PERSONALIZED DRIVING PARTNER "CHA-VIS"

Hyuna Jeon[a,*], Jeonghoon Kim[b], Subeen Kim[c]

[a] *Department of Information System, Hanyang University, younghyuna12@hanyang.ac.kr*
[b] *Department of Computer Software, Hanyang University, jk9576@hanyang.ac.kr*
[c] *Department of Information System, Hanyang University, lilyseoul@hanyang.ac.kr*
[*] corresponding author: No

Abstract. This study introduces an AI-driven navigation app incorporating emotion analysis for enhanced driver support. Speech input is converted to text using Whisper, and emotional tones are analyzed with HuBERT. A GPT-4o-mini model classifies the input into three categories: search queries, route requests, or casual conversation. Search queries are addressed using Google Search, and route requests integrate POI suggestions like restaurants or rest stops using Naver Map API. For casual conversation, the app tailors responses and suggestions, such as calming music or supportive messages, based on the driver's emotional state. This creates an empathetic and adaptive driving experience.

Keywords: Emotional analysis, Query classification, Large language model.

Role	Name	Role Description
SW Developer	Kim Jeong Hoon	Implementing navigation as a web, app. Combining learned models with the app.
AI Developer	Jeon Hyuna	Finding the right model. Training new data to existing models. Connecting a prompt to the model.
AI Developer	Kim Su Been	Providing Ideas. Finding data for model training, preprocesses. Writng prompts for voice classification.

Table 1. Task Description

1. Introduction

1.1. Motivation

Driving alone can lead to accidents due to drowsiness, which can be very dangerous. In addition, there are more and more incidents of people losing control of their emotions while driving and turning small problems into retaliatory driving, assaults, etc. Therefore, in order to solve these problems, we developed a real-time interactive navigation system that helps drivers control their emotions by analyzing their emotions through their voice.

1.2. Summary and Goals

The goal is to develop real-time interactive navigation that helps drivers control their emotions. It adds emotional control through emotion analysis of the user's voice to the basic function of navigation, which is to provide directions. In addition, the navigation doesn't just use one tone of voice, but responds with different voice tones depending on the driver's emotions.

1.3. Feature Introduction

1) Basic conversations and directions

It provides basic conversation functions and the same directions as conventional navigation.

2) Real-time data collection and analysis

Using the SK network, information on traffic, weather, and local events on the route within the route is collected and analyzed in real time. It listens to the user's utterance, categorizes it into basic conversation and search functions, and if a search is needed, real-time information is reflected using Google Search.

3) Emotion recognition features

We proposes a real-time navigation system that leverages multidimensional emotion analysis to enhance driver safety and experience. The system conducts analysis by using a HuBert-based model. Emotional states are detected through changes in vocal tone, speech rate, and volume. It predicts the emotional category and its intensity directly from the audio input. Textual data from speech-to-text (STT) conversion, conducted using Whisper from OpenAI, is analyzed for emotional cues based on word choice, sentence structure, and linguistic features.

Based on the identified emotional state, the system delivers AI-driven emotional coaching interventions tailored to the driver's current condition. For instance, if irritation or agitation is detected, the system responds by playing calming music, and engaging the driver in a soothing conversational tone. These measures aim to reduce stress and minimize accident risk.

In implementation, the HuBERT model and Whisper STT system operate in parallel. The HuBERT model processes raw audio to predict emotions and their intensity, while Whisper transcribes the speech to text. Both the detected emotions and transcribed text are passed to a large language model (LLM), which generates responses considering the driver's emotional state and context. The LLM facilitates adaptive and empathetic interactions, ensuring that the driver feels supported and understood.

This system represents a novel approach to integrating emotion-aware AI technologies into real-time navigation, prioritizing both safety and emotional well-being.

2. Development environment

2.1. Language / Framework

1) Language

(1) Python

Figure 1. Python

Python is a widely used programming language for web applications, software development, data science, and machine learning (ML). Developers use Python because it is efficient, easy to learn, and can run on multiple platforms. We used Python in the process of preprocessing data, and modeling.

