## College of Engineering (COE)

*National Chung Cheng University*

*Report of 2024 TEEP@AsiaPlus Research Internship*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Student | Harjas Kaur | | | Faculty Mentor | | Dr. Pao-Ann Hsiung |
| Department | | | | Department: Department of Computer Science and Information Engineering | | |
| Research Period |  |  |  | Date of Report Submission | | |
| 2024/05/01 to 2024/07/16 | |  |  | 2024/07/16 | |  |
| Report Title | Human – AI Interaction + Explainability | | | | | |
| Highlights of Report | This research project investigates the hypothesis that the effectiveness of multimodal explanations in AI systems varies based on task complexity and user expertise. Specifically, it posits that an optimal combination of visual, textual, and numerical modalities enhances human understanding in classification tasks.  The study employs a survey designed to evaluate these combinations, requiring participants to complete 20 classification tasks while interacting with the AI model in real-time. The survey is hosted on Clickworker, ensuring broad accessibility. Each task utilizes different modality combinations to assess their impact on user decision-making.  The intention behind this design is to gather empirical data on Human-AI collaboration and identify the most effective explanation strategies. By analyzing participant responses, the research aims to contribute valuable insights into the development of transparent and user-friendly AI applications. | | | | | |
|  |  |  | . |  |  |
| Signature of |  | | | Date | |  |
| Student |
| Mentor Review | □ Excellent | □ Good | □ Average | | □ Poor | |
| Comment |  |  |  | |  | |
| Signature of  Mentor |  | | | Date | |  |

*College of Engineering*

National Chung Cheng University



**Trustworthy AI for Smart City Applications**

***Human- AI Interaction + Explainability***



*Harjas Kaur*

*Department of Electrical Engineering*

***Indian Institute of Technology, Ropar***

# Acknowledgement

I would like to express my heartfelt gratitude to Professor Pao-Ann Hsiung for providing us with this wonderful opportunity and for his unwavering support throughout our stay and the duration of this project. His guidance and assistance were invaluable, and his dedication greatly contributed to the progress and success of our research.

I would also like to extend my thanks to Ellen and Phoebe for their continuous help and support. Their contributions were essential in ensuring the smooth progress of our project, and their assistance in every aspect was greatly appreciated.

I would also like to acknowledge the TEEP AsiaPlus program for offering this remarkable opportunity. The experience gained through this program has been immensely valuable, enriching my understanding and skills in the field of human-AI interaction and explainability. Thank you for this incredible experience and for the chance to grow both academically and personally.

*Harjas Kaur,*

***UG Student, Department of Electrical Engineering***

***Indian Institute of Technology, Ropar***

# Table of Contents

[**Acknowledgement**](#_xy0z7pipegxe) **2**

[**Abstract**](#_5tinakjbs3nt) **3**

[**1. Introduction**](#_ke5amgsk85gj) **4**

[1.1. The Problem Statement](#_7ccer3gbprsv) 4

[1.2. Laying out the Hypothesis:](#_rbwyiof1jl31) 5

[1.3. Amazon Mechanical Turk API and Alternative: Clickworker](#_pehysrc92dox) 6

[1.4. Survey Design: Basic Layout](#_xgrsveuc53ph) 7

[1.5. FastAPI and Uvicorn](#_geuyb86olvkg) 7

[1.6. Code Structure](#_srxww4gi1k4o) 8

[Directory Structure](#_9jhwyu7pcwu) 8

[● requirements.txt: Lists all dependencies required to run the application.](#_yxx501s6wx74) 9

[Python Files](#_dxbc2domalyc) 9

[Frontend](#_yph4sem2vj7p) 9

[1.7. Setting up and running the application:](#_hvuvy6lm5prv) 10

[App: main.py](#_kl2mkgxcxhso) 10

[**2. Challenges Faced**](#_ipufyqnw3ada) **10**

[2.1. Selecting the Right Platform to Deploy the Survey](#_a8rmw4h5jbqw) 10

[2.2. Concurrent Running of Two Models](#_m27121ckftek) 11

[2.3. Learning Curves with JavaScript and Python](#_11xhtfll0j6q) 11

[**3. Impact of the work**](#_xgotggxjb737) **11**

[**4. Conclusion**](#_f1jlrw7mpeou) **12**

[**References:**](#_74piqsiopjs8) **13**

[Resources:](#_orestxwp0itb) 13

[Models: (Which we previously thought to use)](#_3o0np13b4ty8) 13

[API and Frameworks documentations:](#_cjsn46o6pipi) 13

# 

# Abstract

This research project delves into the effectiveness of multimodal explanations in enhancing human comprehension and trust in AI systems. Our objective was to develop a disaster image classification model incorporating both textual and visual modalities and to evaluate various combinations of these modalities to determine the most effective approach for human understanding. Utilizing Lime for generating explanations, we aimed to provide users with insight into the model's decision-making process.

