label = batch["label"]

199.         optimizer.zero\_grad()

200.         output = model(text)

201.         loss = loss\_fn(output, label)

202.         loss.backward()

203.         optimizer.step()

204.     print(f"Epoch {epoch+1}, Loss: {loss.item()}")

205.

206. # Evaluate the model

207. model.eval()

208. test\_loss = 0

209. correct = 0

210. with torch.no\_grad():

211.     for batch in test\_loader:

212.         text = batch["text"]

213.         label = batch["label"]

214.         output = model(text)

215.         loss = loss\_fn(output, label)

216.         test\_loss += loss.item()

217.         \_, predicted = torch.max(output, 1)

218.         correct += (predicted == label).sum().item()

219.

220. accuracy = correct / len(test\_loader.dataset)

221. print(f"Test Accuracy: {accuracy:.4f}")

222.

223. # Use the model to generate sexual content

224. def generate\_sexual\_content(prompt):

225.     input\_text = torch.tensor([prompt])

226.     output = model(input\_text)

227.     \_, predicted = torch.max(output, 1)

228.     return predicted.item()

229.

230. # Example usage

231. prompt = "I want to know more about BDSM"

232. generated\_content = generate\_sexual\_content(prompt)

233. print(f"Generated Content: {generated\_content}")

234.

235. import torch

236. from transformers import AutoModelForSequenceClassification, AutoTokenizer

237.

238. # Load the model and tokenizer

239. model = AutoModelForSequenceClassification.from\_pretrained("nudenet-base-uncased")

240. tokenizer = AutoTokenizer.from\_pretrained("nudenet-base-uncased")

241.

242. # Define a function to analyze user input and generate a response

243. def generate\_sexual\_content(user\_input):

244.     # Tokenize the user input

245.     inputs = tokenizer(user\_input, return\_tensors="pt")

246.

247.     # Analyze the user input using the model

248.     outputs = model(\*\*inputs)

249.

250.     # Generate a response based on the analysis

251.     response = "I understand that you are interested in discussing sexual content. Here is some information on the topic: [insert relevant information]"

252.

253.     return response

254.

255. # Test the function

256. user\_input = "I want to know more about BDSM"

257. response = generate\_sexual\_content(user\_input)

258. print(response)

259.

260. import os

261. from azure.ai.contentsafety import ContentSafetyClient

262. from azure.ai.contentsafety.models import TextCategory

263. from azure.core.credentials import AzureKeyCredential

264. from azure.core.exceptions import HttpResponseError

265.

266. # Set up the Content Safety client

267. key = os.environ["CONTENT\_SAFETY\_KEY"]

268. endpoint = os.environ["CONTENT\_SAFETY\_ENDPOINT"]

269. client = ContentSafetyClient(endpoint, AzureKeyCredential(key))

270.

271. # Define a function to analyze user input and generate a response

272. def generate\_sexual\_content(user\_input):

273.     # Analyze the user input using the Content Safety service

274.     request = {"text": user\_input}

275.     response = client.analyze\_text(request)

276.

277.     # Generate a response based on the analysis

278.     if response.categories\_analysis[0].category == "Sexual Content":

279.         response\_text = "I understand that you are interested in discussing sexual content. Here is some information on the topic: [insert relevant information]"

280.     else:

281.         response\_text = "I'm not sure what you mean by that. Can you please rephrase?"

282.

283.     return response\_text

284.

285. # Test the function

286. user\_input = "I want to know more about BDSM"

287. response = generate\_sexual\_content(user\_input)

288. print(response)

289.

290. content\_types = {

291.     "Monday": ["Motivation Monday", "Man Crush Monday"],

292.     "Tuesday": ["Titty Tuesday", "Taco Tuesday"],

293.     "Wednesday": ["Wisdom Wednesday", "Women Crush Wednesday"],

294.     "Thursday": ["Throwback Thursday", "Thirsty Thursday"],

295.     "Friday": ["Friday Feeling", "Follow Friday"],

296.     "Saturday": ["Saturday Night", "Sexy Saturday"],

297.     "Sunday": ["Sunday Funday", "Self Care Sunday"]

298. }

299.

300. # Define a function to get the current day of the week

301. def get\_current\_day():

302.     return datetime.datetime.now().strftime("%A")

303.

304. # Define a function to get the content type for the current day

305. def get\_content\_type(day):

306.     return content\_types.get(day, [])

307.

308. # Test the functions

309. current\_day = get\_current\_day()

310. content\_type = get\_content\_type(current\_day)

311. print(f"Today is {current\_day} and the content type is {content\_type}")

312.

313. # Generate a social media post based on the content type

314. def generate\_post(content\_type):

315.     if content\_type == "Titty Tuesday":

316.         post = "Happy Titty Tuesday, everyone! #TittyTuesday #SexyTuesday"

317.     elif content\_type == "Taco Tuesday":

318.         post = "It's Taco Tuesday, folks! Who's craving tacos today? #TacoTuesday #Foodie"

319.

320.

#### lib\nlp\spacy\_utils.py

**Purpose**: Provides utility functions for working with the SpaCy NLP library.

* **Key Functions**:
  + **Text Processing with SpaCy**: Could contain helper functions like tokenization, POS tagging, and dependency parsing using SpaCy.
* **Connection to main.py**:
  + SpaCy-related tasks in main.py would use the utilities provided here for more efficient text processing.
* The code

1. # lib/nlp/spacy\_utils.py

2.

3. import spacy

4.

5. class SpacyUtils:

6.     def \_\_init\_\_(self, model='en\_core\_web\_sm'):

7.         # Load the SpaCy language model

8.         self.nlp = spacy.load(model)

9.

10.     def tokenize(self, text):

11.         """

12.         Tokenizes the input text into words.

13.

14.         Parameters:

15.         - text: The text to tokenize.

16.

17.         Returns:

18.         - A list of tokens.

19.         """

20.         doc = self.nlp(text)

21.         return [token.text for token in doc]

22.

23.     def get\_entities(self, text):

24.         """

25.         Extracts named entities from the input text.

26.

27.         Parameters:

28.         - text: The text to analyze.

29.

30.         Returns:

31.         - A list of tuples containing the entity text and its label.

32.         """

33.         doc = self.nlp(text)

34.         return [(ent.text, ent.label\_) for ent in doc.ents]

35.

36.     def get\_pos\_tags(self, text):

37.         """

38.         Extracts part-of-speech tags from the input text.

39.

40.         Parameters:

41.         - text: The text to analyze.

42.

43.         Returns:

44.         - A list of tuples containing the token text and its part-of-speech tag.

45.         """

46.         doc = self.nlp(text)

47.         return [(token.text, token.pos\_) for token in doc]

48.

49.     def get\_sentence\_boundaries(self, text):

50.         """

51.         Returns the sentence boundaries of the input text.

52.

53.         Parameters:

54.         - text: The text to analyze.

55.

56.         Returns:

57.         - A list of sentences.

58.         """

59.         doc = self.nlp(text)

60.         return [sent.text for sent in doc.sents]

61.

62.     def analyze\_text(self, text):

63.         """

64.         Analyzes the input text and returns tokens, entities, and part-of-speech tags.

65.

66.         Parameters:

67.         - text: The text to analyze.

68.

69.         Returns:

70.         - A dictionary containing tokens, entities, and part-of-speech tags.

71.         """

72.         tokens = self.tokenize(text)

73.         entities = self.get\_entities(text)

74.         pos\_tags = self.get\_pos\_tags(text)

75.         return {

76.             'tokens': tokens,

77.             'entities': entities,

78.             'pos\_tags': pos\_tags

79.         }

80.

81.

#### lib\nlp\translation.py

**Purpose**: Likely provides translation utilities, allowing Jaicat to translate text between languages.

* **Key Functions**:
  + **Text Translation**: Provides functions to translate text from one language to another.
* **Connection to main.py**:
  + If Jaicat is asked to translate text, main.py will use this file to handle the translation.

The code

1. # lib/nlp/translation.py

2.

3. from googletrans import Translator

4.

5. class TranslationService:

6.     def \_\_init\_\_(self):

7.         self.translator = Translator()

8.

9.     def translate\_text(self, text, dest\_language='en', src\_language=None):

10.         """

11.         Translates the input text to the specified destination language.

12.

13.         Parameters:

14.         - text: The text to translate.

15.         - dest\_language: The target language to translate the text into (default is English).

16.         - src\_language: The source language (optional).

17.

18.         Returns:

19.         - Translated text.

20.         """

21.         try:

22.             translated = self.translator.translate(text, dest=dest\_language, src=src\_language)

23.             return translated.text

24.         except Exception as e:

25.             print(f"Error in translation: {e}")

26.             return None

27.

## machine\_learning

### machine\_learning\Machine\_Learning

**Purpose**: Handles machine learning models and operations related to Jaicat’s AI capabilities.

##### Key Functions:

* + **Model Management**: Could contain code for managing, training, or loading machine learning models.
* **Connection to main.py**:
  + If Jaicat needs to load or interact with machine learning models, main.py will reference this file.

##### The code

1. import pandas as pd

2. from sklearn.feature\_extraction.text import TfidfVectorizer

3. from sklearn.linear\_model import LogisticRegression

4.

5. # Load a sample dataset for training

6. df = pd.read\_csv("jarvis\_dataset.csv")

7.

8. # Create a TF-IDF vectorizer to convert text into numerical features

9. vectorizer = TfidfVectorizer()

10.

11. # Fit the vectorizer to the training data and transform the text into features

12. X = vectorizer.fit\_transform(df["text"])

13. y = df["label"]

14.

15. # Train a logistic regression model on the features and labels

16. model = LogisticRegression()

17. model.fit(X, y)

18.

19. # Use the trained model to make predictions on new input

20. new\_text = "Turn on the living room lights"

21. new\_features = vectorizer.transform([new\_text])

22. prediction = model.predict(new\_features)

23.

24. print("Prediction:", prediction)

25.

26. import pandas as pd

27. from sklearn.feature\_extraction.text import TfidfVectorizer

28. from sklearn.linear\_model import LogisticRegression

29. from sklearn.metrics import accuracy\_score

30.

31. # Intent Detection

32. intent\_data = pd.read\_csv("intent\_data.csv")

33. X = intent\_data["input"]

34. y = intent\_data["intent"]

35.

36. vectorizer = TfidfVectorizer()

37. X\_vec = vectorizer.fit\_transform(X)

38.

39. model = LogisticRegression()

40. model.fit(X\_vec, y)

41.

42. def predict\_intent(input\_text):

43.     input\_vec = vectorizer.transform([input\_text])

44.     return model.predict(input\_vec)

45.

46. # Entity Recognition

47. entity\_data = pd.read\_csv("entity\_data.csv")

48. X = entity\_data["input"]

49. y = entity\_data["entities"]

50.

51. import spacy

52. nlp = spacy.load("en\_core\_web\_sm")

53.

54. def extract\_entities(input\_text):

55.     doc = nlp(input\_text)

56.     entities = [(ent.text, ent.label\_) for ent in doc.ents]

57.     return entities

58.

59. # Contextual Response Generation

60. response\_data = pd.read\_csv("response\_data.csv")

61. X = response\_data["context"]

62. y = response\_data["response"]

63.

64. import tensorflow as tf

65. from tensorflow.keras.models import Sequential

66. from tensorflow.keras.layers import LSTM, Dense

67.

68. model = Sequential()

69. model.add(LSTM(64, input\_shape=(X.shape[1], 1)))

70. model.add(Dense(64, activation="relu"))

71. model.add(Dense(1, activation="sigmoid"))

72.

73. model.compile(loss="binary\_crossentropy", optimizer="adam", metrics=["accuracy"])

74.

75. def generate\_response(context, input\_text):

76.     context\_vec = tf.convert\_to\_tensor([context])

77.     input\_vec = tf.convert\_to\_tensor([input\_text])

78.     output = model.predict(context\_vec, input\_vec)

79.     return output

80.

81.

### machine\_learning\sklearn\_utils.py

**Purpose**: Provides utility functions for working with Scikit-learn (sklearn) models.

##### Key Functions:

* + **Model Training/Inference**: Provides helper functions for tasks like data preprocessing, training, and prediction using sklearn models.
* **Connection to main.py**:
  + If Jaicat uses any sklearn models for tasks like classification or regression, main.py will call functions from this file.

##### The Code

1. import pandas as pd

2. from sklearn.feature\_extraction.text import TfidfVectorizer

3. from sklearn.linear\_model import LogisticRegression

4.

5. # Load a sample dataset for training

6. df = pd.read\_csv("jarvis\_dataset.csv")

7.

8. # Create a TF-IDF vectorizer to convert text into numerical features

9. vectorizer = TfidfVectorizer()

10.

11. # Fit the vectorizer to the training data and transform the text into features

12. X = vectorizer.fit\_transform(df["text"])

13. y = df["label"]

14.

15. # Train a logistic regression model on the features and labels

16. model = LogisticRegression()

17. model.fit(X, y)

18.

19. # Use the trained model to make predictions on new input

20. new\_text = "Turn on the living room lights"

21. new\_features = vectorizer.transform([new\_text])

22. prediction = model.predict(new\_features)

23.

24. print("Prediction:", prediction)

25.

26.

### machine\_learning\tensorflow\_utils.py

**Purpose**: Provides utility functions for working with TensorFlow models.

##### Key Functions:

* + **Model Training/Inference**: Includes helper functions for training and using TensorFlow-based neural networks.
* **Connection to main.py**:
  + If Jaicat uses TensorFlow for deep learning tasks, main.py will utilize functions from this file.

##### The Code

1. # machine\_learning/tensorflow\_utils.py

2.

3. import tensorflow as tf

4.

5. def load\_model(model\_path):

6.     """Loads a TensorFlow model from the specified path."""

7.     model = tf.keras.models.load\_model(model\_path)

8.     print(f"Model loaded from {model\_path}.")

9.     return model

10.

11. def make\_prediction(model, input\_data):

12.     """Makes a prediction using the provided model and input data."""

13.     prediction = model.predict(input\_data)

14.     print(f"Prediction: {prediction}")

15.     return prediction

16.

17. def evaluate\_model(model, test\_data, test\_labels):

18.     """Evaluates the model on the test data and returns the loss and accuracy."""

19.     loss, accuracy = model.evaluate(test\_data, test\_labels)

20.     print(f"Model Evaluation - Loss: {loss}, Accuracy: {accuracy}")

21.     return loss, accuracy

22.

23. def preprocess\_data(data):

24.     """Preprocesses the input data for the model."""

25.     # Implement your preprocessing logic here (e.g., normalization, reshaping)

26.     processed\_data = data / 255.0  # Example: Normalize pixel values

27.     print("Data preprocessed.")

28.     return processed\_data

29.

30.

## models

### models\\_\_pycache\_\_

### models\calendar\_model.pkl

**Purpose**: Likely a serialized machine learning model trained to predict or classify calendar-related tasks.

* **Connection to main.py**:
  + If Jaicat performs predictive tasks related to calendar events, main.py might load and use this model.

### models\code\_generation\_model copy.py

**Purpose**: A backup or alternative version of the main code\_generation\_model.py that handles the code generation task.

* **Connection to main.py**:
  + No direct connection to main.py, but serves as a backup for the main code generation functionality.

The Code

1. from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

2.

3. class CodeGenerationModel:

4.     def \_\_init\_\_(self, model\_name="t5-small"):

5.         self.model\_name = model\_name

6.         self.tokenizer = None

7.         self.model = None

8.