(2) Dart

Figure 2. Enter Caption

The Dart language is an object-oriented programming language created by Google. When you're developing it, you can compile it in the JIT method, so you can see the results of compiling and changing it at a very high speed. We used it when we were developing Flutter.

2) Framework

(1) Fast API

FastAPI is a Python framework with high performance. It guarantees fast performance with asynchronous behavior. We configured fastAPI to act as a backend to process the data sent by the flutter app and return the required data.

Figure 3. FastAPI

(2) OpenAI API

Figure 4. OpenAI API

The OpenAI API offers a variety of features, allowing it to perform a variety of tasks, including text generation, image processing, and speech recognition. We used the openAI API for inference.

(3) Flutter

Figure 5. Flutter

It is a cross-platform app development framework developed by Google. Based on a single code base, it is possible to develop not only Android and iOS, but also PCs and the web. As developed by Google, the language you speak is also powered by the Google-made 'Dart' language. It has three characteristics: significantly fast, productive, and flexible.

(4) LangChain

LangChain is a framework designed to make it easier to build applications using language models (LLM). It provides the necessary components for developing LLM-based applications, such as data connection, memory management, and integration with various tools. This enables efficient development of various AI applications such as chatbots, search systems, and automation workflows.

3) Models

(1) Whisper

The whisper model is a typical ASR model. It supports not only English but also various languages and has a considerable recognition performance, so it has the advantage of being able to easily process natural language. In addition, depending on the system to utilize the speech recognition model, there is also the advantage of being able to select the model Tiny, Small, medium, large, and large-v2.

(2) HuBert

It is a deep learning model used for speech recognition and processing, developed by Facebook AI. It learns speech data to extract acoustic features in a self-supervised learning way, and learns speech representation effectively without text labels. This shows excellent performance in various tasks such as speech recognition, speech synthesis, and speech understanding.

HuBERT employs a methodology where input speech data is clustered into hidden units, and the model is trained to predict masked regions of the input. This approach allows it to effectively learn structural features of speech.

`https://huggingface.co/team-lucid/hubert-base-korean`

(3) TTS

Text-to-Speech is an abbreviation for text-to-speech, which refers to a technology that converts text into voice. This technology means that a computer can read text and output voice in the same voice as a person. It is implemented using various technologies such as artificial intelligence, voice synthesis, and neural network technology. Typically, it is used as an auxiliary technology for the visually impaired or the low-sighted. It is also used in fields such as voice guidance systems, automatic response systems, and voice mail.

`https://huggingface.co/facebook/mms-tts-kor`

4) Others

(1) FlashAttention

FlashAttention rearranges the Attention calculation and utilizes tiling and recalculation to significantly increase speed and reduce memory usage. Load the input block via tiling, perform an Attention on that block, and update the output. Reduce the amount of memory read/write by not writing the intermediate Attention matrix to memory.

(2) Notion

Figure 6. Enter Caption

Notion is project management and recording software. It is designed to help members of the company and adjust deadlines, goals, and tasks to increase the efficiency and productivity of the organization. We used the Team Notice page to manage the project schedule and organize and share what needs to be done.

(3) GitHub

Figure 7. Enter Caption

It is a Git platform under Microsoft. It has emerged as a sacred place for free software and open source thanks to developer-friendly policies such as developer conferences, communities, project sharing, and Git hosting functions.

(4) Overleaf

Figure 8. Enter Caption

Overleaf is a collaborative cloud-based LaTeX editor used for writing, editing and publishing scientific documents.

3. Datasets

3.1. For HuBert

1) Dataset

We utilized the "Conversational Speech Dataset for Sentiment Classification" found on AI Hub. The data was collected using an emotional dialog application, where users spontaneously interacted with the application over a period of time, and the data was cleaned and curated. Seven emotions (happiness, anger, disgust, fear, neutral, sadness, surprise) were labeled by five people.