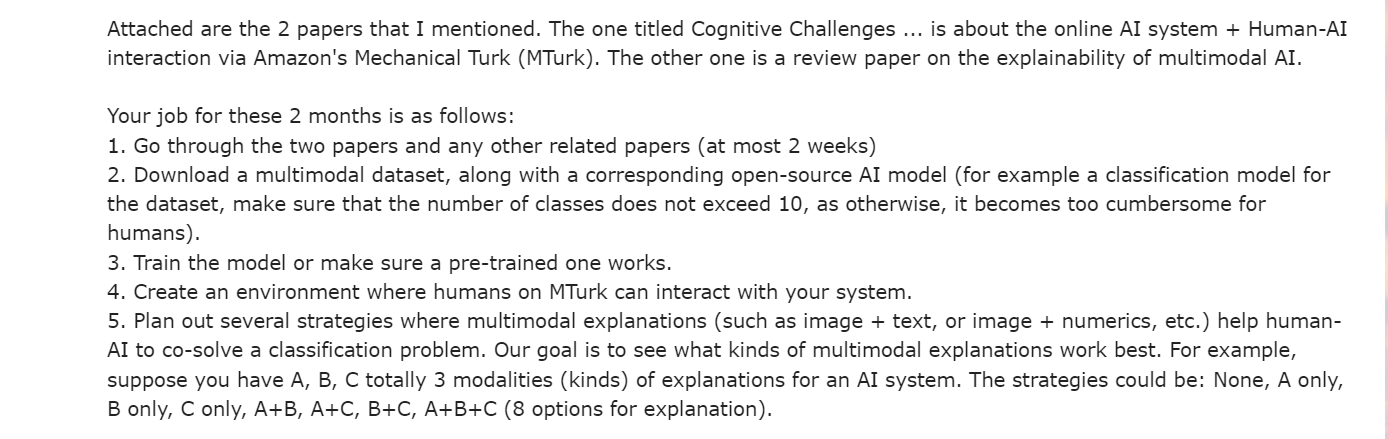
We aimed to design a comprehensive survey intended to be deployed on Clickworker, allowing participants to interact with the model in real-time. The survey, along with the model, was intended to be hosted on our servers. However, the platform is still under development, utilizing FastAPI and Uvicorn.

This work aims to contribute to the field of human-AI interaction by aiding in identifying the optimal modality combinations for explainability in AI systems, thereby advancing the design of more transparent and user-friendly AI applications. Additionally, I have proposed a hypothesis for further research, which is discussed later in the report, offering a potential direction for future investigations in this domain.

# Introduction

## The Problem Statement

The primary objective of this research is to investigate the efficacy of multimodal explanations in improving Human-AI collaboration. Specifically, we aim to understand which combinations of explanations—be they visual, textual, or numerical—most effectively enhance human users' ability to co-solve classification problems with an AI system. The complexity of multimodal data presents unique challenges in explainability, and this study seeks to identify optimal strategies for presenting AI decisions to human users.



*Fig 1: Screenshot of the exact statement given to us*

## Laying out the Hypothesis:

The effectiveness of multimodal explanations in aiding decision-making is contingent upon the complexity of the task and the expertise level of the human user. The optimal combination of modalities—visual, textual, and numerical—will vary based on

these factors. Our hypothesis is that:

The most effective multimodal explanation strategy incorporates all available modalities (image, text, and numerical data), but its success is conditioned by the specialization required for the task and the user's domain knowledge.

