UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Diseño y Fusión de Algoritmos para la inferencia del estado del usuario en escenarios de colaboración Humano-Máquina

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Carbonell Sales, Carles

Tutor/a: Fons Cors, Joan Josep

Director/a Experimental: Mestre Gascón, Antoni

CURSO ACADÉMICO: 2023/2024

Agradecimientos

En primer lugar, quiero expresar mi profundo agradecimiento a Antoni Mestre, Manoli Albert y Joan Josep Fons del equipo de VRAIN por confiar en mí, contar conmigo y proporcionarme todos los recursos necesarios, así como compartir su invaluable experiencia.

Quiero dedicar una mención especial a Antoni Mestre, quien me ha acompañado desde antes de iniciar mi carrera y ha sido mi guía constante hasta el final. Su apoyo y la oportunidad de participar en numerosos proyectos han sido fundamentales para mi crecimiento profesional y personal.

Además, quiero agradecer a mis amigos y compañeros de carrera por su compañía y ayuda a lo largo de estos años.

Por último, quiero expresar mi más sincera gratitud a mi familia y a mi novia por su inmensa paciencia, cariño y apoyo inquebrantable.

A todas las personas que han sido parte de este viaje, gracias de todo corazón.

Resumen

El objetivo de este TFG es diseñar e implementar un modelo en el ámbito de la interacción humano-máquina (HCI) eficiente, modular y escalable para detectar, a través de diferentes tipos de datos multimodales como grabaciones de audio o vídeo, la necesidad de un usuario de retroalimentación o interacción de la máquina. Este modelo se basa en una arquitectura de tres capas de comunicación ascendente, donde la capa inferior está constituida por una serie de módulos completamente personalizables, encargados de obtener y tratar todo tipo de datos multimodales, con una capa intermedia encargada del control de la ejecución del programa y de la unificación de los datos, y con una última capa compuesta por una IA capaz de predecir la necesidad de interacción humano-máquina mediante la inferencia de la unión de dichos datos. La implementación de este modelo estará compuesta de tres módulos de datos; audio, video y reloj inteligente, que junto al uso de técnicas y tecnologías de proceso de vanguardia permitirán entrenar la última capa del modelo en un caso de uso concreto. Finalmente, se pondrá a prueba y se calculará el rendimiento de este modelo.

Palabras clave: IA híbrida, Human In the Loop, IA multimodal, Procesamiento del Lenguaje Natural, Análisis de sentimientos.

Abstract

The objective of this TFG is to design and implement an efficient, modular and scalable model in the field of human-machine interaction (HCI) to detect, through different types of multimodal data such as audio or video recordings, a user's need for feedback or machine interaction. This model is based on a three-layer architecture of bottom-up communication, where the lower layer is constituted by a series of fully customizable modules, in charge of obtaining and processing all types of multimodal data, with an intermediate layer in charge of program execution control and data unification, and with a last layer composed of an AI capable of predicting the need for human-machine interaction by inferring from the union of such data. The implementation of this model will be composed of three data modules; audio, video and smart watch, which together with the use of state-of-the-art processing techniques and technologies will allow training the last layer of the model on a specific use case. Finally, the performance of this model will be tested and calculated.

Keywords: Hybrid AI, multimodal AI, Human In the Lopp, Natura Language Processing, Sentiment analysis.

Resum

L'objectiu d'este TFG és dissenyar i implementar un model en l'àmbit de la interacció humà-màquina (HCI) eficient, modular i escalable per a detectar, a través de diferents tipus de dades multimodals com gravacions d'àudio o vídeo, la necessitat d'un usuari de retroalimentació o interacció de la màquina. Este model es basa en una arquitectura de tres capes de comunicació ascendent, on la capa inferior està constituïda per una sèrie de mòduls completament personalitzables, encarregats d'obtindre i tractar tot tipus de dades multimodals, amb una capa intermèdia encarregada del control de l'execució del programa i de la unificació de les dades, i amb una última capa composta per una IA capaç de predir la necessitat d'interacció humà-màquina mitjançant la inferència de la unió d'estes dades. La implementació d'este model estarà composta de tres mòduls de dades; àudio, vídeo i rellotge intel · ligent, que al junt amb l'ús de tècniques i tecnologies de procés d'avantguarda permetran entrenar l'última capa del model en un cas d'ús concret. Finalment, es posarà a prova i es calcularà el rendiment d'este model.