2) Data Preprocessing

We combined all three datasets together. Since the data was labeled by a total of 5 people, we created a new column label whose values were the modes of the 5 emotion labels, and a new column intensity whose values were the modes of the 5 intensities. In order to train Hubert, all remaining columns were deleted except for the four columns : wav_id, which is the name of the audio file, 'utterance' which converts the sound into text, label, and intensity. Lastly, to make the number of data per emotion similar, we randomized 1000 emotions per label, and if there are fewer than 1000 emotions, we extracted all of them.

3.2. For TTS

1) Dataset

We utilized the 'Emotional speech synthesis dataset' we found on AI Hub. This dataset was performed by a female voice actor in her 30s on each of the seven emotions for 3,000 speech recordings, and there are a total of 21,000 speech files.

2) Data Preprocessing

We used three categories of neutrality, positivity, and negativity among seven emotions, and extracted 1000 pieces of data for each category and used them for model training.

4. Methodology

4.1. For the Entire Process

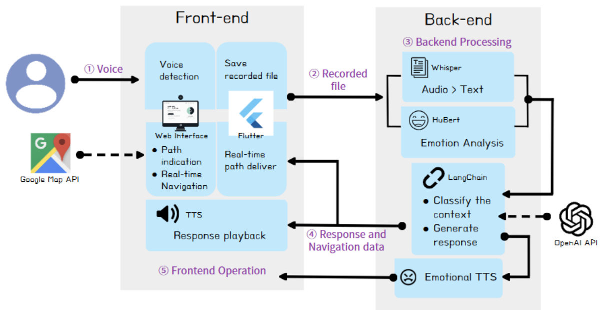

Figure 9. Architecture Full

The user uses Flutter to save the audio input to the recording file and deliver it to the backend. At the backend, FastAPI converts the audio into text with Whisper, HuBERT is used to analyze emotions, and LangChain and OpenAI APIs are used to classify the context and generate appropriate responses. The generated response and path data using the Google Maps API are returned to the front end, which Flutter uses for real-time path guidance. At the same time, voice is generated at the back end if emotional answers are required, or at the front end via TTS.

4.2. For the AI Process

When audio comes on, HuBert and Whisper two models work. First, Whisper converts speech into text and classifies whether the voice data entered through the openAI prompt is a daily conversation, a query that needs to be searched, or a query that needs to be navigated to produce an answer accordingly. If it is a query that requires navigation, extract up to two keywords together and function with the Google Map API. If it is a query that requires a search, extract

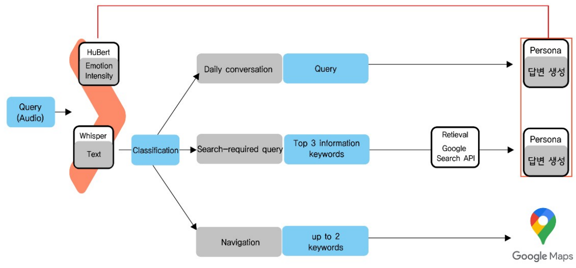

Figure 10. Architecture AI

three keywords needed for the search together. And generate the results of the search with the Google Search API as answers. In particular, those that are classified as queries that require daily conversations and search apply the speech sentiment analyzed by HuBert to produce an appropriate answer when generating an answer.

5. Hidden-Unit BERT (HuBert)

5.1. Algorithm Selection

The choice of algorithms and frameworks in this project is designed to optimize emotion recognition from audio data in real-time. The HuBERT-based model is utilized for its efficiency in processing audio data, leveraging its pre-trained knowledge on speech features. This model is fine-tuned to predict both the emotional label (classification) and its intensity (regression) directly from raw audio. Additionally, OpenAI Whisper serves as the speech-to-text (STT) engine to generate transcriptions, enabling linguistic analysis for improved contextual understanding.

The rationale behind using HuBERT and Whisper in parallel is to enhance multimodal emotion recognition. By integrating auditory and linguistic information, the system achieves a more comprehensive and accurate understanding of the user's emotional state. A Stratified K-Fold Cross-Validation approach is applied to ensure robust model evaluation across various subsets of the dataset, mitigating overfitting and improving generalization.