* *High Specialization and Low User Knowledge:* In tasks requiring high specialization where users possess limited knowledge, multimodal explanations might lead to confusion and misinterpretation. In such cases, simpler, single-modality explanations (e.g., visual only or textual only) may be more effective in aiding user understanding and decision-making.
* *Low Specialization and High User Knowledge:* For tasks with lower specialization demands, users with high domain knowledge can benefit from comprehensive multimodal explanations. The combination of image, text, and numerical data can provide a more holistic view, allowing these users to leverage their expertise to make informed decisions.
* *Balanced Tasks and Intermediate Knowledge*: For tasks of moderate complexity and users with intermediate knowledge, a balanced approach using two modalities (e.g., image + text or text + numerical) may be optimal. This strikes a balance between providing sufficient detail and avoiding information overload.

This hypothesis can be visualized in a linear graph where the x-axis represents the user's knowledge level and the y-axis represents task specialization. The ideal explanation strategy shifts from single-modality to multi-modality as user knowledge increases and task specialization decreases.

## Amazon Mechanical Turk API and Alternative: Clickworker

Amazon Mechanical Turk (MTurk) provides a robust platform for leveraging human intelligence to perform tasks that are challenging for AI. The MTurk API allows seamless integration with various applications, enabling the creation, management, and retrieval of Human Intelligence Tasks (HITs) programmatically. This API is essential for setting up large-scale experiments and collecting data efficiently from a diverse pool of participants. However, MTurk is not available in Taiwan.

To address this, we will utilize Clickworker, an alternative crowdsourcing platform that supports a wide range of tasks and has a global reach, including Taiwan. Clickworker offers similar functionalities, enabling us to distribute tasks, gather responses, and integrate the results into our system effectively. This switch ensures that our project can proceed without geographic restrictions, maintaining the integrity and scalability of our research.

Our application is hosted within an iframe to facilitate seamless integration with various web platforms and ensure a consistent user experience across different devices and browsers. This approach allows us to embed our application into existing websites or survey platforms, providing flexibility and ease of access for users.

To achieve this, our initial focus was on laying out the basic structure of the application. We ensured that the core functionalities were robust and user-friendly, setting a strong foundation for further development. This included designing a clean and intuitive user interface, establishing secure and efficient backend processes, and implementing essential features for data collection and analysis.

## Survey Design: Basic Layout

In this survey, each participant will be asked to complete 20 classification tasks. After successfully completing all tasks, a Clickworker code will be displayed, which participants can use to claim their payment. The tasks are distributed evenly across different modality combinations, with each participant receiving 5 tasks for each combination.

The survey involves real-time interaction between the participant and the AI model, providing an interactive and engaging experience. The AI model, which will assist in the classification tasks, is hosted on our own server to ensure smooth and secure operation.

## FastAPI and Uvicorn

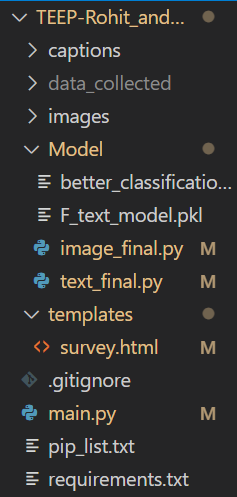
The entire application is developed in Python and is managed using FastAPI and Uvicorn. This setup allows for efficient handling of requests and responses, ensuring that the survey runs seamlessly for all participants. The application is hosted on our server, providing a reliable and controlled environment for the survey. Then data for each participant will be stored on server side, in a separate file in CSV format, to easily weed out any problematic responses by people etc.

Initially, our application was developed using Flask. However, we encountered challenges when implementing asynchronous JavaScript functions, which are essential for real-time interactions between the AI model and participants. Flask's support for asynchronous programming is limited, making it difficult to handle the asynchronous operations needed for our survey.

To overcome these challenges, we transitioned to using FastAPI, a modern web framework for building APIs with Python 3.6+ based on standard Python-type hints. FastAPI is designed to be fast, highly performant, and easy to use. It supports asynchronous programming out of the box, which allowed us to seamlessly integrate asynchronous JavaScript functions into our application. This change significantly improved the responsiveness and efficiency of our survey platform.

One of the standout features of FastAPI is its automatic endpoint testing capability. This feature allows us to quickly and efficiently debug our application, ensuring that each endpoint works correctly before deployment. The ability to test endpoints in real-time has been invaluable in maintaining the reliability and functionality of our application.

To serve our FastAPI application, we use Uvicorn, a lightning-fast ASGI server. Uvicorn is designed to handle asynchronous workloads efficiently, making it an excellent choice for our needs. It ensures that our server can handle multiple concurrent connections smoothly, providing a seamless experience for survey participants.