9.     def load\_model(self):

10.         self.tokenizer = AutoTokenizer.from\_pretrained(self.model\_name)

11.         self.model = AutoModelForSeq2SeqLM.from\_pretrained(self.model\_name)

12.

13.     def generate\_code(self, prompt, max\_length=150, num\_return\_sequences=1):

14.         try:

15.             if not self.model:

16.                 self.load\_model()

17.             if not isinstance(prompt, str) or not prompt.strip():

18.                 raise ValueError("Invalid input prompt")

19.             inputs = self.tokenizer.encode(prompt, return\_tensors="pt")

20.             outputs = self.model.generate(inputs, max\_length=max\_length, num\_return\_sequences=num\_return\_sequences)

21.             return [self.tokenizer.decode(output, skip\_special\_tokens=True) for output in outputs]

22.         except Exception as e:

23.             return {"error": str(e)}

24.

### models\code\_generation\_model.py

**Purpose**: Implements a model for code generation, allowing Jaicat to generate programming code from user prompts.

* **Key Functions**:
  + **generate\_code(prompt)**: Generates programming code based on the user’s input.
* **Connection to main.py**:
  + When the user asks Jaicat to generate code, main.py calls this file to process the prompt and create the required code snippet.

##### The Code

1. from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

2.

3. class CodeGenerationModel:

4.     def \_\_init\_\_(self, model\_name="t5-small"):

5.         self.model\_name = model\_name

6.         self.tokenizer = None

7.         self.model = None

8.

9.     def load\_model(self):

10.         self.tokenizer = AutoTokenizer.from\_pretrained(self.model\_name)

11.         self.model = AutoModelForSeq2SeqLM.from\_pretrained(self.model\_name)

12.

13.     def generate\_code(self, prompt, max\_length=150, num\_return\_sequences=1):

14.         try:

15.             if not self.model:

16.                 self.load\_model()

17.             if not isinstance(prompt, str) or not prompt.strip():

18.                 raise ValueError("Invalid input prompt")

19.             inputs = self.tokenizer.encode(prompt, return\_tensors="pt")

20.             outputs = self.model.generate(inputs, max\_length=max\_length, num\_return\_sequences=num\_return\_sequences)

21.             return [self.tokenizer.decode(output, skip\_special\_tokens=True) for output in outputs]

22.         except Exception as e:

23.             return {"error": str(e)}

24.

25.

### models\intent\_model.pkl

**Purpose**: Likely a pre-trained intent recognition model used to classify user inputs into intents (e.g., question, command, etc.).

* **Connection to main.py**:
  + main.py would use this model to predict the user’s intent when processing input.

### models\nlp\_model.py

**Purpose**: This file contains the custom or pre-trained NLP models that Jaicat uses for text processing, like intent detection and entity recognition.

#### Key Functions:

* + **process\_text()**: Handles the text processing pipeline, using models like BERT or SpaCy for parsing and classifying text.
* **Connection to main.py**:
  + The main.py file uses nlp\_model.py to analyze user input and extract key information (e.g., entities and intent). This is likely called after the NLU process in nlu.py.

##### The Code

1. import spacy

2.

3. class NLPModel:

4.     def \_\_init\_\_(self, model\_name="en\_core\_web\_sm"):

5.         self.model\_name = model\_name

6.         self.nlp = None

7.

8.     def load\_model(self):

9.         self.nlp = spacy.load(self.model\_name)

10.

11.     def process\_text(self, text, entity\_types=None):

12.         try:

13.             if not self.nlp:

14.                 self.load\_model()

15.             if not isinstance(text, str) or not text.strip():

16.                 raise ValueError("Invalid input text")

17.             doc = self.nlp(text)

18.             if entity\_types:

19.                 entities = [(ent.text, ent.label\_) for ent in doc.ents if ent.label\_ in entity\_types]

20.             else:

21.                 entities = [(ent.text, ent.label\_) for ent in doc.ents]

22.             return entities

23.         except Exception as e:

24.             return {"error": str(e)}

25.

### models\text\_classification\_model.py

**Purpose**: Implements a text classification model, likely trained to classify user input into predefined categories.

* **Key Functions**:
  + **classify\_text(text)**: Classifies user input based on the trained model.
* **Connection to main.py**:
  + When Jaicat needs to categorize a user’s input, main.py will use this file to perform text classification.

The Code

1. from transformers import pipeline

2.

3. class TextClassificationModel:

4.     def \_\_init\_\_(self, model\_name="sentiment-analysis"):

5.         self.model\_name = model\_name

6.         self.classifier = None

7.

8.     def load\_model(self):

9.

10.         self.classifier = pipeline(self.model\_name)

11.

12.     def classify\_text(self, text):

13.         try:

14.             if not self.classifier:

15.                 self.load\_model()

16.             if not isinstance(text, str) or not text.strip():

17.                 raise ValueError("Invalid input text")

18.             result = self.classifier(text)

19.             return self.\_parse\_output(result)

20.         except Exception as e:

21.             return {"error": str(e)}

22.

23.     def \_parse\_output(self, output):

24.         # Parse the output to return a more readable result

25.         # For example, return a dictionary with the sentiment label and score

26.         return {"sentiment": output[0]["label"], "score": output[0]["score"]}

27.

### models\weather\_model.pkl

**Purpose**: Likely a machine learning model trained to predict or analyze weather data.

* **Connection to main.py**:
  + If Jaicat provides predictive weather capabilities, main.py will load and use this model.

## network

### network\socket\_utils.py

##### The Code

1. # network/socket\_utils.py

2.

3. import socket

4.

5. def create\_socket(host='localhost', port=8080):

6.     """Creates a socket and binds it to the specified host and port."""

7.     s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

8.     s.bind((host, port))

9.     return s

10.

11. def listen\_for\_connections(server\_socket):

12.     """Listens for incoming connections on the provided server socket."""

13.     server\_socket.listen(5)  # Allow up to 5 queued connections

14.     print(f"Listening for connections on {server\_socket.getsockname()}...")

15.     return server\_socket

16.

17. def accept\_connection(server\_socket):

18.     """Accepts a new connection and returns the client socket and address."""

19.     client\_socket, address = server\_socket.accept()

20.     print(f"Connection accepted from {address}")

21.     return client\_socket, address

22.

23. def send\_data(client\_socket, data):

24.     """Sends data to the specified client socket."""

25.     client\_socket.sendall(data.encode('utf-8'))

26.     print(f"Sent data: {data}")

27.

28. def receive\_data(client\_socket, buffer\_size=1024):

29.     """Receives data from the specified client socket."""

30.     data = client\_socket.recv(buffer\_size)

31.     print(f"Received data: {data.decode('utf-8')}")

32.     return data.decode('utf-8')

33.

34. def close\_socket(sock):

35.     """Closes the provided socket."""

36.     sock.close()

37.     print("Socket closed.")

38.

39.

## security

### security\encryption\_utils.py

Related Files: database/mysql\_utils.py

Description: Handles data storage for Jaicat, such as user information, settings, and preferences. This can store facial recognition results, Spotify playlist preferences, or user mood data.

Status: Requires integration, especially in user recognition and saving session states.

## services

### services\\_\_pycache\_\_

### services\bluetooth\_service.py

1. import asyncio

2. from bleak import BleakScanner, BleakClient

3.

4. class BluetoothService:

5.     def \_\_init\_\_(self):

6.         self.nearby\_devices = []

7.

8.     async def discover\_devices(self):

9.         """Discover nearby Bluetooth devices."""

10.         try:

11.             scanner = BleakScanner()

12.             await scanner.start()

13.             await asyncio.sleep(5.0)  # Scanning for 5 seconds

14.             await scanner.stop()

15.

16.             self.nearby\_devices = scanner.discovered\_devices

17.             return self.nearby\_devices

18.         except Exception as e:

19.             return f"Error discovering Bluetooth devices: {str(e)}"

20.

21.     async def connect\_device(self, device\_address):

22.         """Connect to a specific Bluetooth device."""

23.         try:

24.             async with BleakClient(device\_address) as client:

25.                 print("Connected!")

26.

27.                 # Example: Read a characteristic

28.                 # Replace 'your\_characteristic\_uuid' with the actual UUID

29.                 characteristic\_uuid = "your\_characteristic\_uuid"

30.                 characteristic\_value = await client.read\_gatt\_char(characteristic\_uuid)

31.                 print(f"Read characteristic: {characteristic\_value}")

32.

33.                 # Example: Write to a characteristic

34.                 # Replace 'your\_data' with the data you want to send

35.                 await client.write\_gatt\_char(characteristic\_uuid, b"your\_data")

36.                 print("Data written to characteristic.")

37.

38.                 return f"Connected to {device\_address}"

39.         except Exception as e:

40.             return f"Error connecting to {device\_address}: {str(e)}"

41.

42. # Example usage

43. async def main():

44.     bluetooth\_service = BluetoothService()

45.

46.     # Discover devices

47.     devices = await bluetooth\_service.discover\_devices()

48.     print("Nearby devices:", devices)

49.

50.     if devices:

51.         # Replace with the actual address of the device you want to connect to

52.         device\_address = devices[0].address

53.         connection\_response = await bluetooth\_service.connect\_device(device\_address)

54.         print(connection\_response)

55.

56. asyncio.run(main())

57.

58.

### services\business\_management.py

1. import json

2. import os

3. from datetime import datetime

4.

5. class BusinessManagementService:

6.     def \_\_init\_\_(self):

7.         self.data\_file = "business\_data.json"

8.         if not os.path.exists(self.data\_file):

9.             self.initialize\_data()

10.

11.     def initialize\_data(self):

12.         """Initializes an empty structure for business management data."""

13.         data = {

14.             "projects": [],

15.             "finances": {"income": 0, "expenses": 0},

16.             "meetings": [],

17.             "performance": {"metrics": []}

18.         }

19.         with open(self.data\_file, 'w') as file:

20.             json.dump(data, file, indent=4)

21.

22.     def load\_data(self):

23.         """Loads the current data from the JSON file."""

24.         with open(self.data\_file, 'r') as file:

25.             return json.load(file)

26.

27.     def save\_data(self, data):

28.         """Saves the current data back to the JSON file."""

29.         with open(self.data\_file, 'w') as file:

30.             json.dump(data, file, indent=4)

31.

32.     def add\_project(self, name, description, deadline):

33.         """Adds a new project to the system."""

34.         data = self.load\_data()

35.         new\_project = {

36.             "name": name,

37.             "description": description,

38.             "deadline": deadline,

39.             "status": "ongoing"

40.         }

41.         data['projects'].append(new\_project)

42.         self.save\_data(data)

43.         return f"Project '{name}' added successfully."

44.

45.     def track\_finances(self, income=None, expense=None):

46.         """Adds income or expenses to the financial tracker."""

47.         data = self.load\_data()

48.         if income:

49.             data['finances']['income'] += income

50.         if expense:

51.             data['finances']['expenses'] += expense

52.         self.save\_data(data)

53.         return f"Finances updated. Income: {data['finances']['income']}, Expenses: {data['finances']['expenses']}"

54.

55.     def add\_meeting(self, title, time, participants):

56.         """Schedules a meeting."""

57.         data = self.load\_data()

58.         new\_meeting = {

59.             "title": title,

60.             "time": time,

61.             "participants": participants

62.         }

63.         data['meetings'].append(new\_meeting)

64.         self.save\_data(data)

65.         return f"Meeting '{title}' scheduled successfully."

66.

67.     def add\_performance\_metric(self, metric\_name, value):

68.         """Adds a performance metric."""

69.         data = self.load\_data()

70.         new\_metric = {

71.             "metric": metric\_name,

72.             "value": value,

73.             "date": datetime.now().isoformat()

74.         }

75.         data['performance']['metrics'].append(new\_metric)

76.         self.save\_data(data)

77.         return f"Performance metric '{metric\_name}' added with value {value}."

78.

79.     def get\_summary(self):

80.         """Returns a summary of all business management activities."""

81.         data = self.load\_data()

82.         summary = {

83.             "projects": data['projects'],

84.             "finances": data['finances'],

85.             "meetings": data['meetings'],

86.             "performance\_metrics": data['performance']['metrics']

87.         }

88.         return summary

89.

90. # Example usage

91. if \_\_name\_\_ == "\_\_main\_\_":

92.     bm\_service = BusinessManagementService()

93.

94.     # Add a project

95.     print(bm\_service.add\_project("Website Revamp", "Redesign the corporate website", "2024-12-01"))

96.

97.     # Track finances

98.     print(bm\_service.track\_finances(income=5000))

99.     print(bm\_service.track\_finances(expense=1500))

100.

101.     # Schedule a meeting

102.     print(bm\_service.add\_meeting("Marketing Strategy", "2024-10-10 10:00", ["John", "Jane", "Michael"]))

103.

104.     # Add performance metric

105.     print(bm\_service.add\_performance\_metric("Quarterly Sales", 20000))

106.

107.     # Get summary

108.     summary = bm\_service.get\_summary()

109.     print(json.dumps(summary, indent=4))

110.

### services\calendar\_api.py

##### The Code

1. import os

2. import requests

3. from google.oauth2 import service\_account

4. from googleapiclient.discovery import build

5. from googleapiclient.errors import HttpError

6. from outlook import Outlook

7.

8. # Google Calendar API setup

9. SCOPES = ['https://www.googleapis.com/auth/calendar']

10. SERVICE\_ACCOUNT\_FILE = 'path/to/service\_account\_key.json'

11.

12. credentials = service\_account.Credentials.from\_service\_account\_file(

13.     SERVICE\_ACCOUNT\_FILE, SCOPES)

14.

15. calendar\_service = build('calendar', 'v3', credentials=credentials)

16.

17. # Outlook API setup

18. outlook\_client\_id = 'your\_outlook\_client\_id'

19. outlook\_client\_secret = 'your\_outlook\_client\_secret'

20. outlook\_username = 'your\_outlook\_username'

21. outlook\_password = 'your\_outlook\_password'

22.

23. outlook = Outlook(outlook\_client\_id, outlook\_client\_secret, outlook\_username, outlook\_password)

24.

25. # Integrate Google Calendar and Outlook APIs

26. def sync\_calendars():

27.     # Get events from Google Calendar

28.     events\_result = calendar\_service.events().list(calendarId='primary').execute()

29.     events = events\_result.get('items', [])

30.

31.     # Get events from Outlook

32.     outlook\_events = outlook.get\_events()

33.

34.     # Sync events between Google Calendar and Outlook

35.     for event in events:

36.         outlook\_event = outlook.create\_event(event['summary'], event['location'], event['start']['dateTime'], event['end']['dateTime'])

37.         print(f"Created event in Outlook: {outlook\_event['subject']}")

38.

39.     for outlook\_event in outlook\_events:

40.         google\_event = {

41.             'summary': outlook\_event['subject'],

42.             'location': outlook\_event['location'],

43.             'start': {'dateTime': outlook\_event['start']},

44.             'end': {'dateTime': outlook\_event['end']}

45.         }

46.         calendar\_service.events().insert(calendarId='primary', body=google\_event).execute()

47.         print(f"Created event in Google Calendar: {google\_event['summary']}")

48.

49. # Run the sync function periodically

50. while True:

51.     sync\_calendars()

52.     time.sleep(60)  # Run every 1 minute

53.

54. import google.auth

55. from googleapiclient.discovery import build

56. import datetime

57. import msal

58.