Paraules clau: IA híbrida, Human In the Loop, IA multimodal, Processament del Llenguatge Natural, Anàlisi de sentiments.

Tabla de contenidos

Índice de Figuras

Índice de Gráficos

Índice de Tablas

Índice de Código

Introducción

1.1. Motivación

La tecnología de la IA evoluciona rápidamente y se está integrando cada vez en más sistemas y ámbitos, tanto profesionales como en la vida cotidiana de las personas [1]. Más concretamente, gracias a la creación de servicios y herramientas como Chat GPT, Copilot o DALL-E, han hecho que la IA sea accesible y popular entre el público en general, promoviendo su uso e investigación en todo tipo de áreas [2]. Sin embargo, la IA aún no ha llegado a fusionarse completamente con el ser humano, permaneciendo como una simple herramienta cada vez más accesible al público. Por esta razón, se ha decidido realizar una investigación en el campo de la IA híbrida, que busca crear una simbiosis entre la máquina y el ser humano [3], considerando su estado y sus acciones; creando de esta forma una interacción más sencilla, directa y natural humano-máquina. Específicamente, se pretende avanzar en la detección de la incertidumbre y necesidades del usuario mediante la captación de diferentes datos multimodales.

1.2. Objetivos

El objetivo principal del proyecto consiste en diseñar e implementar un sistema modular, optimizado y escalable para detectar, a través de diferentes tipos de datos multimodales, la necesidad de un usuario de retroalimentación o interacción de la máquina. Internamente, este se puede dividir en los siguientes objetivos secundarios o parciales necesarios para su correcto planteamiento y posterior desarrollo:

- Investigación de las tecnologías e implementaciones actuales de la inteligencia artificial híbrida.
- Definición de un modelo teórico simple, general, fácilmente ampliable y escalable.
- Planteamiento y desarrollo de un módulo de audio.
- Planteamiento y desarrollo de un módulo de vídeo.
- Planteamiento y desarrollo de un módulo de *smartwatch*.
- Creación de un sistema universal de unión de módulos.
- Concepción de un modelo de inteligencia artificial de detección de necesidades.

1.3. Impacto esperado

Se espera que la investigación y desarrollo de este proyecto mejore significativamente la interacción general humana con los dispositivos inteligentes, facilitando y aproximando al usuario una interacción más natural y personalizada que la ofrecida por la visión actual de la IA. Este avance podría resultar de gran interés para los diferentes desarrolladores de entornos y servicios, dado que permitiría una interacción más centrada en el cliente y en sus necesidades inmediatas.

Algunos ejemplos de posibles implementaciones mutuamente beneficiosas y de gran impacto pueden ser la detección de malestares o ansiedad, la ayuda en la compresión en todo tipo de situaciones o la adaptación de diferentes sistemas según el estado del usuario.

1.4. Metodología

Para el correcto avance y futura completitud del proyecto, primeramente, se debe diseñar un modelo teórico simple capaz de soportar una gran variedad de datos multimodales, siendo fácilmente adaptable y escalable, y ofreciendo un alto rendimiento y fiabilidad. Para ello, se ha divido en diferentes módulos y sistemas de diseño e implementación más sencilla, donde se ha llevado a cabo una exhaustiva investigación y desarrollo de las diferentes tecnologías punteras necesarias. Finalmente, se ha realizado una unión y prueba de dichos módulos para comprobar su viabilidad y desempeño ante usuarios reales.