By using FastAPI and Uvicorn, we have created a robust and responsive application that meets the demands of our real-time interactive survey. This setup allows us to deliver a high-quality user experience and collect valuable data on human-AI collaboration effectively.

## Code Structure

### Directory Structure

**TEEP-Rohit\_and\_Harjas** (Root Directory)

* **captions/**: This directory contains caption-related .txt files. Each file with for example name image\_abc.jpeg has a corresponding caption image\_abc.txt
* **data\_collected/**: A folder designated for storing collected data during the survey.
* **images/**: Contains images used within the survey or generated by the model.
* **Model/**: Houses model files and related scripts.
  + better\_classification\_model.pkl: A serialized model file for classification tasks.
  + F\_text\_model.pkl: A serialized text model file.
  + image\_final.py: Script for running the image model.
  + text\_final.py: Script for running the text model.
* **templates/**: Contains HTML templates for the web application.
  + survey.html: The main HTML file for the survey's frontend.
* .gitignore: Specifies files and directories to be ignored by Git.
* main.py: The primary script that integrates different components and configures the endpoints for the application.
* pip\_list.txt: Lists all installed packages and their versions, useful for debugging compatibility issues.

### requirements.txt: Lists all dependencies required to run the application.

### Python Files

* image\_final.py: This script handles all operations related to the image model. It includes loading the image model, preprocessing input data, running predictions, and post-processing the results.
* text\_final.py: Similar to image\_final.py, this script deals with the text model. It encompasses loading the text model, preprocessing text data, executing predictions, and processing the output.
* main.py: This is the central script of the application. It integrates the functionalities of the image and text models and sets up the endpoints for our web application using FastAPI. It ensures that all parts of the application work together seamlessly.

### Frontend

* survey.html: This file is the frontend of our application. It is a basic survey app implementation controlled by JavaScript. The HTML layout presents the survey questions to the users and interacts with the backend to fetch predictions from the models. The survey consists of 20 classification tasks, and after completion, users receive a Clickworker code for payment.

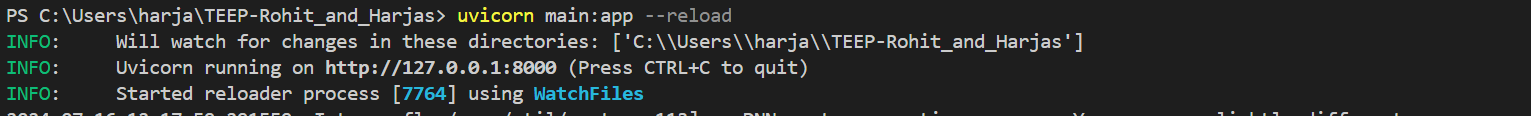
## Setting up and running the application:

* This project was developed using Anaconda with Python version 3.11.7. The choice of Anaconda provides a robust environment for managing packages and dependencies, ensuring consistency across different development setups.
* All the dependencies required for the project are listed in requirements.txt. To install them, in the parent directory, run:

pip install -r requirements.txt

* To run the survey application, navigate to the parent directory and execute the following command:

uvicorn main:app --reload

To view the survey, go to: <http://127.0.0.1:8000/survey>

To see and view the endpoints: [http://127.0.0.1:8000/](http://127.0.0.1:8000/survey)docs

### App: main.py

The code is divided into two sections, separated and labeled by comments. One is the class which initializes all the data from image.py and text.py and another is configuring the endpoints for all the elements on the survey page.

# Challenges Faced

## Selecting the Right Platform to Deploy the Survey

One of the primary challenges was selecting the appropriate platform for deploying our survey. Initially, we considered using Flask, a popular choice for web applications due to its simplicity and the abundance of online resources and tutorials. However, our project involved a multimodal model that required the concurrent running of two models. Additionally, generating explanations with LIME took time to load the images, necessitating asynchronous processing. Flask's limited support for asynchronous programming made it unsuitable for our needs. We ultimately chose FastAPI for its robust asynchronous capabilities, which allowed us to handle the concurrent operations efficiently.

## Concurrent Running of Two Models

Running two models concurrently posed significant challenges. When both models were called simultaneously, they clashed for resources on the asynchronous threading of Uvicorn. This issue remains unresolved and might stem from a coding error on my part, potentially due to declaring objects in the wrong scope. Managing the resources and ensuring smooth concurrent operations require precise and careful coding, and this has been a complex hurdle to overcome.