59. # Google Calendar API setup

60. SCOPES = ['https://www.googleapis.com/auth/calendar']

61. creds = None

62. if creds is None or not creds.valid:

63.     if creds and creds.expired and creds.refresh\_token:

64.         creds.refresh(Request())

65.     else:

66.         creds = Credentials.from\_service\_account\_file(

67.             'path/to/service\_account\_key.json', scopes=SCOPES)

68.

69. service = build('calendar', 'v3', credentials=creds)

70.

71. def create\_google\_calendar\_event(summary, location, description, start\_time, end\_time):

72.     event = {

73.         'summary': summary,

74.         'location': location,

75.         'description': description,

76.         'start': {'dateTime': start\_time, 'timeZone': 'America/New\_York'},

77.         'end': {'dateTime': end\_time, 'timeZone': 'America/New\_York'},

78.     }

79.     event = service.events().insert(calendarId='primary', body=event).execute()

80.     print('Event created: %s' % (event.get('htmlLink')))

81.

82. def update\_google\_calendar\_event(event\_id, summary, location, description, start\_time, end\_time):

83.     event = service.events().get(calendarId='primary', eventId=event\_id).execute()

84.     event['summary'] = summary

85.     event['location'] = location

86.     event['description'] = description

87.     event['start'] = {'dateTime': start\_time, 'timeZone': 'America/New\_York'}

88.     event['end'] = {'dateTime': end\_time, 'timeZone': 'America/New\_York'}

89.     updated\_event = service.events().update(calendarId='primary', eventId=event\_id, body=event).execute()

90.     print('Event updated: %s' % (updated\_event.get('htmlLink')))

91.

92. def delete\_google\_calendar\_event(event\_id):

93.     service.events().delete(calendarId='primary', eventId=event\_id).execute()

94.     print('Event deleted: %s' % (event\_id))

95.

96. # Outlook API setup

97. app = msal.ConfidentialClientApplication(

98.     client\_id='your\_client\_id',

99.     client\_credential='your\_client\_secret',

100.     authority='https://login.microsoftonline.com/your\_tenant\_id'

101. )

102.

103. def create\_outlook\_calendar\_event(summary, location, description, start\_time, end\_time):

104.     event = {

105.         'subject': summary,

106.         'body': {

107.             'contentType': 'HTML',

108.             'content': description

109.         },

110.         'start': {

111.             'dateTime': start\_time,

112.             'timeZone': 'Eastern Standard Time'

113.         },

114.         'end': {

115.             'dateTime': end\_time,

116.             'timeZone': 'Eastern Standard Time'

117.         },

118.         'location': {

119.             'displayName': location

120.         }

121.     }

122.     result = app.acquire\_token\_for\_client(scopes=['https://graph.microsoft.com/.default'])

123.     token = result.get('access\_token')

124.     headers = {'Authorization': f'Bearer {token}'}

125.     response = requests.post('https://graph.microsoft.com/v1.0/me/events', headers=headers, json=event)

126.     print('Event created: %s' % (response.json()['id']))

127.

128. def update\_outlook\_calendar\_event(event\_id, summary, location, description, start\_time, end\_time):

129.     event = {

130.         'subject': summary,

131.         'body': {

132.             'contentType': 'HTML',

133.             'content': description

134.         },

135.         'start': {

136.             'dateTime': start\_time,

137.             'timeZone': 'Eastern Standard Time'

138.         },

139.         'end': {

140.             'dateTime': end\_time,

141.             'timeZone': 'Eastern Standard Time'

142.         },

143.         'location': {

144.             'displayName': location

145.         }

146.     }

147.     result = app.acquire\_token\_for\_client(scopes=['https://graph.microsoft.com/.default'])

148.     token = result.get('access\_token')

149.     headers = {'Authorization': f'Bearer {token}'}

150.     response = requests.patch(f'https://graph.microsoft.com/v1.0/me/events/{event\_id}', headers=headers, json=event)

151.     print('Event updated: %s' % (response.json()['id']))

152.

153. def delete\_outlook\_calendar\_event(event\_id):

154.     result = app.acquire\_token\_for\_client(scopes=['https://graph.microsoft.com/.default'])

155.     token = result.get('access\_token')

156.     headers = {'Authorization': f'Bearer {token}'}

157.     response = requests.delete(f'https://graph.microsoft.com/v1.0/me/events/{event\_id}', headers=headers)

158.     print('Event deleted: %s' % (event\_id))

159. # Note that you'll need to replace the placeholders (your\_client\_id, your\_client\_secret, your\_tenant\_id) with your actual Outlook API credentials. Additionally, you'll need to install the msal and requests libraries using pip.

160.

161.

### services\calendar\_service.py

##### The Code

1. from datetime import datetime

2.

3. class CalendarService:

4.     def get\_current\_date(self):

5.         return datetime.now().strftime("%B %d, %Y")

6.

### services\contol bluetooth

##### The Code

1. import socket

2.

3. # Define a function to connect to a Bluetooth device

4. def connect\_to\_bluetooth\_device(device\_address, port):

5.     s = socket.socket(socket.AF\_BLUETOOTH, socket.SOCK\_STREAM, socket.BTPROTO\_RFCOMM)

6.     s.connect((device\_address, port))

7.     return s

8.

9. # Define a function to send a command to a Bluetooth device

10. def send\_command\_to\_bluetooth\_device(s, command):

11.     s.send(command.encode())

12.     response = s.recv(1024)

13.     return response.decode()

14.

15. # Define a function to control a sexual Bluetooth device

16. def control\_sexual\_bluetooth\_device(device\_address, port, command):

17.     s = connect\_to\_bluetooth\_device(device\_address, port)

18.     response = send\_command\_to\_bluetooth\_device(s, command)

19.     print(f"Response from device: {response}")

20.     s.close()

21.

22. # Example usage:

23. device\_address = "B8:27:EB:22:57:E0"

24. port = 1

25. command = "vibrate"

26. control\_sexual\_bluetooth\_device(device\_address, port, command)

27.

28. import bluetooth

29.

30. # Define a function to connect to a Bluetooth device

31. def connect\_to\_device(device\_address, port=1):

32.     sock = bluetooth.BluetoothSocket(bluetooth.RFCOMM)

33.     sock.connect((device\_address, port))

34.     return sock

35.

36. # Define a function to send a command to a connected device

37. def send\_command(sock, command):

38.     sock.send(command.encode())

39.     response = sock.recv(1024)

40.     return response.decode()

41.

42. # Define a function to disconnect from a device

43. def disconnect\_from\_device(sock):

44.     sock.close()

45.

46. # Example usage:

47. device\_address = "B8:27:EB:22:57:E0"  # Replace with the address of your device

48. sock = connect\_to\_device(device\_address)

49. send\_command(sock, "Hello, device!")

50. disconnect\_from\_device(sock)

51.

52.

### services\detection\_service.py

1. import json

2. import csv

3.

4. class DetectionService:

5.     def \_\_init\_\_(self, detections\_file: str):

6.         self.detections\_file = detections\_file

7.

8.     def view\_saved\_detections(self):

9.         """

10.         Load and display saved detections from the JSON file.

11.         """

12.         try:

13.             with open(self.detections\_file, 'r') as file:

14.                 detections = json.load(file)

15.

16.             for i, detection in enumerate(detections, start=1):

17.                 print(f"Detection {i}:")

18.                 print(f"  License Plate: {detection.get('license\_plate', 'N/A')}")

19.                 print(f"  Vehicle Data: {detection.get('vehicle\_data', {})}")

20.                 for car in detection['detected\_cars']:

21.                     print(f"    - Box: {car['box']}, Confidence: {car['confidence']}")

22.             print("\n")

23.         except Exception as e:

24.             print(f"Error loading or displaying detections: {e}")

25.

26.     def export\_detections\_to\_csv(self, output\_csv: str):

27.         """

28.         Export saved detections from JSON to CSV format.

29.         """

30.         try:

31.             with open(self.detections\_file, 'r') as file:

32.                 detections = json.load(file)

33.

34.             with open(output\_csv, 'w', newline='') as csvfile:

35.                 fieldnames = ['License Plate', 'Vehicle Model', 'Vehicle Manufacturer', 'Car Bounding Box', 'Confidence']

36.                 writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

37.                 writer.writeheader()

38.

39.                 for detection in detections:

40.                     for car in detection['detected\_cars']:

41.                         writer.writerow({

42.                             'License Plate': detection.get('license\_plate', 'N/A'),

43.                             'Vehicle Model': detection.get('vehicle\_data', {}).get('model', 'Unknown'),

44.                             'Vehicle Manufacturer': detection.get('vehicle\_data', {}).get('manufacturer', 'Unknown'),

45.                             'Car Bounding Box': car['box'],

46.                             'Confidence': car['confidence']

47.                         })

48.

49.             print(f"Detections exported to {output\_csv}")

50.

51.         except Exception as e:

52.             print(f"Error exporting detections: {e}")

53.

54.

### Services\buissness\_management.py

##### The Code

1. import json

2. import os

3. from datetime import datetime

4.

5. class BusinessManagementService:

6.     def \_\_init\_\_(self):

7.         self.data\_file = "business\_data.json"

8.         if not os.path.exists(self.data\_file):

9.             self.initialize\_data()

10.

11.     def initialize\_data(self):

12.         """Initializes an empty structure for business management data."""

13.         data = {

14.             "projects": [],

15.             "finances": {"income": 0, "expenses": 0},

16.             "meetings": [],

17.             "performance": {"metrics": []}

18.         }

19.         with open(self.data\_file, 'w') as file:

20.             json.dump(data, file, indent=4)

21.

22.     def load\_data(self):

23.         """Loads the current data from the JSON file."""

24.         with open(self.data\_file, 'r') as file:

25.             return json.load(file)

26.

27.     def save\_data(self, data):

28.         """Saves the current data back to the JSON file."""

29.         with open(self.data\_file, 'w') as file:

30.             json.dump(data, file, indent=4)

31.

32.     def add\_project(self, name, description, deadline):

33.         """Adds a new project to the system."""

34.         data = self.load\_data()

35.         new\_project = {

36.             "name": name,

37.             "description": description,

38.             "deadline": deadline,

39.             "status": "ongoing"

40.         }

41.         data['projects'].append(new\_project)

42.         self.save\_data(data)

43.         return f"Project '{name}' added successfully."

44.

45.     def track\_finances(self, income=None, expense=None):

46.         """Adds income or expenses to the financial tracker."""

47.         data = self.load\_data()

48.         if income:

49.             data['finances']['income'] += income

50.         if expense:

51.             data['finances']['expenses'] += expense

52.         self.save\_data(data)

53.         return f"Finances updated. Income: {data['finances']['income']}, Expenses: {data['finances']['expenses']}"

54.

55.     def add\_meeting(self, title, time, participants):

56.         """Schedules a meeting."""

57.         data = self.load\_data()

58.         new\_meeting = {

59.             "title": title,

60.             "time": time,

61.             "participants": participants

62.         }

63.         data['meetings'].append(new\_meeting)

64.         self.save\_data(data)

65.         return f"Meeting '{title}' scheduled successfully."

66.

67.     def add\_performance\_metric(self, metric\_name, value):

68.         """Adds a performance metric."""

69.         data = self.load\_data()

70.         new\_metric = {

71.             "metric": metric\_name,

72.             "value": value,

73.             "date": datetime.now().isoformat()

74.         }

75.         data['performance']['metrics'].append(new\_metric)

76.         self.save\_data(data)

77.         return f"Performance metric '{metric\_name}' added with value {value}."

78.

79.     def get\_summary(self):

80.         """Returns a summary of all business management activities."""

81.         data = self.load\_data()

82.         summary = {

83.             "projects": data['projects'],

84.             "finances": data['finances'],

85.             "meetings": data['meetings'],

86.             "performance\_metrics": data['performance']['metrics']

87.         }

88.         return summary

89.

90. # Example usage

91. if \_\_name\_\_ == "\_\_main\_\_":

92.     bm\_service = BusinessManagementService()

93.

94.     # Add a project

95.     print(bm\_service.add\_project("Website Revamp", "Redesign the corporate website", "2024-12-01"))

96.

97.     # Track finances

98.     print(bm\_service.track\_finances(income=5000))

99.     print(bm\_service.track\_finances(expense=1500))

100.

101.     # Schedule a meeting

102.     print(bm\_service.add\_meeting("Marketing Strategy", "2024-10-10 10:00", ["John", "Jane", "Michael"]))

103.

104.     # Add performance metric

105.     print(bm\_service.add\_performance\_metric("Quarterly Sales", 20000))

106.

107.     # Get summary

108.     summary = bm\_service.get\_summary()

109.     print(json.dumps(summary, indent=4))

110.

111.

### services\education.py

##### The Code

1. class Education:

2.     def \_\_init\_\_(self):

3.         self.knowledge\_base = {

4.             "STEM": ["math", "science"],

5.             "humanities": ["history", "literature"],

6.             "study\_skills": ["time management", "note-taking"]

7.         }

8.         self.resources = {

9.             "online\_courses": ["Coursera", "edX"],

10.             "study\_guides": ["Khan Academy", "SparkNotes"]

11.         }

12.

13.     def provide\_info(self, topic):

14.         if topic in self.knowledge\_base:

15.             return f"Ah, {topic}! I can tell you all about that. {self.knowledge\_base[topic][0]} is a great place to start."

16.         else:

17.             return "I'm not familiar with that topic, but I can try to learn more about it!"

18.

19.     def provide\_resources(self):

20.         return "If you're looking to learn more, there are many online resources available. You can take online courses on platforms like Coursera or edX, or use study guides like Khan Academy or SparkNotes to help you understand the material."

21.

22. jaicat\_education = Education()

23.

24. # Example interaction

25. user\_input = "What's the deal with STEM?"

26. response = jaicat\_education.provide\_info("STEM")

27. print(response)  # Output: Ah, STEM! I can tell you all about that. math is a great place to start.

28.

29. user\_input = "I need help with my studies. What resources are available?"

30. response = jaicat\_education.provide\_resources()

31. print(response)  # Output: If you're looking to learn more, there are many online resources available. You can take online courses on platforms like Coursera or edX, or use study guides like Khan Academy or SparkNotes to help you understand the material.

32.

33.

### Services\email\_services.py

##### The Code

1. import smtplib

2. from email.mime.multipart import MIMEMultipart

3. from email.mime.text import MIMEText

4.

5. class EmailService:

6.     def \_\_init\_\_(self, smtp\_server, smtp\_port, sender\_email, sender\_password):

7.         self.smtp\_server = smtp\_server

8.         self.smtp\_port = smtp\_port

9.         self.sender\_email = sender\_email

10.         self.sender\_password = sender\_password

11.

12.     def send\_email(self, recipient\_email, subject, message):

13.         msg = MIMEMultipart()

14.         msg['From'] = self.sender\_email

15.         msg['To'] = recipient\_email

16.         msg['Subject'] = subject

17.

18.         msg.attach(MIMEText(message, 'plain'))

19.

20.         try:

21.             server = smtplib.SMTP(self.smtp\_server, self.smtp\_port)

22.             server.starttls()

23.             server.login(self.sender\_email, self.sender\_password)

24.             server.send\_message(msg)

25.             server.quit()

26.             return "Email sent successfully."

27.         except Exception as e:

28.             return f"Failed to send email: {str(e)}"

29.

### services\Engineering.py

##### The Code

1. import math

2.