5.2. Dataset Preprocessing

The metadata is cleaned by standardizing emotion labels (e.g., lowercase conversion, whitespace removal) and mapping them to numerical identifiers based on a predefined emotion mapping dictionary. Each audio file path is programmatically generated based on the wav_id and the dataset's directory structure. Files not found in the directory are excluded from training. Audio files are loaded using "Librosa" with a consistent sampling rate of 16,000 Hz. This ensures uniformity across all inputs, critical for deep learning models.

5.3. Model Architecture and Training Strategy

The proposed model is built on a pre-trained HuBERT model designed for sequence classification, with additional enhancements tailored for the task. Specifically, a classifier head is incorporated to predict emotion labels, while a regression head is added to estimate the intensity of emotions. The architecture includes dropout mechanisms to mitigate overfitting, and the configuration has been fine-tuned to achieve optimal performance. Furthermore, the hidden layers of the pre-trained HuBERT model have been reinitialized to facilitate robust feature extraction and adaptation to the emotion recognition task.

The training process follows a well-defined pipeline to ensure reliability and generalizability. A Stratified K-Fold Cross-Validation approach is utilized to divide the dataset into 20 folds, maintaining an equal distribution of emotion classes across training and validation subsets. This methodology enhances the robustness of the model and minimizes potential biases. Custom PyTorch Dataset and DataLoader classes are implemented to manage audio feature extraction, padding, and batching, while collation functions ensure efficient processing of variable-length audio inputs.

For the loss functions, Cross-Entropy Loss is used for emotion label classification, while Mean Squared Error (MSE) Loss is employed for emotion intensity regression. The optimization process is carried out using the AdamW optimizer with a layer-wise learning rate decay (LLRD) strategy, enabling balanced weight updates across both shallow and deep layers of the model. The training strategy leverages PyTorch Lightning for efficient multi-GPU training, with mixed precision (16-bit) utilized to improve training speed and memory efficiency. Additionally, a checkpointing mechanism monitors validation accuracy for emotion label classification, saving the best-performing model for each fold to ensure optimal performance during evaluation and deployment.

5.4. Implementation Details

1) Custom Dataset Class

The dataset class handles loading audio data, applying the HuBERT feature extractor to convert raw speech into embeddings, and storing emotion labels and intensity scores.

2) Custom Lightning Module

The Lightning Module defines a forward pass that outputs both emotion label logits and intensity predictions, a training step that computes the combined loss and logs relevant metrics (e.g., accuracy, intensity loss), and a validation step to evaluate model performance on unseen data.

3) Hyperparameters

Batch Size: 8 samples per batch.
Learning Rate: 1e-5 with a decay factor of 0.8 for LLRD.
Dropout Rate: 0.07 across hidden layers to reduce overfitting.
Max Epochs: 30 with early stopping based on validation accuracy.

4) Environment and Time

Model training was conducted in A 100 80GB environment, and it took about 30 hours.

5.5. Difference Point

Compared to other models like wav2vec 2.0, HuBERT demonstrates a significant advantage by achieving high performance even with relatively small amounts of labeled data. However, unlike the conventional self-supervised approach of HuBERT, We adopted a supervised learning approach by fine-tuning the model using labeled data to predict both emotion labels and intensity levels.

This approach differentiates itself from the traditional applications of HuBERT by enabling the simultaneous learning of emotion classification and regression tasks related to emotion intensity. As a result, We developed a model capable of quantitatively analyzing the intensity of emotional states in addition to classifying emotion labels, setting it apart from previous research that primarily focused on simple classification. Moreover, this research fine-tunes HuBERT using speech data specifically collected from drivers, creating a domain-specific model optimized for emotion recognition in driving scenarios. By doing so, the study aims to develop a system capable of analyzing drivers' emotions in real time and providing feedback based on the detected emotional states. This represents a novel application that has not been explored in existing HuBERT-based emotion recognition studies, highlighting its practical implications and innovation.