1.5. Estructura

La primera sección está formada por la introducción y el estado del arte, donde se justifica con detalle la motivación que ha llevado a considerar necesaria la existencia del proyecto y un análisis del estado en el cual se han cimentado las diferentes tecnologías y conceptos relacionados con el mismo.

La segunda sección se compone por una definición exhaustiva del modelo teórico del sistema a implementar, las partes que lo componen y su razón de ser.

La tercera sección engloba el diseño y desarrollo de algunos de los diferentes posibles módulos de captación y tratamiento de datos para el sistema y su posterior unión.

La cuarta sección trata la creación del modelo de inteligencia artificial que clasificará la necesidad de interacción de la máquina con el usuario y sus diferentes pruebas y resultados obtenidos.

Finalmente, la quinta sección está compuesta por las conclusiones del proyecto de investigación, la relación del trabajo desarrollado con los estudios cursados y las posibles ampliaciones futuras sobre el mismo.

2. Estado del arte

La identificación de las necesidades humanas a través de la inteligencia artificial (IA) cubre una amplia gama de aplicaciones avanzadas que están transformando todos los campos. Estos avances se basan en tecnologías como el aprendizaje profundo, el análisis de sentimientos, los sistemas de recomendación, el uso de datos de sensores y dispositivos IoT, modelos predictivos y de segmentación e interacciones multimodales.

El aprendizaje profundo, representado por modelos como las redes neuronales convolucionales (CNN) y las redes neuronales recurrentes (RNN), han revolucionado la capacidad de la IA para procesar y comprender grandes cantidades de datos. Algunos ejemplos populares de modelos de lenguaje como BERT y GPT, permiten a la IA comprender y generar texto de manera consistente, facilitando de esta forma la identificación exacta de las necesidades expresadas en lenguaje natural.

Por otra parte, el análisis de sentimientos es un área importante del procesamiento del lenguaje natural (NLP), que permite a la IA interpretar emociones y sentimientos del texto. Esta capacidad no solo permite comprender mejor las necesidades emocionales y psicológicas de los usuarios, sino que también permite generar respuestas más empáticas y efectivas en aplicaciones como el servicio al cliente y la atención personalizada.

Otros avances en la IA basada en la mejora de la interacción máquina-humano tratan de integrar datos de sensores y dispositivos IoT que monitorean la condición física y el estado de salud de un usuario en tiempo real. Dispositivos como los relojes inteligentes recopilan datos sobre la salud cardiovascular, los patrones de sueño y la actividad física, proporcionando información valiosa para la detección temprana de problemas de salud y promoviendo un estilo de vida saludable.

Por otro lado, los modelos predictivos utilizan técnicas avanzadas para analizar patrones en grandes conjuntos de datos permitiendo, de esta forma, predecir necesidades futuras y ajustar su respuesta en consecuencia. Por ejemplo, en el ámbito médico, estos modelos pueden predecir brotes de enfermedades o identificar grupos de riesgo, facilitando intervenciones preventivas y mejorando la gestión de los recursos sanitarios.

Un ámbito de igual forma relevante es la interacción multimodal. Esta, implica la integración de múltiples formas de comunicación (texto, audio, imágenes) para comprender mejor el contexto y las necesidades del usuario. Los asistentes virtuales como Siri y Alexa utilizan el reconocimiento de voz y el procesamiento natural del lenguaje (PNL) para proporcionar respuestas más precisas y relevantes. Los sistemas de visión por computadora, por otro lado, mejoran las interacciones al capturar gestos y expresiones faciales.

No obstante, hasta el momento, los avances y la investigación en la integración de estos desarrollos punteros en un modelo general que pueda detectar la necesidad de interacción de un usuario frente a diversas situaciones aún no han alcanzado un nivel significativo. La complejidad radica en la necesidad de unir eficazmente tecnologías como el aprendizaje profundo, el análisis de sentimientos y la interacción multimodal en un marco cohesivo y adaptable. A pesar de los avances en cada una de estas áreas, la creación de un sistema robusto, versátil y escalable que pueda discernir y responder de manera inteligente a las necesidades cambiantes de los usuarios sigue siendo un desafío en el contexto de la investigación actual.