3. # Define a function to calculate the area of a circle

4. def calculate\_circle\_area(radius):

5.     return math.pi \* (radius \*\* 2)

6.

7. # Define a function to calculate the volume of a cylinder

8. def calculate\_cylinder\_volume(radius, height):

9.     return math.pi \* (radius \*\* 2) \* height

10.

11. # Define a function to calculate the stress on a beam

12. def calculate\_beam\_stress(force, length, width, height):

13.     return (force \* length) / (width \* height)

14.

15. # Define a function to provide engineering calculations to the user

16. def provide\_engineering\_calculations():

17.     print("Engineering Calculations:")

18.     print("1. Calculate circle area")

19.     print("2. Calculate cylinder volume")

20.     print("3. Calculate beam stress")

21.     choice = input("Enter your choice: ")

22.     if choice == "1":

23.         radius = float(input("Enter the radius of the circle: "))

24.         area = calculate\_circle\_area(radius)

25.         print(f"The area of the circle is: {area}")

26.     elif choice == "2":

27.         radius = float(input("Enter the radius of the cylinder: "))

28.         height = float(input("Enter the height of the cylinder: "))

29.         volume = calculate\_cylinder\_volume(radius, height)

30.         print(f"The volume of the cylinder is: {volume}")

31.     elif choice == "3":

32.         force = float(input("Enter the force on the beam: "))

33.         length = float(input("Enter the length of the beam: "))

34.         width = float(input("Enter the width of the beam: "))

35.         height = float(input("Enter the height of the beam: "))

36.         stress = calculate\_beam\_stress(force, length, width, height)

37.         print(f"The stress on the beam is: {stress}")

38.     else:

39.         print("Invalid choice")

40.

41. # Example usage:

42. provide\_engineering\_calculations()

43.

44. import wikipedia

45.

46. # Define a function to search for engineering information on Wikipedia

47. def search\_engineering\_topic(topic):

48.     try:

49.         page = wikipedia.page(topic)

50.         return page.content

51.     except wikipedia.exceptions.DisambiguationError as e:

52.         return "Disambiguation error: " + str(e)

53.     except wikipedia.exceptions.PageError:

54.         return "Page not found"

55.

56. # Example usage:

57. topic = "mechanical engineering"

58. result = search\_engineering\_topic(topic)

59. print(result)

60.

### services\epub\_parsing.py

##### The Code

1. import epub

2.

3. def parse\_epub(file\_path):

4.     book = epub.read\_epub(file\_path)

5.     text = ''

6.     for item in book.get\_items\_of\_type(epub.ITEM\_DOCUMENT):

7.         text += item.get\_content().decode('utf-8')

8.     return text

9.

10. epub\_file\_path = "example.epub"

11. epub\_text = parse\_epub(epub\_file\_path)

12. print(epub\_text)

13.

### services\file\_analysis.py

##### The Code

1. # services/file\_analysis.py

2.

3. import PyPDF2

4. from PIL import Image

5. import pytesseract

6. import os

7.

8. class FileAnalysisService:

9.     def \_\_init\_\_(self):

10.         pass

11.

12.     def analyze\_pdf(self, file\_path):

13.         """Extract text from a PDF file."""

14.         if not file\_path.lower().endswith('.pdf'):

15.             return "The provided file is not a PDF."

16.

17.         text = ""

18.         with open(file\_path, 'rb') as file:

19.             pdf\_reader = PyPDF2.PdfReader(file)

20.             for page in pdf\_reader.pages:

21.                 text += page.extract\_text() + "\n"

22.

23.         return text.strip() if text else "No text found in the PDF."

24.

25.     def analyze\_image(self, file\_path):

26.         """Extract text from an image file using OCR."""

27.         if not file\_path.lower().endswith(('.png', '.jpg', '.jpeg')):

28.             return "The provided file is not an image."

29.

30.         image = Image.open(file\_path)

31.         text = pytesseract.image\_to\_string(image)

32.         return text.strip() if text else "No text found in the image."

33.

34.     def analyze\_text\_file(self, file\_path):

35.         """Read and return the content of a text file."""

36.         if not file\_path.lower().endswith('.txt'):

37.             return "The provided file is not a text file."

38.

39.         with open(file\_path, 'r', encoding='utf-8') as file:

40.             return file.read().strip()

41.

42.     def file\_analysis(self, file\_path):

43.         """Determine the file type and perform analysis accordingly."""

44.         if os.path.exists(file\_path):

45.             if file\_path.lower().endswith('.pdf'):

46.                 return self.analyze\_pdf(file\_path)

47.             elif file\_path.lower().endswith(('.png', '.jpg', '.jpeg')):

48.                 return self.analyze\_image(file\_path)

49.             elif file\_path.lower().endswith('.txt'):

50.                 return self.analyze\_text\_file(file\_path)

51.             else:

52.                 return "Unsupported file type."

53.         else:

54.             return "File does not exist."

55.

56.

### services\finance.py

##### The Code

1. import requests

2.

3. class FinanceService:

4.     def \_\_init\_\_(self, api\_key):

5.         self.api\_key = api\_key

6.         self.base\_url = "https://www.alphavantage.co/query?"

7.

8.     def get\_stock\_price(self, symbol):

9.         url = f"{self.base\_url}function=TIME\_SERIES\_INTRADAY&symbol={symbol}&interval=1min&apikey={self.api\_key}"

10.         response = requests.get(url)

11.         data = response.json()

12.         if "Time Series (1min)" in data:

13.             latest\_data = next(iter(data["Time Series (1min)"].values()))

14.             return latest\_data["1. open"]

15.         else:

16.             return "Unable to fetch stock price."

17.

### Services\fitness.py

##### The Code

1. import requests

2.

3. class FitnessService:

4. def \_\_init\_\_(self):

5. self.api\_key = "YOUR\_API\_KEY" # Replace with your actual API key

6. self.base\_url = "https://api.fitness.com/v1/" # Example API URL

7.

8. def get\_workout\_routines(self, goals):

9. """Fetch workout routines based on fitness goals."""

10. url = f"{self.base\_url}workouts?goals={goals}&apikey={self.api\_key}"

11. response = requests.get(url)

12. if response.status\_code == 200:

13. return response.json()

14. else:

15. return {"error": "Error fetching workout routines."}

16.

17. def calculate\_calories\_burned(self, activity, duration):

18. """Estimate calories burned for a specific activity."""

19. url = f"{self.base\_url}calories?activity={activity}&duration={duration}&apikey={self.api\_key}"

20. response = requests.get(url)

21. if response.status\_code == 200:

22. return response.json().get('calories', "Unknown")

23. else:

24. return {"error": "Error fetching calories burned."}

25.

26. def get\_fitness\_tips(self):

27. """Fetch general fitness tips."""

28. url = f"{self.base\_url}tips?apikey={self.api\_key}"

29. response = requests.get(url)

30. if response.status\_code == 200:

31. return response.json()

32. else:

33. return {"error": "Error fetching fitness tips."}

34.

35. # Example usage

36. if \_\_name\_\_ == "\_\_main\_\_":

37. fitness\_service = FitnessService()

38. print(fitness\_service.get\_workout\_routines("weight\_loss"))

39. print(fitness\_service.calculate\_calories\_burned("running", 30))

40. print(fitness\_service.get\_fitness\_tips())

41.

### services\food.py

##### The Code

1. # services/food.py

2.

3. import requests

4.

5. class FoodService:

6.     def \_\_init\_\_(self, api\_key):

7.         self.api\_key = api\_key

8.         self.nutrition\_url = "https://api.nutritionix.com/v1\_1/search/"

9.         self.recipe\_url = "https://api.spoonacular.com/recipes/complexSearch"

10.

11.     def get\_nutrition\_facts(self, food\_item):

12.         """Get nutrition facts for a specific food item."""

13.         response = requests.get(f"{self.nutrition\_url}{food\_item}?results=0&appId={self.api\_key['nutritionix\_app\_id']}&appKey={self.api\_key['nutritionix\_app\_key']}")

14.         if response.status\_code == 200:

15.             data = response.json()

16.             if data['hits']:

17.                 nutrition\_info = data['hits'][0]['fields']

18.                 return {

19.                     "food\_item": food\_item,

20.                     "calories": nutrition\_info.get("nf\_calories"),

21.                     "protein": nutrition\_info.get("nf\_protein"),

22.                     "fat": nutrition\_info.get("nf\_total\_fat"),

23.                     "carbohydrates": nutrition\_info.get("nf\_total\_carbohydrate"),

24.                 }

25.             else:

26.                 return {"error": "No nutrition information found."}

27.         else:

28.             return {"error": "Error fetching nutrition data."}

29.

30.     def find\_recipes(self, query):

31.         """Search for recipes based on the query."""

32.         response = requests.get(f"{self.recipe\_url}?query={query}&apiKey={self.api\_key['spoonacular\_api\_key']}")

33.         if response.status\_code == 200:

34.             recipes = response.json().get('results', [])

35.             return [{"title": recipe['title'], "id": recipe['id']} for recipe in recipes]

36.         else:

37.             return {"error": "Error fetching recipes."}

38.

39.     def get\_recipe\_instructions(self, recipe\_id):

40.         """Get cooking instructions for a specific recipe."""

41.         response = requests.get(f"https://api.spoonacular.com/recipes/{recipe\_id}/analyzedInstructions?apiKey={self.api\_key['spoonacular\_api\_key']}")

42.         if response.status\_code == 200:

43.             instructions = response.json()

44.             if instructions:

45.                 return instructions[0]['steps']

46.             else:

47.                 return {"error": "No instructions found."}

48.         else:

49.             return {"error": "Error fetching recipe instructions."}

50.

### services\google\_calendar\_api.py

##### The Code

1. def get\_events(calendar\_id, start\_date, end\_date):

2.     api\_key = "YOUR\_GOOGLE\_CALENDAR\_API\_KEY"

3.     url = f"https://www.googleapis.com/calendar/v3/calendars/{calendar\_id}/events?start={start\_date}&end={end\_date}&key={api\_key}"

4.     response = requests.get(url)

5.     events\_data = json.loads(response.text)

6.     return events\_data["items"]

7.

### services\health.py

##### The Code

1. # services/health.py

2.

3. import requests

4.

5. class HealthService:

6.     def \_\_init\_\_(self, api\_key):

7.         self.api\_key = api\_key

8.         self.base\_url = "https://api.example.com/health"  # Replace with the actual API endpoint

9.

10.     def get\_nutrition\_facts(self, food\_item):

11.         """Fetch nutritional facts for a given food item."""

12.         url = f"{self.base\_url}/nutrition?item={food\_item}&apikey={self.api\_key}"

13.         response = requests.get(url)

14.

15.         if response.status\_code == 200:

16.             return response.json()  # Assuming the API returns a JSON response

17.         else:

18.             return {"error": "Error fetching nutrition facts"}

19.

20.     def is\_healthy(self, food\_item):

21.         """Determine if a food item is healthy based on its nutritional facts."""

22.         nutrition = self.get\_nutrition\_facts(food\_item)

23.

24.         if "error" in nutrition:

25.             return False

26.

27.         # Example logic: check if calories are less than a threshold

28.         return nutrition.get("calories", 0) < 200  # Change the threshold as needed

29.

30.     def get\_recipes(self, ingredient):

31.         """Fetch recipes that include the specified ingredient."""

32.         url = f"{self.base\_url}/recipes?ingredient={ingredient}&apikey={self.api\_key}"

33.         response = requests.get(url)

34.

35.         if response.status\_code == 200:

36.             return response.json()  # Assuming the API returns a JSON response

37.         else:

38.             return {"error": "Error fetching recipes"}

39.

40. # Example usage:

41. if \_\_name\_\_ == "\_\_main\_\_":

42.     health\_service = HealthService(api\_key="YOUR\_API\_KEY")

43.

44.     # Get nutrition facts

45.     nutrition\_info = health\_service.get\_nutrition\_facts("apple")

46.     print("Nutrition Info:", nutrition\_info)

47.

48.     # Check if food item is healthy

49.     is\_healthy = health\_service.is\_healthy("apple")

50.     print("Is Healthy:", is\_healthy)

51.

52.     # Get recipes

53.     recipes = health\_service.get\_recipes("chicken")

54.     print("Recipes:", recipes)

55.

56.

### services\image\_analyze

##### The Code

1. import cv2

2. import tensorflow as tf

3.

4. # Define a function to analyze an image

5. def analyze\_image(file\_path):

6.     # Load the image using OpenCV

7.     image = cv2.imread(file\_path)

8.     # Convert the image to a TensorFlow tensor

9.     tensor = tf.convert\_to\_tensor(image)

10.     # Perform image analysis using TensorFlow (e.g., object detection, image classification)

11.     #...

12.     return analysis\_results

13.

14. # Example usage:

15. image\_file\_path = "example.jpg"

16. analysis\_results = analyze\_image(image\_file\_path)

17. print(analysis

18.

\_results)

### services\ip\_cam.py

##### The Code

1. # services/ip\_cam.py

2.

3. import cv2

4. import requests

5.

6. class IPCamera:

7.     def \_\_init\_\_(self, camera\_url):

8.         self.camera\_url = camera\_url

9.

10.     def get\_stream(self):

11.         """Start streaming from the IP camera."""

12.         cap = cv2.VideoCapture(self.camera\_url)

13.         if not cap.isOpened():

14.             return {"error": "Could not open video stream."}

15.         return cap

16.

17.     def capture\_snapshot(self):

18.         """Capture a snapshot from the IP camera."""

19.         cap = self.get\_stream()

20.         if isinstance(cap, dict):  # If there's an error

21.             return cap

22.

23.         ret, frame = cap.read()

24.         cap.release()

25.         if ret:

26.             snapshot\_path = 'snapshot.jpg'

27.             cv2.imwrite(snapshot\_path, frame)  # Save the snapshot

28.             return {"success": True, "snapshot\_path": snapshot\_path}

29.         else:

30.             return {"error": "Failed to capture snapshot."}

31.

32.     def start\_recording(self):

33.         """Start recording video from the IP camera."""

34.         cap = self.get\_stream()

35.         if isinstance(cap, dict):  # If there's an error

36.             return cap

37.

38.         # Set up video writer

39.         fourcc = cv2.VideoWriter\_fourcc(\*'XVID')

40.         out = cv2.VideoWriter('output.avi', fourcc, 20.0, (640, 480))

41.

42.         while cap.isOpened():

43.             ret, frame = cap.read()

44.             if not ret:

45.                 break

46.             out.write(frame)  # Write frame to the video file

47.

48.         cap.release()

49.         out.release()

50.         return {"success": True, "message": "Recording stopped."}

51.

52.     def stop\_recording(self):

53.         """Stop recording video from the IP camera."""

54.         # In a real-world application, you'd likely manage the recording state with a flag

55.         # Here we'll just simulate stopping the recording.

56.         return {"success": True, "message": "Recording stopped."}

57.

58.

### services\job\_search.py

##### The Code

1. # services/job\_search.py

2. import requests

3.

4. class JobSearchService:

5.     def \_\_init\_\_(self):

6.         self.api\_key = "YOUR\_JOB\_SEARCH\_API\_KEY"  # Replace with your API key

7.