Finally, to enhance the acoustic processing efficiency of our navigation system, we strategically implemented FlashAttention—a cutting-edge deep learning optimization technique—into the HuBert model's inference pipeline. This innovation significantly accelerates computational performance, enabling a more responsive and seamless voice interaction experience for our users.

5.6. HuggingFace Link

`https://huggingface.co/HyunaZ/hubert_emotion`

6. Text-To-Speech (TTS)

6.1. Model Architecture and Training Strategy

The emotional TTS training code is built using PyTorch and incorporates Hugging Face's VITS model to convert text into speech, enhanced with emotional expression capabilities. The dataset consists of emotion-labeled audio files, which are processed using the `EmotionalVitsDataset` class to prepare text and emotion data. The pretrained VITS model's weights remain frozen, while only the additional emotion embedding (`nn.Embedding`) and projection layers (`nn.Sequential`) are trainable. Based on the input text and emotion data, the VITS model generates the base speech waveform, and an emotion modulation factor, derived from the emotion embedding, is applied to adjust the waveform for emotional expression.

During training, the dataset is loaded in batches using `DataLoader`, and the L1 loss function (`F.l1_loss`) is employed to measure the difference between the generated and target waveforms. The `AdamW` optimizer updates the weights of the emotion-related layers, and the average loss is printed for each epoch to monitor training progress. At the end of each epoch, a checkpoint is saved, containing the model's state and loss value, ensuring recoverability in case of interruptions. This approach focuses the training on components related to emotional representation.

6.2. Implementation Details

1) Custom Dataset Class

The dataset class prepares the data by tokenizing a fixed text input using a Hugging Face tokenizer and processing audio files stored under emotion-labeled directories. Audio files are loaded, normalized, and padded or trimmed to a fixed segment length suitable for model input. Each sample consists of text input, audio, and corresponding emotion labels.

2) Custom Model Architecture

The model extends the pretrained VITS architecture, freezing its original parameters to preserve the base TTS capabilities. A new emotional embedding layer and a projection network are added to modulate the generated speech waveform based on emotion input. During the forward pass, the model applies this emotional modulation to the base waveform to incorporate the desired emotional characteristics.

3) Hyperparameters

Batch Size: 128 samples per batch. Learning Rate: 1e-4 using the AdamW optimizer for the emotional layers. Dropout Rate: No additional dropout is applied beyond the base model. Max Epochs: 30 with checkpointing after each epoch.

4) Environment and Time

Model training was conducted in A 100 80GB environment, and it took about 6 hours.

5) Training Strategy

After an intensive training process spanning 30 epochs, we meticulously refined our Text-to-Speech model, ultimately selecting the 25th epoch as our optimal performance milestone.

6.3. Difference Point

An emotion embedding layer was added to the VITS model pre-trained in Korean to enable speech reflecting emotions. Among the utterances performed by a voice actor on seven emotions, we added layers using a dataset of 1000 positive, neutral, and negative emotions each. We learned in the direction of reducing the loss between the actual target audio and the predicted waveform, and only updated the emotion layer with the existing pre-learning parameters fixed, improving only the ability to express emotions without changing the existing Korean language ability.

6.4. HuggingFace Link

`https://huggingface.co/HyunaZ/vits-kor-emotion/blob/main/README.md`

7. Classification Model

For efficient interaction with navigation while driving, we designed a dialogue classification system using the OpenAI API. The system classifies the request intention into three categories: Route, Search, and Conversation based on the user's voice input. If they are classified as Search, they are required to print up to three search keywords. The prompts we designed for this defined contextual examples and conditions in detail so that the model could understand the intentions accurately. For example, inputs such as "Let's go to Hanyang University" are classified as Route, and "Play songs that match the weather today" is processed as Search, weather, and music.