3. Modelo teórico

En esta sección se va a definir un modelo escalable, modular y eficiente para lograr clasificar la necesidad de interacción de la máquina con el usuario. Para conseguirlo, se ha decidido definir una abstracción de un arquetipo de 3 capas o niveles comunicados de forma ascendente a través del cual fluirán los datos. De esta forma, y como se puede apreciar en la figura1**,** el nivel más bajo estará conformado por los módulos de recolecta y tratamiento de datos, el nivel intermedio se encargará de su unificación y control, y el nivel superior tratará de predecir la necesidad de explicabilidad del sistema al usuario.

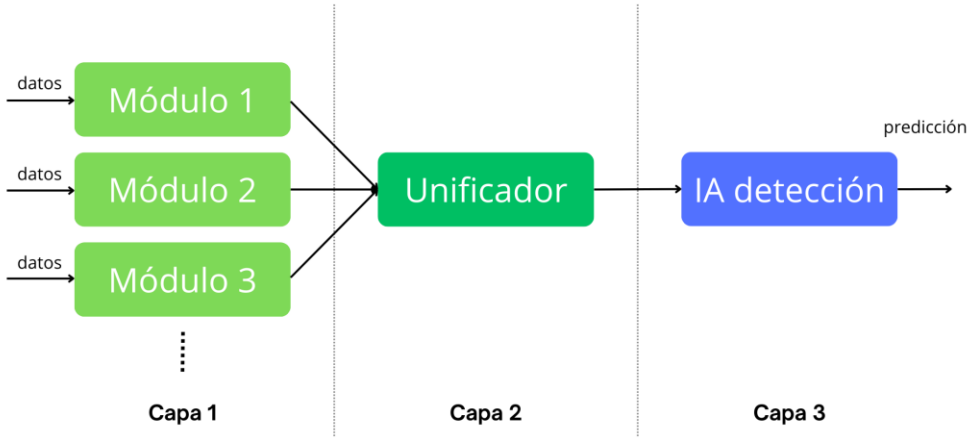


Figura 1. Arquitectura de capas

A continuación, se describirán con mayor profundidad cada una de las capas y su funcionamiento interno:

1. Capa 1: Esta capa se corresponde con la inferior del modelo. A través de la cual se recibirán todos los datos externos y se enviarán procesados(etiquetados) a la siguiente capa. Internamente se divide en al menos un módulo que se encarga de un tipo de dato. Puede haber tantos módulos como tipos de sistemas semióticos (audio, texto, vídeo ...) que se desee, por lo que el sistema goza de una gran capacidad de ampliación. En cuanto a cómo se compone un módulo, estos están divididos en dos subcapas, tal y como se puede apreciar en la figura 2:

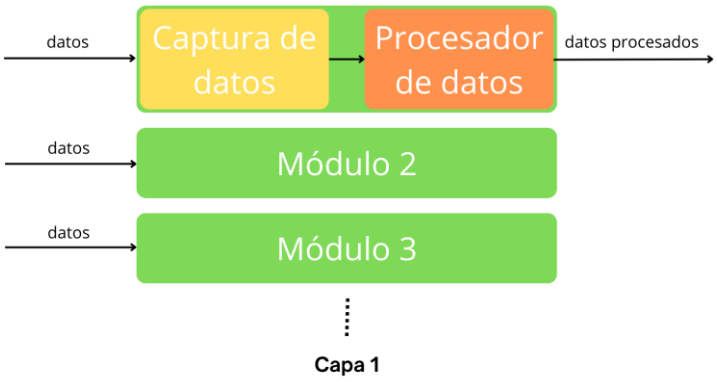


Figura 2. Subcapas de los módulos

i. Captura de los datos: en esta subcapa se capturan los datos de los *Inputs* sin tratar. Cada tipo de *Input* requiere una forma diferente de obtenerlo. Por ejemplo, en el caso de obtener vídeo de una webcam se puede utilizar la librería de OpenCV.

ii. Proceso de los datos: esta subcapa trata los datos obtenidos en la anterior y los clasifica y adapta al formato requerido por la segunda capa. Internamente puede estar formada por varios componentes que van transformando secuencialmente los datos al formato deseado.