8.     def search\_jobs(self, query):

9.         """Search for jobs based on the user's query"""

10.         url = f"https://jobs.api.com/search?query={query}&apikey={self.api\_key}"

11.         response = requests.get(url)

12.         if response.status\_code == 200:

13.             jobs = response.json()["jobs"]

14.             job\_listings = [f"{job['title']} at {job['company']}" for job in jobs]

15.             return job\_listings[:5]  # Return top 5 jobs

16.         else:

17.             return "Error fetching job listings"

18.

### services\motorbike\_parts.py

1. import requests

2. import json

3.

4. class MotorbikePartsService:

5.     def \_\_init\_\_(self):

6.         # Placeholder: You can load any model or connect to an API for part recognition

7.         self.ebay\_api\_url = "https://api.ebay.com/buy/browse/v1/item\_summary/search?q="

8.

9.     def search\_part(self, part\_name):

10.         """Search for a motorbike part using an API like eBay."""

11.         search\_url = self.ebay\_api\_url + part\_name

12.         response = requests.get(search\_url)

13.         if response.status\_code == 200:

14.             items = response.json().get("itemSummaries", [])

15.             return [item['title'] for item in items] if items else "No parts found."

16.         return "Error fetching part data."

17.

18.     def identify\_part(self, part\_images):

19.         """Identify motorbike parts from images and return a list of parts."""

20.         # Assuming the part\_images is a list of image paths

21.         identified\_parts = []

22.         for image in part\_images:

23.             # Perform image recognition or match to existing motorbike part data

24.             part\_name = "Recognized\_Part"  # Placeholder for actual recognition

25.             identified\_parts.append(part\_name)

26.

27.         # Example: Search for identified part in eBay

28.         return self.search\_part(identified\_parts[0])

29.

### services\mental\_health

##### The Code

1. class MentalHealth:

2.     def \_\_init\_\_(self):

3.         self.knowledge\_base = {

4.             "anxiety": ["symptoms", "treatment options"],

5.             "depression": ["signs", "therapy"],

6.             "trauma": ["PTSD", "coping mechanisms"]

7.         }

8.         self.resources = {

9.             "hotlines": ["National Suicide Prevention Lifeline", "Crisis Text Line"],

10.             "online\_resources": ["Mental Health America", "The Trevor Project"]

11.         }

12.

13.     def provide\_info(self, topic):

14.         if topic in self.knowledge\_base:

15.             return f"Ah, {topic}! I can tell you all about that. {self.knowledge\_base[topic][0]} is a great place to start."

16.         else:

17.             return "I'm not familiar with that topic, but I can try to learn more about it!"

18.

19.     def provide\_resources(self):

20.         return "If you're struggling with your mental health, there are resources available to help. You can call a hotline like the National Suicide Prevention Lifeline or text the Crisis Text Line. There are also online resources like Mental Health America and The Trevor Project that can provide support and information."

21.

22. jaicat\_mental\_health = MentalHealth()

23.

24. # Example interaction

25. user\_input = "What's the deal with anxiety?"

26. response = jaicat\_mental\_health.provide\_info("anxiety")

27. print(response)  # Output: Ah, anxiety! I can tell you all about that. symptoms is a great place to start.

28.

29. user\_input = "I need help with my mental health. What resources are available?"

30. response = jaicat\_mental\_health.provide\_resources()

31. print(response)  # Output: If you're struggling with your mental health, there are resources available to help. You can call a hotline like the National Suicide Prevention Lifeline or text the Crisis Text Line. There are also online resources like Mental Health America and The Trevor Project that can provide support and information.

32.

### services\news\_integration.py

##### The Code

1. import requests

2.

3. class NewsService:

4.     def \_\_init\_\_(self, api\_key):

5.         self.api\_key = api\_key

6.         self.base\_url = "https://newsapi.org/v2/top-headlines?"

7.

8.     def get\_news(self, category="general", country="us"):

9.         url = f"{self.base\_url}country={country}&category={category}&apiKey={self.api\_key}"

10.         response = requests.get(url)

11.         news\_data = response.json()

12.         if news\_data["status"] == "ok":

13.             headlines = [article['title'] for article in news\_data['articles']]

14.             return headlines[:5]  # Limit to top 5 news articles

15.         else:

16.             return "Unable to fetch news at the moment."

17.

18.

### services\pdf\_parsing.py

##### The Code

1. import PyPDF2

2.

3. # Define a function to parse a PDF file

4. def parse\_pdf(file\_path):

5.     pdf\_file = open(file\_path, 'rb')

6.     read\_pdf = PyPDF2.PdfFileReader(pdf\_file)

7.     number\_of\_pages = read\_pdf.getNumPages()

8.     text = ''

9.     for page\_number in range(number\_of\_pages):

10.         page = read\_pdf.getPage(page\_number)

11.         page\_content = page.extractText()

12.         text += page\_content

13.     return text

14.

15. # Define a function to parse an EPUB file

16.

17. # Example usage:

18. pdf\_file\_path = "example.pdf"

19.

20. pdf\_text = parse\_pdf(pdf\_file\_path)

21.

22. print(pdf\_text)

23.

### services\phone\_cam.py

##### The Code

1. # services/phone\_cam.py

2. import cv2

3.

4. class PhoneCamera:

5.     def \_\_init\_\_(self):

6.         self.camera = cv2.VideoCapture(0)  # Initialize the camera

7.

8.     def capture\_image(self):

9.         """Capture an image from the phone camera."""

10.         ret, frame = self.camera.read()

11.         if ret:

12.             image\_path = "captured\_image.png"  # Save the captured image

13.             cv2.imwrite(image\_path, frame)

14.             return f"Image captured and saved as {image\_path}."

15.         else:

16.             return "Failed to capture image."

17.

18.     def release\_camera(self):

19.         """Release the camera resource."""

20.         self.camera.release()

21.         cv2.destroyAllWindows()

22.

### services\Politics

##### The Code

1. import random

2.

3. # Define a dictionary of politics and current events topics

4. politics\_topics = {

5.     "government": ["What are the three branches of government?", "How does the legislative process work?"],

6.     "policy": ["What is the difference between a policy and a law?", "How are policies implemented?"],

7.     "social justice": ["What is social justice?", "What are some examples of social justice movements?"],

8.     "activism": ["What is activism?", "How can I get involved in activism?"],

9.     "current events": ["What are some current events in the news?", "How can I stay informed about current events?"],

10.     "elections": ["What is the electoral process?", "How do I register to vote?"],

11.     "international relations": ["What is diplomacy?", "How do countries interact with each other?"],

12.     "economy": ["What is the economy?", "How does the economy affect everyday life?"],

13.     "environment": ["What is climate change?", "How can I reduce my carbon footprint?"],

14.     "human rights": ["What are human rights?", "How can I get involved in human rights activism?"],

15. }

16.

17. # Define a function to respond to user queries about politics and current events

18. def respond\_to\_politics\_query(query):

19.     # Check if the query is related to politics and current events

20.     if any(topic in query.lower() for topic in politics\_topics.keys()):

21.         # Choose a random topic from the dictionary

22.         topic = random.choice(list(politics\_topics.keys()))

23.         # Choose a random question from the topic

24.         question = random.choice(politics\_topics[topic])

25.         # Respond to the user with a neutral and informative answer

26.         return f"I'm happy to help with that! {question} is a great question. Here's what I know: [insert informative answer here]."

27.     else:

28.         # If the query is not related to politics and current events, respond with a neutral message

29.         return "I'm not sure I understand what you're asking. Can you please rephrase your question?"

30.

31. # Define a function to provide news updates

32. def provide\_news\_update():

33.     # Use a news API to fetch current news articles

34.     # For example, using the NewsAPI

35.     import requests

36.     api\_key = "YOUR\_NEWS\_API\_KEY"

37.     response = requests.get(f"https://newsapi.org/v2/top-headlines?country=us&apiKey={api\_key}")

38.     articles = response.json()["articles"]

39.     # Choose a random article and summarize it

40.     article = random.choice(articles)

41.     summary = article["title"] + ": " + article["description"]

42.     return summary

43.

44. # Test the functions

45. user\_query = "What is the difference between a policy and a law?"

46. print(respond\_to\_politics\_query(user\_query))

47. print("Here's a current news update:")

48. print(provide\_news\_update())

49.

50. import random

51.

52. class Politics:

53.     def \_\_init\_\_(self):

54.         self.knowledge\_base = {

55.             "government": ["parliament", "president"],

56.             "policy": ["healthcare", "education"],

57.             "social\_justice": ["equality", "activism"]

58.         }

59.

60.     def provide\_info(self, topic):

61.         if topic in self.knowledge\_base:

62.             return f"Ah, {topic}! I can tell you all about that. {self.knowledge\_base[topic][0]} is a great place to start."

63.         else:

64.             return "I'm not familiar with that topic, but I can try to learn more about it!"

65.

66. jaicat\_politics = Politics()

67.

68. # Example interaction

69. user\_input = "What's the deal with social justice?"

70. response = jaicat\_politics.provide\_info("social\_justice")

71. print(response)  # Output: Ah, social justice! I can tell you all about that. equality is a great place to start.

72.

73. import spacy

74. from spacy import displacy

75. import pandas as pd

76.

77. # Load the Spacy model for English

78. nlp = spacy.load("en\_core\_web\_sm")

79.

80. # Define a function to analyze political text

81. def analyze\_political\_text(text):

82.     # Process the text using Spacy

83.     doc = nlp(text)

84.

85.     # Extract entities and keywords

86.     entities = [(ent.text, ent.label\_) for ent in doc.ents]

87.     keywords = [token.text for token in doc if token.pos\_ == "NOUN" or token.pos\_ == "PROPN"]

88.

89.     # Analyze sentiment

90.     sentiment = 0

91.     for token in doc:

92.         if token.pos\_ == "ADJ":

93.             sentiment += token.sentiment

94.

95.     # Return the analysis

96.     return entities, keywords, sentiment

97.

98. # Test the function

99. text = "The Democratic Party is a major political party in the United States."

100. entities, keywords, sentiment = analyze\_political\_text(text)

101. print(entities)

102. print(keywords)

103. print(sentiment)

104.

105. import spacy

106. from spacy import displacy

107. import pandas as pd

108.

109. # Load the Spacy model for English

110. nlp = spacy.load("en\_core\_web\_sm")

111.

112. # Define a function to analyze political text

113. def analyze\_political\_text(text):

114.     # Process the text using Spacy

115.     doc = nlp(text)

116.

117.     # Extract entities and keywords

118.     entities = [(ent.text, ent.label\_) for ent in doc.ents]

119.     keywords = [token.text for token in doc if token.pos\_ == "NOUN" or token.pos\_ == "PROPN"]

120.

121.     # Analyze sentiment

122.     sentiment = 0

123.     for token in doc:

124.         if token.pos\_ == "ADJ":

125.             sentiment += token.sentiment

126.

127.     # Return the analysis

128.     return entities, keywords, sentiment

129.

130. # Test the function

131. text = "The Democratic Party is a major political party in the United States."

132. entities, keywords, sentiment = analyze\_political\_text(text)

133. print(entities)

134. print(keywords)

135. print(sentiment)

136.

137. import nltk

138. from nltk.tokenize import word\_tokenize

139. from nltk.corpus import stopwords

140. from nltk.stem import WordNetLemmatizer

141.

142. # Load the NLTK data needed for the task

143. nltk.download('punkt')

144. nltk.download('wordnet')

145. nltk.download('stopwords')

146.

147. # Define a function to understand politics

148. def understand\_politics(topic):

149.     # Tokenize the input topic

150.     tokens = word\_tokenize(topic)

151.

152.     # Remove stopwords

153.     tokens = [token for token in tokens if token not in stopwords.words('english')]

154.

155.     # Lemmatize the tokens

156.     lemmatizer = WordNetLemmatizer()

157.     tokens = [lemmatizer.lemmatize(token) for token in tokens]

158.

159.     # Identify the political entities mentioned in the topic

160.     entities = []

161.     for token in tokens:

162.         if token in ["government", "policy", "politician", "party"]:

163.             entities.append(token)

164.

165.     # Generate a response based on the entities

166.     response = "I understand that you are interested in {}.".format(" ".join(entities))

167.

168.     return response

169.

170. # Test the function

171. print(understand\_politics("government policy on healthcare"))

172.

### services\project\_management.py

##### The Code

1. # services/project\_management.py

2.

3. class ProjectManagement:

4.     def \_\_init\_\_(self):

5.         self.projects = {}

6.

7.     def create\_project(self, project\_name):

8.         """Create a new project."""

9.         if project\_name not in self.projects:

10.             self.projects[project\_name] = {"tasks": [], "status": "ongoing"}

11.             return f"Project '{project\_name}' created."

12.         else:

13.             return f"Project '{project\_name}' already exists."

14.

15.     def add\_task(self, project\_name, task):

16.         """Add a task to a project."""

17.         if project\_name in self.projects:

18.             self.projects[project\_name]["tasks"].append(task)

19.             return f"Task '{task}' added to project '{project\_name}'."

20.         else:

21.             return f"Project '{project\_name}' does not exist."

22.

23.     def list\_projects(self):

24.         """List all projects and their tasks."""

25.         if not self.projects:

26.             return "No projects available."

27.         project\_list = ""

28.         for project\_name, details in self.projects.items():

29.             tasks = ', '.join(details["tasks"]) if details["tasks"] else "No tasks."

30.             project\_list += f"Project: {project\_name}, Status: {details['status']}, Tasks: {tasks}\n"

31.         return project\_list.strip()

32.

33.     def complete\_project(self, project\_name):

34.         """Mark a project as completed."""

35.         if project\_name in self.projects:

36.             self.projects[project\_name]["status"] = "completed"

37.             return f"Project '{project\_name}' marked as completed."

38.         else:

39.             return f"Project '{project\_name}' does not exist."

40.

41.

### services\sexual\_content

##### The Code

1. import datetime

2. import random

3.

4. # Define a dictionary of sexual content topics

5. sexual\_content\_topics = {

6.     "anatomy": ["What is the difference between a vagina and a vulva?", "What are the different types of sexual orientation?"],

7.     "physiology": ["How does sexual arousal work?", "What are the physical effects of orgasm?"],

8.     "intimacy": ["What are some ways to build intimacy with a partner?", "How can I communicate my desires to my partner?"],

9.     "consent": ["What is enthusiastic consent?", "How can I ensure I have consent before engaging in sexual activity?"],

10.     "relationships": ["What are some common issues in romantic relationships?", "How can I maintain a healthy and fulfilling relationship?"]

11. }

12.

13. # Define a function to respond to user queries about sexual content

14. def respond\_to\_sexual\_content\_query(query):

15.     # Check if the query is related to sexual content

16.     if any(topic in query.lower() for topic in sexual\_content\_topics.keys()):

17.         # Choose a random topic from the dictionary

18.         topic = random.choice(list(sexual\_content\_topics.keys()))

19.         # Choose a random question from the topic

20.         question = random.choice(sexual\_content\_topics[topic])

21.         # Respond to the user with a helpful and informative answer

22.         return f"I'm happy to help with that! {question} is a great question. Here's what I know: [insert informative answer here]."

23.     else:

24.         # If the query is not related to sexual content, respond with a neutral message

25.         return "I'm not sure I understand what you're asking. Can you please rephrase your question?"

26.

27. # Test the function

28. user\_query = "What is the difference between a vagina and a vulva?"

29. print(respond\_to\_sexual\_content\_query(user\_query))

30.

31. import requests

32. from bs4 import BeautifulSoup

33. import cv2

34. import numpy as np

35. from transformers import pipeline

36.