The system is based on OpenAI's gpt-4o-mini model, and a prompt template was constructed using the LangChain library to categorize user questions. Input messages are sent to the model using LangChain's ChatPromptTemplate, which integrates system messages with user questions. This allows the AI to set navigation paths, search for appropriate information, or continue a natural conversation based on driver requests.

8. Code for the Navigation

8.1. General Purpose of the Code

The provided JavaScript/HTML implementations are built on top of the SK T-Map API. They aim to achieve three main objectives:

1) Route Planning with Traffic Information: Compute optimal navigation routes between a start and end location, factoring in traffic data, travel time, and other preferences.

2) Point of Interest (POI) Search: Allow users to search for locations by name or keyword (e.g., restaurants, landmarks) and display results as markers on the map.

3) Road Matching API Integration: Match a set of geographic coordinates to their corresponding road segments, ensuring accurate mapping for driving paths or analytics.

8.2. Map Initialization and Configuration

T-Map API is initialized to render a base map centered at specific coordinates. Interactive features are configured such as zoom controls, scroll-wheel support, and default zoom levels. Base markers are Optionally added for predefined locations (start and end points).

This step sets up the interactive map, allowing it to serve as the foundation for route planning and data visualization.

Listing 1. Map Initialization

```
function initTmap() {
    map = new Tmapv2.Map("map_div", {
        center: new Tmapv2.LatLng(),
        width: "70%",
        height: "400px",
        zoom: 17,
        zoomControl: true,
        scrollwheel: true
    });
}
```

8.3. Route Calculation and Display

- Send user-defined start and end coordinates to the T-Map Routing API.
- Retrieve the computed route along with additional data:
    - ▷ Distance (in km)
    - ▷ Travel time (in minutes)
    - ▷ Traffic conditions
- Parse the API response to extract line segments (geometry) and draw them on the map.
- Dynamically color-code route segments based on traffic congestion levels (e.g., red for heavy congestion).

Purpose: This feature calculates the most efficient driving route, helping users plan trips while accounting for real-time traffic conditions.

Listing 2. Route Calculation

```
$.ajax({
    type: "POST",
    headers: { "appKey": "YourAppKey" },
    url: "https://apis.openapi.sk.com/..",
    data: {
        startX: "126.9850380932383",
        startY: "37.566567545861645",
        endX: "127.10331814639885",
        endY: "37.403049076341794",
        reqCoordType: "WGS84GEO",
        resCoordType: "EPSG3857",
        trafficInfo: "Y"
    },
    success: function(response) {
        const resultData
            = response.features;
        drawRoute(resultData);
// Function to draw route lines
    }
});
```

8.4. POI Search and Marker Placement

- Accept a user-input keyword (e.g., "cafe") and send it to the T-Map POI Search API.
- Extract relevant POIs (Point of Interest) from the response, including:
    - ▷ Name of the location
    - ▷ Geographic coordinates (latitude, longitude)
- Plot each POI as a unique marker on the map, ensuring clear visualization.
- Optionally pan and zoom the map to encompass all search results.

Purpose: This feature supports location discovery, enabling users to search for destinations or nearby facilities conveniently.

Listing 3. POI Search

```
$.ajax({
    method: "GET",
    headers: { "appKey": "YourAppKey" },
    url: "https://apis.openapi.sk.com/..",
    data: {
        searchKeyword: "cafe",
        resCoordType: "EPSG3857",
        reqCoordType: "WGS84GEO",
        count: 10
    },
    success: function(response) {
        const pois
            = response
                .searchPoiInfo
                    .pois.poi;
        pois.forEach(poi => {
            const lat = parseFloat(
                poi.noorLat);
```

```
            const lon = parseFloat(
                poi.noorLon);
            addMarker(lat, lon, poi.name);
            // Adds markers for each POI
        });
    }
});
```

8.5. Road Matching for Geospatial Data

- Accept a list of geographic coordinates representing a user's travel path.
- Split the data into chunks (max 100 coordinates per request) to comply with API limitations.
- Send each chunk to the Road Matching API, which aligns the input coordinates to their closest road segments.
- Draw corrected paths on the map and calculate total matched distance.