## Learning Curves with JavaScript and Python

My proficiency in JavaScript and Python was not very advanced at the project's outset, leading to steep learning curves. Implementing asynchronous functions, debugging, and managing server operations often left me feeling overwhelmed. There were instances when the entire application crashed, leaving me dumbfounded and requiring significant effort to diagnose and resolve the issues. Despite these difficulties, each challenge provided valuable learning experiences, gradually enhancing my skills in both languages.

# Impact of the work

The development and deployment of our multimodal survey application have yielded significant impacts across several dimensions. Firstly, by incorporating multiple modalities in the survey, it provides deeper insights into participant responses, enabling better analysis and understanding. The integration of real-time models and human interaction fosters an engaging and interactive experience for users, staying true to the theme of Human AI interaction.

Secondly, the choice of FastAPI over Flask has allowed us to leverage asynchronous processing, significantly improving the efficiency and performance of our application. This decision not only addresses the technical demands of running concurrent models but also sets a precedent for future projects requiring similar capabilities. The use of FastAPI's endpoint testing feature has streamlined our debugging process, leading to more robust and reliable code.

Navigating the complexities of asynchronous programming, concurrent model execution, and real-time interactions has expanded our technical expertise. Overcoming the challenges associated with resource management and server crashes has honed our problem-solving skills and resilience.

# Conclusion

In conclusion, the development of our multimodal survey application has been a technically demanding process, yielding incremental progress in our exploration of Human-AI collaboration. Our primary objective to investigate the efficacy of multimodal explanations in improving user decision-making remains only partially achieved.

The transition from Flask to FastAPI addressed critical limitations, particularly in handling asynchronous processing and concurrent model execution. This change has enhanced the application's performance and responsiveness, yet it has also exposed areas requiring further refinement. Significant challenges, including platform selection, concurrent model operations, and navigating the complexities of JavaScript and Python, have been confronted with varying degrees of success.

The survey design, which tasks participants with completing 20 classification challenges with real-time AI support, aims to produce initial data on human-AI interaction. Hosting the application within an iframe ensures compatibility with various web platforms, thereby enhancing accessibility. Despite these advancements, the application still necessitates minor bug fixes and additional polishing to meet deployment standards.

While our achievements are modest, the foundational work established here provides a basis for ongoing research. The journey underscores the necessity for adaptability and meticulous problem-solving in complex technical environments. Future efforts will focus on refining the application and contributing to the broader field of multimodal AI explanations and human-AI interaction.

# References:

### Resources:

<https://www.youtube.com/watch?v=3DroMVNb2vg>

<https://www.youtube.com/watch?v=VQw72XQJ8-0>

<https://www.linkedin.com/pulse/how-confident-your-ai-uncertainty-estimation-methods-ai-clearing/>

<https://openaccess.thecvf.com/content/CVPR2022/papers/Han_Multimodal_Dynamics_Dynamical_Fusion_for_Trustworthy_Multimodal_Classification_CVPR_2022_paper.pdf>

### Models: (Which we previously thought to use)

<https://github.com/maelfabien/Multimodal-Emotion-Recognition/tree/master>

<https://github.com/bharathichezhiyan/Multimodal-Meme-Classification-Identifying-Offensive-Content-in-Image-and-Text>

### API and Frameworks documentations:

#### Clickworker:

<https://support.clickworker.com/hc/en-us>

<https://www.clickworker.com/customer-support/api-documentation/>

#### MTurk:

<https://docs.aws.amazon.com/AWSMechTurk/latest/AWSMechanicalTurkRequester/>

<https://docs.aws.amazon.com/AWSMechTurk/latest/AWSMturkAPI/Welcome.html>

#### FastAPI:

<https://fastapi.tiangolo.com/>

<https://fastapi.tiangolo.com/tutorial/>

#### Flask:

<https://flask.palletsprojects.com/>

<https://flask.palletsprojects.com/en/latest/quickstart/>

#### Uvicorn:

<https://www.uvicorn.org/>

<https://github.com/encode/uvicorn>

#### Pillow:

<https://pillow.readthedocs.io/>

<https://github.com/python-pillow/Pillow>

GitHub Repository:

<https://github.com/harjas-kaur/TEEP-Rohit_and_Harjas>