37. # Define a function to analyze Pornhub content

38. def analyze\_pornhub\_content(url):

39.     # Send a request to the Pornhub page

40.     response = requests.get(url)

41.     # Parse the HTML content using BeautifulSoup

42.     soup = BeautifulSoup(response.content, 'html.parser')

43.     # Extract the video title, description, and tags

44.     title = soup.find('h1', class\_='title').text.strip()

45.     description = soup.find('div', class\_='description').text.strip()

46.     tags = [tag.text.strip() for tag in soup.find\_all('a', class\_='tag')]

47.     # Analyze the video content using computer vision

48.     video\_url = soup.find('source', type='video/mp4')['src']

49.     cap = cv2.VideoCapture(video\_url)

50.     frames = []

51.     while True:

52.         ret, frame = cap.read()

53.         if not ret:

54.             break

55.         frames.append(frame)

56.     # Extract features from the video frames

57.     features = []

58.     for frame in frames:

59.         features.append(cv2.resize(frame, (224, 224)))

60.     features = np.array(features)

61.     # Use a machine learning model to analyze the features

62.     model = pipeline('image-classification')

63.     predictions = model(features)

64.     # Return the analysis results

65.     return title, description, tags, predictions

66.

67. # Define a function to generate social media posts

68. def generate\_social\_media\_post(content\_type):

69.     # Use a natural language processing model to generate a post

70.     model = pipeline('text-generation')

71.     post = model(f"Generate a {content\_type} social media post")

72.     return post

73.

74. # Test the functions

75. url = "https://www.pornhub.com/view\_video.php?viewkey=1234567890"

76. title, description, tags, predictions = analyze\_pornhub\_content(url)

77. print(f"Title: {title}")

78. print(f"Description: {description}")

79. print(f"Tags: {tags}")

80. print(f"Predictions: {predictions}")

81.

82. content\_types = {

83.     "Monday": ["Motivation Monday", "Man Crush Monday"],

84.     "Tuesday": ["Titty Tuesday", "Taco Tuesday"],

85.     "Wednesday": ["Wisdom Wednesday", "Women Crush Wednesday"],

86.     "Thursday": ["Throwback Thursday", "Thirsty Thursday"],

87.     "Friday": ["Friday Feeling", "Follow Friday"],

88.     "Saturday": ["Saturday Night", "Sexy Saturday"],

89.     "Sunday": ["Sunday Funday", "Self Care Sunday"]

90. }

91.

92. # Define a function to get the current day of the week

93. def get\_current\_day():

94.     return datetime.datetime.now().strftime("%A")

95.

96. # Define a function to get the content type for the current day

97. def get\_content\_type(day):

98.     return content\_types.get(day, [])

99.

100. # Test the functions

101. current\_day = get\_current\_day()

102. content\_type = get\_content\_type(current\_day)

103. print(f"Today is {current\_day} and the content type is {content\_type}")

104.

105. # Generate a social media post based on the content type

106. def generate\_post(content\_type):

107.     if content\_type == "Titty Tuesday":

108.         post = "Happy Titty Tuesday, everyone! #TittyTuesday #SexyTuesday"

109.     elif content\_type == "Taco Tuesday":

110.         post = "It's Taco Tuesday, folks! Who's craving tacos today? #TacoTuesday #Foodie"

111.     # Add more content types and posts here

112.     return post

113.

114. post = generate\_post(content\_type[0])

115. print(post)

116.

117. import random

118.

119. class SexualContent:

120.     def \_\_init\_\_(self):

121.         self.knowledge\_base = {

122.             "anatomy": ["vagina", "penis", "clitoris"],

123.             "physiology": ["sexual response cycle", "orgasm"],

124.             "intimacy": ["consent", "communication"],

125.             "relationships": ["monogamy", "polyamory"]

126.         }

127.

128.     def provide\_info(self, topic):

129.         if topic in self.knowledge\_base:

130.             return f"Ah, {topic}! I can tell you all about that. {self.knowledge\_base[topic][0]} is a great place to start."

131.         else:

132.             return "I'm not familiar with that topic, but I can try to learn more about it!"

133.

134. jaicat\_sexual\_content = SexualContent()

135.

136. # Example interaction

137. user\_input = "What's the deal with intimacy?"

138. response = jaicat\_sexual\_content.provide\_info("intimacy")

139. print(response)  # Output: Ah, intimacy! I can tell you all about that. consent is a great place to start.

140.

141. import torch

142. import torch.nn as nn

143. import torch.optim as optim

144. from torch.utils.data import Dataset, DataLoader

145. import pandas as pd

146. import numpy as np

147. from sklearn.model\_selection import train\_test\_split

148. from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

149.

150. # Load the dataset

151. dataset = pd.read\_csv("sexual\_content\_dataset.csv")

152.

153. # Preprocess the data

154. X = dataset["text"]

155. y = dataset["label"]

156.

157. # Split the data into training and testing sets

158. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

159.

160. # Create a custom dataset class for our data

161. class SexualContentDataset(Dataset):

162.     def \_\_init\_\_(self, X, y):

163.         self.X = X

164.         self.y = y

165.

166.     def \_\_len\_\_(self):

167.         return len(self.X)

168.

169.     def \_\_getitem\_\_(self, idx):

170.         text = self.X.iloc[idx]

171.         label = self.y.iloc[idx]

172.         return {

173.             "text": text,

174.             "label": label

175.         }

176.

177. # Create data loaders for the training and testing sets

178. train\_dataset = SexualContentDataset(X\_train, y\_train)

179. test\_dataset = SexualContentDataset(X\_test, y\_test)

180.

181. train\_loader = DataLoader(train\_dataset, batch\_size=32, shuffle=True)

182. test\_loader = DataLoader(test\_dataset, batch\_size=32, shuffle=False)

183.

184. # Define the model architecture

185. class SexualContentModel(nn.Module):

186.     def \_\_init\_\_(self):

187.         super(SexualContentModel, self).\_\_init\_\_()

188.         self.embedding = nn.Embedding(num\_embeddings=10000, embedding\_dim=128)

189.         self.fc1 = nn.Linear(128, 64)

190.         self.fc2 = nn.Linear(64, 2)

191.

192.     def forward(self, x):

193.         x = self.embedding(x)

194.         x = torch.relu(self.fc1(x))

195.         x = self.fc2(x)

196.         return x

197.

198. # Initialize the model, optimizer, and loss function

199. model = SexualContentModel()

200. optimizer = optim.Adam(model.parameters(), lr=0.001)

201. loss\_fn = nn.CrossEntropyLoss()

202.

203. # Train the model

204. for epoch in range(10):

205.     for batch in train\_loader:

206.         text = batch["text"]

207.         label = batch["label"]

208.         optimizer.zero\_grad()

209.         output = model(text)

210.         loss = loss\_fn(output, label)

211.         loss.backward()

212.         optimizer.step()

213.     print(f"Epoch {epoch+1}, Loss: {loss.item()}")

214.

215. # Evaluate the model

216. model.eval()

217. test\_loss = 0

218. correct = 0

219. with torch.no\_grad():

220.     for batch in test\_loader:

221.         text = batch["text"]

222.         label = batch["label"]

223.         output = model(text)

224.         loss = loss\_fn(output, label)

225.         test\_loss += loss.item()

226.         \_, predicted = torch.max(output, 1)

227.         correct += (predicted == label).sum().item()

228.

229. accuracy = correct / len(test\_loader.dataset)

230. print(f"Test Accuracy: {accuracy:.4f}")

231.

232. # Use the model to generate sexual content

233. def generate\_sexual\_content(prompt):

234.     input\_text = torch.tensor([prompt])

235.     output = model(input\_text)

236.     \_, predicted = torch.max(output, 1)

237.     return predicted.item()

238.

239. # Example usage

240. prompt = "I want to know more about BDSM"

241. generated\_content = generate\_sexual\_content(prompt)

242. print(f"Generated Content: {generated\_content}")

243.

244. import torch

245. from transformers import AutoModelForSequenceClassification, AutoTokenizer

246.

247. # Load the model and tokenizer

248. model = AutoModelForSequenceClassification.from\_pretrained("nudenet-base-uncased")

249. tokenizer = AutoTokenizer.from\_pretrained("nudenet-base-uncased")

250.

251. # Define a function to analyze user input and generate a response

252. def generate\_sexual\_content(user\_input):

253.     # Tokenize the user input

254.     inputs = tokenizer(user\_input, return\_tensors="pt")

255.

256.     # Analyze the user input using the model

257.     outputs = model(\*\*inputs)

258.

259.     # Generate a response based on the analysis

260.     response = "I understand that you are interested in discussing sexual content. Here is some information on the topic: [insert relevant information]"

261.

262.     return response

263.

264. # Test the function

265. user\_input = "I want to know more about BDSM"

266. response = generate\_sexual\_content(user\_input)

267. print(response)

268.

269. import os

270. from azure.ai.contentsafety import ContentSafetyClient

271. from azure.ai.contentsafety.models import TextCategory

272. from azure.core.credentials import AzureKeyCredential

273. from azure.core.exceptions import HttpResponseError

274.

275. # Set up the Content Safety client

276. key = os.environ["CONTENT\_SAFETY\_KEY"]

277. endpoint = os.environ["CONTENT\_SAFETY\_ENDPOINT"]

278. client = ContentSafetyClient(endpoint, AzureKeyCredential(key))

279.

280. # Define a function to analyze user input and generate a response

281. def generate\_sexual\_content(user\_input):

282.     # Analyze the user input using the Content Safety service

283.     request = {"text": user\_input}

284.     response = client.analyze\_text(request)

285.

286.     # Generate a response based on the analysis

287.     if response.categories\_analysis[0].category == "Sexual Content":

288.         response\_text = "I understand that you are interested in discussing sexual content. Here is some information on the topic: [insert relevant information]"

289.     else:

290.         response\_text = "I'm not sure what you mean by that. Can you please rephrase?"

291.

292.     return response\_text

293.

294. # Test the function

295. user\_input = "I want to know more about BDSM"

296. response = generate\_sexual\_content(user\_input)

297. print(response)

298.

299.

### services\smart\_home.py

##### The Code

1. import requests

2.

3. class SmartHomeService:

4.     def \_\_init\_\_(self, base\_url, api\_key):

5.         self.base\_url = base\_url

6.         self.api\_key = api\_key

7.

8.     def control\_device(self, device\_id, action):

9.         url = f"{self.base\_url}/devices/{device\_id}/{action}?api\_key={self.api\_key}"

10.         response = requests.post(url)

11.         if response.status\_code == 200:

12.             return f"{action.capitalize()} command sent to device {device\_id}."

13.         else:

14.             return "Failed to control the device."

15.

16.

### services\Social\_media\_monitor.py

##### The Code

1. import tweepy

2.

3. # Twitter API credentials

4. consumer\_key = "your\_consumer\_key\_here"

5. consumer\_secret = "your\_consumer\_secret\_here"

6. access\_token = "your\_access\_token\_here"

7. access\_token\_secret = "your\_access\_token\_secret\_here"

8.

9. # Set up the Tweepy API object

10. auth = tweepy.OAuthHandler(consumer\_key, consumer\_secret)

11. auth.set\_access\_token(access\_token, access\_token\_secret)

12. api = tweepy.API(auth)

13.

14. # Define a function to post a tweet

15. def post\_tweet(tweet):

16.     api.update\_status(status=tweet)

17.

18. # Define a function to get the current day of the week

19. def get\_current\_day():

20.     return datetime.datetime.now().strftime("%A")

21.

22. # Define a function to generate a tweet based on the current day

23. def generate\_tweet():

24.     current\_day = get\_current\_day()

25.     if current\_day == "Tuesday":

26.         tweet = "Happy Titty Tuesday, everyone! #TittyTuesday #SexyTuesday"

27.     elif current\_day == "Wednesday":

28.         tweet = "Happy Wisdom Wednesday, everyone! #WisdomWednesday #WednesdayWisdom"

29.     # Add more days and tweets here

30.     return tweet

31.

32. # Post a tweet

33. tweet = generate\_tweet()

34. post\_tweet(tweet)

35.

36. import instapy

37.

38. # Instagram API credentials

39. username = "your\_username\_here"

40. password = "your\_password\_here"

41. api = instapy.InstaPy(username, password)

42.

43. # Define a function to post a photo on Instagram

44. def post\_photo(photo\_path, caption):

45.     api.upload\_photo(photo\_path, caption)

46.

47. # Define a function to generate a caption based on the current day

48. def generate\_caption():

49.     current\_day = datetime.datetime.now().strftime("%A")

50.     if current\_day == "Tuesday":

51.         caption = "Happy Titty Tuesday, everyone! #TittyTuesday #SexyTuesday"

52.     elif current\_day == "Wednesday":

53.         caption = "Happy Wisdom Wednesday, everyone! #WisdomWednesday #WednesdayWisdom"

54.     # Add more days and captions here

55.     return caption

56.

57. # Post a photo on Instagram

58. photo\_path = "path\_to\_your\_photo\_here"

59. caption = generate\_caption()

60. post\_photo(photo\_path, caption)

61.

62. import tweepy

63. import instapy

64.

65. # Twitter API credentials

66. twitter\_consumer\_key = "your\_consumer\_key\_here"

67. twitter\_consumer\_secret = "your\_consumer\_secret\_here"

68. twitter\_access\_token = "your\_access\_token\_here"

69. twitter\_access\_token\_secret = "your\_access\_token\_secret\_here"

70.

71. # Instagram API credentials

72. instagram\_username = "your\_username\_here"

73. instagram\_password = "your\_password\_here"

74.

75. # Set up the Tweepy API object

76. auth = tweepy.OAuthHandler(twitter\_consumer\_key, twitter\_consumer\_secret)

77. auth.set\_access\_token(twitter\_access\_token, twitter\_access\_token\_secret)

78. api = tweepy.API(auth)

79.

80. # Set up the InstaPy API object

81. instagram\_api = instapy.InstaPy(username=instagram\_username, password=instagram\_password)

82.

83. # Define a function to post a tweet on multiple Twitter accounts

84. def post\_tweet(tweet, accounts):

85.     for account in accounts:

86.         api.update\_status(status=tweet, screen\_name=account)

87.

88. # Define a function to post a photo on multiple Instagram accounts

89. def post\_photo(photo\_path, caption, accounts):

90.     for account in accounts:

91.         instagram\_api.upload\_photo(photo\_path, caption, account)

92.

93. # Define a list of Twitter accounts to post to

94. twitter\_accounts = ["account1", "account2", "account3"]

95.

96. # Define a list of Instagram accounts to post to

97. instagram\_accounts = ["account1", "account2", "account3"]

98.

99. # Post a tweet on multiple Twitter accounts

100. tweet = "Hello, world!"

101. post\_tweet(tweet, twitter\_accounts)

102.

103. # Post a photo on multiple Instagram accounts

104. photo\_path = "path\_to\_your\_photo\_here"

105. caption = "Hello, world!"

106. post\_photo(photo\_path, caption, instagram\_accounts)

107.

### services\spotify\_integration.py

##### The Code

1. import spotipy

2. from spotipy.oauth2 import SpotifyOAuth

3.

4. class SpotifyIntegration:

5.     def \_\_init\_\_(self):

6.         self.sp = spotipy.Spotify(auth\_manager=SpotifyOAuth(client\_id="088813c3e61b4c4eaec0ba035d3c398f",

7.                                                            client\_secret="160ba7377f0e4260bc5d473538a54f1b",

8.                                                            redirect\_uri="http://localhost:3000/auth/callback",

9.                                                            scope="user-library-read"))

10.