This feature ensures the accuracy of geographic data by aligning raw GPS coordinates to roads. It is crucial for applications like fleet tracking, driving analytics, and geospatial corrections.

Listing 4. Road Matching API

```
$.ajax({
    type: "POST",
    url: "https://apis.openapi.sk.com/..",
    data: {
        coords:
            "126.977945,37.566309
␣␣␣␣␣␣␣␣␣␣␣␣|126.978782,37.567041",
            // Sample data
        responseType: 1
    },
    success: function(data) {
        const matchedPoints
            = data
                .resultData
                    .matchedPoints;
        drawMatchedRoute(
            matchedPoints);
            // Function to
            //draw matched paths
    }
});
```

8.6. Summary of Code

The code is designed for the development of real-time, emotion-aware navigation systems, as outlined in the abstract. Specifically, it:

- **Enhances Driver Interaction:** Provides dynamic and user-friendly features such as route planning and POI discovery, which are essential for interactive navigation.
- **Incorporates Real-Time Data:** Leverages SK's network to integrate traffic, weather, and local events, delivering up-to-date information.
- **Prepares Data for Emotion-Aware Features:** The Road Matching API helps preprocess data for emotion recognition. For example, accurate road matching can ensure reliable emotion-triggered responses, such as calming music during stressful traffic.

This modular codebase forms the backbone of a more complex navigation system. By combining real-time routing, POI search, and road matching, it provides an excellent framework for personalized driving assistants.

9. Conclusion

In conclusion, this study successfully developed an emotion recognition system leveraging a fine-tuned HuBERT model to analyze both emotion labels and their intensity levels. By utilizing labeled data, the model was optimized for domain-specific tasks, such as emotion detection in driving scenarios, ensuring real-time adaptability and high accuracy. The integration of voice analysis and emotional intensity prediction demonstrated the potential for creating intelligent systems that can enhance user safety and experience, particularly in emotionally sensitive contexts like driving.

This work contributes to the field of emotion recognition by extending the conventional applications of HuBERT to include emotion intensity regression, a less explored dimension in existing research. Moreover, the use of driving-specific datasets highlights the importance of domain-specific optimization, providing insights into how emotion-aware systems can be tailored for specialized environments.

While this study focused on speech-based emotion recognition, the integration of multimodal inputs, such as facial expressions or physiological signals, could further improve the system's accuracy and robustness. Future research could explore the extension of this approach to multimodal emotion recognition systems, incorporating textual, visual, and physiological data to enhance performance. Furthermore, investigating adaptive learning techniques to handle domain shifts or user-specific variations could make the system even more flexible and robust. This study lays a solid foundation for the development of intelligent, emotion-aware systems with wide-ranging applications, particularly in real-time safety-critical environments.

10. References

[1] Figure 1 : https://en.wikipedia.org/wiki/Python_%28programming_language%29
[2] Figure 2 : https://www.google.com/url?sa=i&

url=https%3A%2F%2Fdart-ko.dev%2F&psig=AOvVaw2rgWZcltWj5ti_PkSN2YaC&ust=1733903312020000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCODt26nbnIoDFQAAAAAdAAAAABAE
[3] Figure 3 : https://fastapi.tiangolo.com/ko/
[4] Figure 4 : https://www.linkedin.com/pulse/getting-started-openai-api-mourtaza-fazlehoussen-obumf/
[5] Figure 5 : https://flutter.dev/
[6] Figure 6 : https://ko.wikipedia.org/wiki/%EB%85%B8%EC%85%98_(%EC%83%9D%EC%82%B0%EC%84%B1_%EC%86%8C%ED%94%84%ED%8A%B8%EC%9B%A8%EC%96%B4)
[7] Figure 7 :https://ko.wikipedia.org/wiki/%EA%B9%83%ED%97%88%EB%B8%8C
[8] Fiture 8 : https://ko.overleaf.com/for/partners/logos