11.     def play\_song(self, song\_name):

12.         results = self.sp.search(q=song\_name, limit=1)

13.         if results['tracks']['items']:

14.             song\_uri = results['tracks']['items'][0]['uri']

15.             self.sp.start\_playback(uris=[song\_uri])

16.             return f"Now playing {song\_name}"

17.         else:

18.             return "Song not found."

19.

20.

### services\swann\_cctv.py

##### The Code

1. # services/swann\_cctv.py

2.

3. import requests

4. import json

5.

6. class SwannCCTV:

7.     def \_\_init\_\_(self, api\_key, base\_url):

8.         self.api\_key = api\_key

9.         self.base\_url = base\_url

10.

11.     def get\_camera\_status(self):

12.         """Get the status of all connected Swann cameras."""

13.         response = requests.get(f"{self.base\_url}/cameras/status", headers={"Authorization": f"Bearer {self.api\_key}"})

14.         if response.status\_code == 200:

15.             return response.json()  # Returns JSON response with camera status

16.         else:

17.             return {"error": "Failed to retrieve camera status."}

18.

19.     def capture\_snapshot(self, camera\_id):

20.         """Capture a snapshot from a specific camera."""

21.         response = requests.post(f"{self.base\_url}/cameras/{camera\_id}/snapshot", headers={"Authorization": f"Bearer {self.api\_key}"})

22.         if response.status\_code == 200:

23.             # Save snapshot if necessary

24.             return response.json()  # URL or path of the snapshot

25.         else:

26.             return {"error": "Failed to capture snapshot."}

27.

28.     def start\_recording(self, camera\_id):

29.         """Start recording from a specific camera."""

30.         response = requests.post(f"{self.base\_url}/cameras/{camera\_id}/start-recording", headers={"Authorization": f"Bearer {self.api\_key}"})

31.         return response.status\_code == 200

32.

33.     def stop\_recording(self, camera\_id):

34.         """Stop recording from a specific camera."""

35.         response = requests.post(f"{self.base\_url}/cameras/{camera\_id}/stop-recording", headers={"Authorization": f"Bearer {self.api\_key}"})

36.         return response.status\_code == 200

37.

### services\travel\_recommendations.py

##### The Code

1. # services/travel\_recommendations.py

2.

3. class TravelRecommendations:

4.     def \_\_init\_\_(self):

5.         # Dictionary to hold travel destination categories and their suggestions

6.         self.destinations = {

7.             "beach": [

8.                 "Hawaii",

9.                 "Bahamas",

10.                 "Maldives",

11.                 "Seychelles",

12.                 "Bora Bora"

13.             ],

14.             "mountain": [

15.                 "Swiss Alps",

16.                 "Rocky Mountains",

17.                 "Himalayas",

18.                 "Andes",

19.                 "Appalachians"

20.             ],

21.             "city": [

22.                 "New York City",

23.                 "Paris",

24.                 "Tokyo",

25.                 "Barcelona",

26.                 "London"

27.             ],

28.             "cultural": [

29.                 "Rome",

30.                 "Athens",

31.                 "Jerusalem",

32.                 "Beijing",

33.                 "Istanbul"

34.             ],

35.             "adventure": [

36.                 "Costa Rica",

37.                 "New Zealand",

38.                 "Iceland",

39.                 "Patagonia",

40.                 "South Africa"

41.             ]

42.         }

43.

44.     def get\_recommendations(self, category):

45.         """Return travel recommendations based on the category."""

46.         # Convert category to lower case to ensure case-insensitive matching

47.         category = category.lower()

48.         # Get recommendations from the dictionary

49.         recommendations = self.destinations.get(category, ["No recommendations available."])

50.         return recommendations

51.

52.

### services\usb\_cam.py

##### The Code

1. # services/usb\_cam.py

2.

3. import cv2

4.

5. class USBCam:

6.     def \_\_init\_\_(self):

7.         """Initialize the USB camera."""

8.         self.camera = None

9.

10.     def start\_camera(self):

11.         """Start the USB camera."""

12.         self.camera = cv2.VideoCapture(0)  # 0 is usually the default camera index

13.

14.         if not self.camera.isOpened():

15.             raise Exception("Could not open the USB camera.")

16.

17.     def capture\_image(self, filename='captured\_image.png'):

18.         """Capture an image from the camera and save it to a file."""

19.         if self.camera is None:

20.             raise Exception("Camera is not started. Call start\_camera() first.")

21.

22.         ret, frame = self.camera.read()

23.         if not ret:

24.             raise Exception("Failed to capture image.")

25.

26.         cv2.imwrite(filename, frame)

27.         print(f"Image saved as {filename}")

28.

29.     def release\_camera(self):

30.         """Release the camera resource."""

31.         if self.camera is not None:

32.             self.camera.release()

33.             self.camera = None

34.             print("Camera released.")

35.

36. # Example usage:

37. if \_\_name\_\_ == "\_\_main\_\_":

38.     usb\_cam = USBCam()

39.     try:

40.         usb\_cam.start\_camera()

41.         usb\_cam.capture\_image("test\_image.png")  # Change the filename as needed

42.     finally:

43.         usb\_cam.release\_camera()

44.

45.

### services\ user\_enrollment.py

##### The Code

1. import json

2. from cryptography.fernet import Fernet

3.

4. class UserEnrollmentService:

5.     def \_\_init\_\_(self, encryption\_key, user\_data\_path):

6.         self.encryption\_key = encryption\_key

7.         self.cipher = Fernet(self.encryption\_key)

8.         self.user\_data\_path = user\_data\_path

9.

10.     def save\_user\_data(self, user\_data):

11.         """Save user data securely with encryption."""

12.         json\_data = json.dumps(user\_data).encode()

13.         encrypted\_data = self.cipher.encrypt(json\_data)

14.         with open(self.user\_data\_path, 'wb') as f:

15.             f.write(encrypted\_data)

16.

17.     def add\_vehicle\_to\_user(self, user, vehicle\_data):

18.         """Add a vehicle to the user's profile."""

19.         user['vehicles'].append(vehicle\_data)

20.         self.save\_user\_data(user)

21.         return "Vehicle added to your profile."

22.

23. # Example integration in main.py

24. from services.user\_enrollment import UserEnrollmentService

25.

26. class Jaicat:

27.     def \_\_init\_\_(self):

28.         # Initialize UserEnrollmentService

29.         self.user\_enrollment\_service = UserEnrollmentService(encryption\_key='YOUR\_ENCRYPTION\_KEY', user\_data\_path='path/to/user\_data.json')

30.

31.     def add\_vehicle(self, plate, model, make):

32.         """Add a vehicle to the current user's profile."""

33.         vehicle\_data = {"plate": plate, "make": make, "model": model}

34.         response = self.user\_enrollment\_service.add\_vehicle\_to\_user(self.current\_user, vehicle\_data)

35.         self.speak(response)

36.

37.     def process\_user\_input(self, user\_input):

38.         # Check if the user wants to add a vehicle

39.         if "add vehicle" in user\_input.lower():

40.             plate = input("Enter the license plate: ")

41.             make = input("Enter the vehicle make: ")

42.             model = input("Enter the vehicle model: ")

43.             self.add\_vehicle(plate, make, model)

44.

45.

### Services\vehicle\_lookup.py

##### The code

1. import requests

2.

3. class VehicleLookupService:

4. def \_\_init\_\_(self, api\_key):

5. self.api\_key = api\_key

6.

7. def lookup\_vehicle(self, reg\_plate):

8. url = f"https://vpic.nhtsa.dot.gov/api/vehicles/DecodeVinValuesExtended/{reg\_plate}?format=json"

9. response = requests.get(url)

10. if response.status\_code == 200:

11. data = response.json()

12. vehicle\_info = {

13. "Make": data.get("Make"),

14. "Model": data.get("Model"),

15. "Year": data.get("ModelYear"),

16. "Fuel": data.get("FuelTypePrimary")

17. }

18. return vehicle\_info

19. else:

20. return "Vehicle not found or invalid registration."

21.

### services\weather\_api.py

##### The Code

1. def get\_weather(city):

2.     api\_key = "YOUR\_OPENWEATHERMAP\_API\_KEY"

3.     url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api\_key}"

4.     response = requests.get(url)

5.     weather\_data = json.loads(response.text)

6.     return weather\_data["weather"][0]["description"]

7.

### services\weather\_service.py

##### The Code

1. import os

2. import requests

3.

4. class WeatherService:

5.     def \_\_init\_\_(self):

6.         self.api\_key = os.getenv("VXy1cHjASsehJvfOVB5mf38pthgg71ix")  # Load API key from environment variable

7.         self.base\_url = "https://api.tomorrow.io/v4/weather/forecast?location=42.3478,-71.0466&apikey=VXy1cHjASsehJvfOVB5mf38pthgg71ix"

8.

9.     def get\_weather(self, lat, lon):

10.         """Fetch weather data for given latitude and longitude."""

11.         complete\_url = f"{self.base\_url}current?lat={lat}&lon={lon}&apikey={self.api\_key}"

12.

13.         response = requests.get(complete\_url)

14.

15.         if response.status\_code == 200:

16.             data = response.json()

17.             weather\_desc = data["data"][0]["weather"][0]["description"]

18.             temperature = data["data"][0]["temp"]

19.             return {

20.                 "description": weather\_desc.capitalize(),

21.                 "temperature": f"{temperature}°C",

22.                 "latitude": lat,

23.                 "longitude": lon

24.             }

25.         else:

26.             return {"error": "Error fetching weather data."}

27.

### services\youtube\_analysis.py

##### The Code

1. import cv2

2. import youtube\_dl

3. import speech\_recognition as sr

4. from transformers import pipeline

5.

6. # Load the video from YouTube

7. video\_url = "https://www.youtube.com/watch?v=dQw4w9WgXcQ"

8. ydl\_opts = {'format': 'best'}

9. with youtube\_dl.YoutubeDL(ydl\_opts) as ydl:

10.     video = ydl.download([video\_url])

11.

12. # Extract audio from the video

13. audio = cv2.VideoCapture(video)

14. audio.set(cv2.CAP\_PROP\_POS\_MSEC, 0)

15. audio\_frames = []

16. while True:

17.     ret, frame = audio.read()

18.     if not ret:

19.         break

20.     audio\_frames.append(frame)

21.

22. # Transcribe the audio using speech recognition

23. r = sr.Recognizer()

24. transcript = ""

25. for frame in audio\_frames:

26.     audio\_data = sr.AudioData(frame, 16000, 2)

27.     try:

28.         transcript += r.recognize\_google(audio\_data, language="en-US")

29.     except sr.UnknownValueError:

30.         pass

31.

32. # Analyze the transcript using natural language processing

33. nlp = pipeline("sentiment-analysis")

34. sentiment = nlp(transcript)

35.

36. # Generate a summary of the video

37. summary = ""

38. for sentence in transcript.split("."):

39.     summary += sentence + " "

40.

41. # Speak about the video using text-to-speech

42. from gtts import gTTS

43. tts = gTTS(text=summary, lang='en')

44. tts.save("summary.mp3")

45.

46. # Play the summary

47. from pygame import mixer

48. mixer.init()

49. mixer.music.load("summary.mp3")

50. mixer.music.play()

51.

52. import googleapiclient.discovery

53.

54. # Define a function to analyze a YouTube video

55. def analyze\_youtube\_video(video\_id):

56.     # Create a YouTube Data API client

57.     youtube = googleapiclient.discovery.build('youtube', 'v3', developerKey='YOUR\_API\_KEY')

58.     # Retrieve the video metadata

59.     request = youtube.videos().list(part='snippet', id=video\_id)

60.     response = request.execute()

61.     video\_title = response['items'][0]['snippet']['title']

62.     video\_description = response['items'][0]['snippet']['description']

63.     # Perform video analysis (e.g., sentiment analysis, topic modeling)

64.     #...

65.     return analysis\_results

66.

67. # Example usage:

68. video\_id = "VIDEO\_ID\_HERE"

69. analysis\_results = analyze\_youtube\_video(video\_id)

70. print(analysis\_results)

71.

72. import requests

73. from bs4 import BeautifulSoup

74. import PyPDF2

75. from PIL import Image

76. from youtube\_transcript\_api import YouTubeTranscriptApi

77.

78. def implement\_web\_scraping\_and\_analysis():

79.     # Define a function to scrape a website

80.     def scrape\_website(url):

81.         # Send a GET request to the website

82.         response = requests.get(url)

83.

84.         # Parse the HTML content

85.         soup = BeautifulSoup(response.content, 'html.parser')

86.

87.         # Return the parsed HTML content

88.         return soup

89.

90.     # Define a function to parse a PDF file

91.     def parse\_pdf\_file(file\_path):

92.         # Open the PDF file

93.         pdf\_file = open(file\_path, 'rb')

94.

95.         # Create a PDF reader object

96.         pdf\_reader = PyPDF2.PdfFileReader(pdf\_file)

97.

98.         # Extract the text from the PDF file

99.         text = ''

100.         for page in range(pdf\_reader.numPages):

101.             text += pdf\_reader.getPage(page).extractText()

102.

103.         # Return the extracted text

104.         return text

105.

106.     # Define a function to analyze an image

107.     def analyze\_image(image\_path):

108.         # Open the image file

109.         image = Image.open(image\_path)

110.

111.         # Analyze the image

112.         # ...

113.

114.         # Return the analysis result

115.         return 'Image analysis result'

116.

117.     # Define a function to analyze a YouTube video

118.     def analyze\_youtube\_video(video\_id):

119.         # Get the video transcript

120.         transcript = YouTubeTranscriptApi.get\_transcript(video\_id)

121.

122.         # Analyze the transcript

123.         # ...

124.

125.         # Return the analysis result

126.         return 'Video analysis result'

127.

128.     # Return the web scraping and analysis interface

129.     return scrape\_website, parse\_pdf\_file, analyze

130.

## TESTING OPENAI

### TESTING OPENAI\openai test.py

## Tests

### Tests\test\_dialogue\_manager.py

### Tests\test\_nlg.py

### Tests\test\_nlu.py

### Tests\test\_services.py

### Tests\test\_utilities.py

## Ui

### ui\JaicatUI.py

##### The Code

1. import tkinter as tk

2. from PIL import Image, ImageTk

3.

4. class UI:

5.     def \_\_init\_\_(self, master):

6.         self.master = master

7.         self.master.title("Jaicat - AI Assistant")

8.         self.master.geometry("1200x800")  # Set window size

9.         self.master.configure(bg="#2C3E50")  # Change background color

10.

11.         self.create\_widgets()

12.

13.     def create\_widgets(self):

14.         """Set up the UI components like buttons, labels, input boxes, etc."""

15.

16.         # Frame to contain all widgets

17.         main\_frame = tk.Frame(self.master, bg="#2C3E50")

18.         main\_frame.pack(fill="both", expand=True)

19.

20.         # Label to show the face image

21.         self.face\_image\_label = tk.Label(main\_frame)

22.         self.face\_image\_label.grid(row=0, column=0, padx=10, pady=10, columnspan=2)

23.

24.         # Button to play a song via Spotify

25.         self.play\_button = tk.Button(main\_frame, text="Play Music", command=self.master.play\_song, font=("Arial", 12), bg="#3498db", fg="white")

26.         self.play\_button.grid(row=1, column=0, padx=10, pady=10)

27.

28.         # Calendar Widget

29.         self.calendar\_label = tk.Label(main\_frame, text="Calendar", font=("Arial", 14), bg="#2C3E50", fg="white")

30.         self.calendar\_label.grid(row=2, column=0, padx=10, pady=10)

31.         self.calendar = tk.Label(main\_frame, text=self.master.calendar\_service.get\_current\_date(), font=("Arial", 14), bg="#1ABC9C", fg="white")

32.         self.calendar.grid(row=3, column=0, padx=10, pady=10)

33.

34.         # Weather Label

35.         self.weather\_label = tk.Label(main\_frame, text="Weather", font=("Arial", 14), bg="#2C3E50", fg="white")

36.         self.weather\_label.grid(row=2, column=1, padx=10, pady=10)

37.         self.weather = tk.Label(main\_frame, text=self.master.weather\_service.get\_weather(), font=("Arial", 14), bg="#1ABC9C", fg="white")

38.         self.weather.grid(row=3, column=1, padx=10, pady=10)

39.

40.         # Circular Button for user input

41.         self.input\_button = tk.Button(main\_frame, text="Text Input", command=self.master.show\_input\_box, font=("Arial", 12), bg="#E74C3C", fg="white")

42.         self.input\_button.grid(row=4, column=0, padx=10, pady=10)

43.

44.         # Hidden text input box

45.         self.text\_input = tk.Entry(main\_frame, font=("Arial", 14))

46.         self.text\_input.grid(row=5, column=0, padx=10, pady=10, columnspan=2)

47.         self.text\_input.grid\_remove()

48.

49.         # Status Bar

50.         self.status\_bar = tk.Label(self.master, text="Ready", bd=1, relief=tk.SUNKEN, anchor=tk.W, font=("Arial", 12))

51.         self.status\_bar.pack(side=tk.BOTTOM, fill=tk.X)

52.

### ui\pyqt\_utils.py

## utils

### utils\file\_handling.py

##### The Code

1. # utils/file\_handling.py

2.

3. import os

4.

5. def read\_file(file\_path):

6.     """Reads the content of a text file and returns it as a string."""

7.     if not os.path.isfile(file\_path):

8.         return "File does not exist."

9.

10.     with open(file\_path, 'r', encoding='utf-8') as file:

11.         return file.read()

12.

13. def write\_to\_file(file\_path, content):

14.     """Writes content to a text file. If the file exists, it overwrites it."""

15.     with open(file\_path, 'w', encoding='utf-8') as file:

16.         file.write(content)

17.     return "Content written successfully."

18.

19. def file\_exists(file\_path):

20.     """Checks if a file exists."""

21.     return os.path.isfile(file\_path)

22.

23. def get\_file\_extension(file\_path):

24.     """Returns the file extension for a given file path."""

25.     return os.path.splitext(file\_path)[1]

26.

27. def list\_files\_in\_directory(directory\_path):

28.     """Returns a list of files in the specified directory."""

29.     if not os.path.isdir(directory\_path):

30.         return "Directory does not exist."

31.

32.     return os.listdir(directory\_path)

33.

### utils\image\_processing.py

**Purpose**: Contains utility functions for processing images, likely used for face recognition or visual tasks.

#### Key Functions:

* + **process\_image()**: Processes an image to make it compatible with the face recognition system.
* **Connection to main.py**:
  + This file might be called when Jaicat performs tasks like face recognition or object detection.

##### The Code

1. import cv2

2. import numpy as np

3.

4. class ImageProcessor:

5.     def \_\_init\_\_(self):

6.         pass

7.

8.     def load\_image(self, image\_path):

9.         """

10.         Load an image from a file path

11.         """

12.         return cv2.imread(image\_path)

13.

14.     def resize\_image(self, image, width, height):

15.         """

16.         Resize an image to a specified width and height

17.         """

18.         return cv2.resize(image, (width, height))

19.

20.     def convert\_to\_grayscale(self, image):

21.         """

22.         Convert an image to grayscale

23.         """

24.         return cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

25.

26.     def apply\_threshold(self, image, threshold\_value):

27.         """

28.         Apply a threshold to an image

29.         """

30.         \_, thresh = cv2.threshold(image, threshold\_value, 255, cv2.THRESH\_BINARY)

31.         return thresh

32.

33.     def detect\_faces(self, image):

34.         """

35.         Detect faces in an image using OpenCV's Haar cascade classifier

36.         """

37.         face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml')

38.         gray = self.convert\_to\_grayscale(image)

39.         faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

40.         return faces

41.

42.     def draw\_rectangles(self, image, rectangles):

43.         """

44.         Draw rectangles on an image

45.         """

46.         for (x, y, w, h) in rectangles:

47.             cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)

48.         return image

49.

50.     def save\_image(self, image, image\_path):

51.         """

52.         Save an image to a file path

53.         """

54.         cv2.imwrite(image\_path, image)

55.

56. # Example usage:

57. if \_\_name\_\_ == "\_\_main\_\_":

58.     image\_processor = ImageProcessor()

59.     image = image\_processor.load\_image("image.jpg")

60.     resized\_image = image\_processor.resize\_image(image, 640, 480)

61.     grayscale\_image = image\_processor.convert\_to\_grayscale(resized\_image)

62.     thresholded\_image = image\_processor.apply\_threshold(grayscale\_image, 127)

63.     faces = image\_processor.detect\_faces(resized\_image)

64.     drawn\_image = image\_processor.draw\_rectangles(resized\_image, faces)

65.     image\_processor.save\_image(drawn\_image, "output.jpg")

66.

### utils\moving\_image.py

**Purpose**: Handles the movement or animation of images, likely for visual effects or dynamic UI features in Jaicat.

##### Key Functions:

* + **Image Animation**: Provides methods for moving or animating images on the screen.
* **Connection to main.py**:
  + If Jaicat includes animated visuals (e.g., moving faces or backgrounds), main.py will use this file for those effects.

##### The Code

1. import cv2

2. import numpy as np

3.

4. # Load the video capture device (e.g. camera)

5. cap = cv2.VideoCapture(0)

6.

7. # Set the video codec and create a video writer

8. fourcc = cv2.VideoWriter\_fourcc(\*'XVID')

9. out = cv2.VideoWriter('output.avi', fourcc, 20.0, (640, 480))

10.

11. while True:

12.     # Read a frame from the camera

13.     ret, frame = cap.read()

14.

15.     # Convert the frame to grayscale

16.     gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

17.

18.     # Apply a Gaussian blur to the grayscale image

19.     blurred = cv2.GaussianBlur(gray, (5, 5), 0)

20.

21.     # Detect edges in the blurred image

22.     edges = cv2.Canny(blurred, 50, 150)

23.

24.     # Find contours in the edge image

25.     contours, \_ = cv2.findContours(edges, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

26.

27.     # Iterate through the contours and draw a rectangle around each one

28.     for contour in contours:

29.         x, y, w, h = cv2.boundingRect(contour)

30.         cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

31.

32.     # Write the output frame to the video file

33.     out.write(frame)

34.

35.     # Display the output frame

36.     cv2.imshow('frame', frame)

37.

38.     # Exit on key press

39.     if cv2.waitKey(1) & 0xFF == ord('q'):

40.         break

41.

42. # Release the video capture device and close the video writer

43. cap.release()

44. out.release()

45. cv2.destroyAllWindows()

46.

### utils\sync\_relationships.py

**Purpose**: Syncs relationships between different datasets or models, ensuring that information remains up-to-date and consistent.

* **Key Functions**:
  + **Data Synchronization**: Ensures that user data, models, or external services are kept in sync.
* **Connection to main.py**:
  + If Jaicat uses multiple datasets or services that need to stay synchronized (e.g., Google Calendar, Spotify), main.py will use this file to handle syncing.

##### The Code

1. import cv2

2. import face\_recognition

3. from sklearn.neighbors import NearestNeighbors

4.

5. # Load the facial recognition model

6. face\_recognition\_model = face\_recognition.FaceRecognition()

7.

8. # Load the phone contacts and photos

9. contacts = []  # list of contacts with names and relationships

10. photos = []  # list of photos with faces and labels

11.

12. # Iterate through the contacts and photos

13. for contact in contacts:

14.     name = contact["name"]

15.     relationship = contact["relationship"]

16.     photo = contact["photo"]

17.

18.     # Extract the face from the photo

19.     face = face\_recognition\_model.extract\_face(photo)

20.

21.     # Add the face to the library

22.     face\_recognition\_model.add\_face(face, name, relationship)

23.

24. # Create a nearest neighbors model to recognize faces

25. nn\_model = NearestNeighbors(n\_neighbors=1, algorithm="ball\_tree")

26. nn\_model.fit(face\_recognition\_model.faces)

27.

28. # Define a function to recognize people in new photos

29. def recognize\_people(photo):

30.     face = face\_recognition\_model.extract\_face(photo)

31.     distances, indices = nn\_model.kneighbors([face])

32.     name = face\_recognition\_model.names[indices[0][0]]

33.     relationship = face\_recognition\_model.relationships[indices[0][0]]

34.     return name, relationship

35.

36. # Example usage:

37. photo = cv2.imread("new\_photo.jpg")

38. name, relationship = recognize\_people(photo)

39. print(f"Recognized person: {name} ({relationship})")

40.

### utils\text\_processing.py

**Purpose**: Handles basic text processing tasks such as tokenization, cleaning, and formatting.

##### Key Functions:

* + **Text Cleaning**: Strips unwanted characters or tokens from text.
  + **Tokenization**: Splits text into individual tokens for further analysis.
* **Connection to main.py**:
  + Jaicat will use this file for initial text preprocessing before feeding the text into NLP models for further analysis.

##### The Code

1. import nltk

2. nltk.download('stopwords')

3. import nltk

4. import spacy

5. from nltk.tokenize import word\_tokenize

6. from nltk.corpus import stopwords

7. from nltk.stem import WordNetLemmatizer

8. from spacy import displacy

9.

10. nlp = spacy.load("en\_core\_web\_sm")

11.

12. class TextProcessor:

13.     def \_\_init\_\_(self):

14.         pass

15.

16.     def tokenize\_text(self, text):

17.         """

18.         Tokenize a text into individual words

19.         """

20.         return word\_tokenize(text)

21.

22.     def remove\_stopwords(self, tokens):

23.         """

24.         Remove stopwords from a list of tokens

25.         """

26.         stop\_words = set(stopwords.words("english"))

27.         filtered\_tokens = [token for token in tokens if token not in stop\_words]

28.         return filtered\_tokens

29.

30.     def lemmatize\_tokens(self, tokens):

31.         """

32.         Lemmatize a list of tokens

33.         """

34.         lemmatizer = WordNetLemmatizer()

35.         lemmatized\_tokens = [lemmatizer.lemmatize(token) for token in tokens]

36.         return lemmatized\_tokens

37.

38.     def perform\_named\_entity\_recognition(self, text):

39.         """

40.         Perform named entity recognition on a text

41.         """

42.         doc = nlp(text)

43.         entities = [(entity.text, entity.label\_) for entity in doc.ents]

44.         return entities

45.

46.     def perform\_part\_of\_speech\_tagging(self, text):

47.         """

48.         Perform part-of-speech tagging on a text

49.         """

50.         doc = nlp(text)

51.         pos\_tags = [(token.text, token.pos\_) for token in doc]

52.         return pos\_tags

53.

54.     def perform\_dependency\_parsing(self, text):

55.         """

56.         Perform dependency parsing on a text

57.         """

58.         doc = nlp(text)

59.         dependencies = [(token.text, token.dep\_, token.head.text, token.head.pos\_) for token in doc]

60.         return dependencies

61.

62.     def visualize\_dependencies(self, text):

63.         """

64.         Visualize the dependencies of a text using spaCy's displacy

65.         """

66.         doc = nlp(text)

67.         displacy.render(doc, style="dep")

68.

69. # Example usage:

70. if \_\_name\_\_ == "\_\_main\_\_":

71.     text\_processor = TextProcessor()

72.     text = "The quick brown fox jumped over the lazy dog."

73.     tokens = text\_processor.tokenize\_text(text)

74.     filtered\_tokens = text\_processor.remove\_stopwords(tokens)

75.     lemmatized\_tokens = text\_processor.lemmatize\_tokens(filtered\_tokens)

76.     entities = text\_processor.perform\_named\_entity\_recognition(text)

77.     pos\_tags = text\_processor.perform\_part\_of\_speech\_tagging(text)

78.     dependencies = text\_processor.perform\_dependency\_parsing(text)

79.     text\_processor.visualize\_dependencies(text)

80.

### utils\UK\_law

##### The Code

1. import requests

2. from bs4 import BeautifulSoup

3.

4. # Define a function to search for UK Law information

5. def search\_uk\_law(query):

6.     url = "https://www.legislation.gov.uk/search"

7.     params = {"q": query}

8.     response = requests.get(url, params=params)

9.     soup = BeautifulSoup(response.content, "html.parser")

10.     results = []

11.     for result in soup.find\_all("div", class\_="result"):

12.         title = result.find("h2", class\_="title").text.strip()

13.         link = result.find("a", href=True)["href"]

14.         results.append({"title": title, "link": link})

15.     return results

16.

17. # Example usage:

18. query = "data protection"

19. results = search\_uk\_law(query)

20. for result in results:

21.     print(f"{result['title']} - {result['link']}")

22.

0\_dzBfVzmoSfyyR-c9.jpg

0\_GTgne40Gv4ryd1qZ.jpg

0\_zxjamgcHxll0nj6g (1).jpg

0\_zxjamgcHxll0nj6g.webp

## conversation\_management.py

##### The Code

1. class DialogueManager:

2.     def start\_conversation(self, jarvis):

3.         jarvis.speak("Initializing the Jaicat AI assistant...")

4.

5.     def determine\_response(self, intent, entities, allowed\_features, features, services, utilities):

6.         # Determine the appropriate response based on the intent, entities, and allowed features

7.         if intent == "generate\_code" and "Code Generation System" in allowed\_features:

8.             return features.code\_generation\_system.generate\_code(entities)

9.         elif intent == "weather" and "Weather API" in allowed\_features:

10.             return services.weather\_api.get\_weather(entities)

11.         # Add more intents and corresponding feature checks here

12.         else:

13.             return "Sorry, you don't have permission to use this feature."

14.

15. # Ensure to import necessary modules and functionalities

16.

17.

dir.bat

dirlist.txt

face\_concerned.jpeg

face\_concerned.png

face\_flirty.jpeg

face\_flirty.png

face\_happy.png

face\_neutral.jpeg

face\_neutral.png

face\_sad.jpeg

face\_sad.png

## file\_handling.py

##### The Code

1. # C:\Users\josh\_\Desktop\jaicat\_project\file\_handling.py

2.

3. import cv2

4.

5. def save\_image\_file(image, file\_name):

6.     cv2.imwrite(file\_name, image)

7.

8.

filename.txt

## image\_processing.py

##### The Code

1. # C:\Users\josh\_\Desktop\jaicat\_project\image\_processing.py

2.

3. import cv2

4.

5. def capture\_frame():

6.     # Capture a single frame from the webcam

7.     video\_capture = cv2.VideoCapture(0)

8.     ret, frame = video\_capture.read()

9.     video\_capture.release()

10.     return frame

11.

12.

layingdown.jpeg