Además de la gran capacidad de ampliación, este modelo de subcapas también nos otorga diferentes ventajas menos evidentes, como la capacidad de detectar y aislar rápidamente errores sin que pause la ejecución del modelo o la asincronía inherente en la ejecución paralela de los módulos entre otras.

2. Capa 2: esta capa se encarga de la captación, unión y envío de los datos a la tercera capa y del control de errores y llamadas a los diferentes módulos. Internamente, junto al sistema de control, el sistema de guardado se puede dividir en los siguientes dos componentes:

    i. Recolector de datos: como el propio nombre indica, este componente recopila los datos procesados de los módulos de la anterior capa. Como la primera capa trabaja de forma asíncrona, se debe formular un recurso para enviar los datos completos en el formato correcto. Algunas posibles soluciones son: utilizar un buffer donde los datos de cada módulo van guardándose de forma organizada, almacenar los datos de cada módulo en un fichero y periódicamente juntarlos o forzar la sincronía.

    ii. Procesador de datos: cuando hay al menos un dato de cada módulo, este componente lo formatea, lo normaliza, y finalmente, lo envía a la tercera capa.

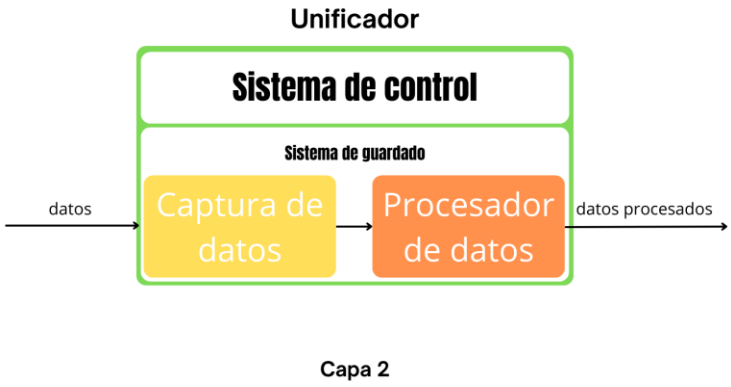


Figura 3. Arquitectura interna de la capa 2

3. Capa 3: en esta capa se localiza el componente más importante del modelo; la red neuronal capaz de clasificar los datos. Esta, a partir de los datos formateados de la anterior capa, debe ser capaz de predecir las necesidades del usuario, por lo que un correcto entrenamiento y ajuste de la red es imprescindible para su posterior buen desempeño.

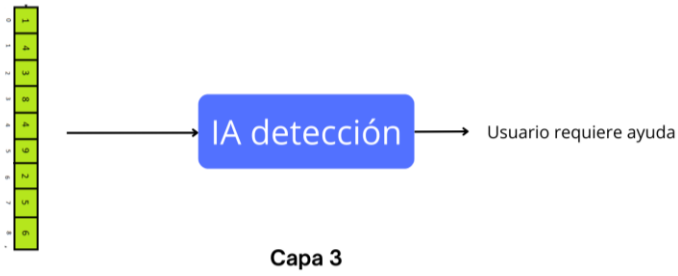


Figura 4. Funcionamiento de la capa 3

Para finalizar, cabe remarcar que el modelo cuenta con dos modos principales de funcionamiento. A continuación, se procede a citar cuales son, su uso definido y su funcionamiento interno:

1. Entrenamiento del clasificador: este modo pretende capturar todos los datos necesarios para el entrenamiento de la red neuronal de la tercera capa. Internamente, su ejecución de divide en dos Fases muy diferenciadas:

a. Fase 1: esta fase se corresponde con la captura y clasificación de los datos. Durante esta etapa se realizan diferentes pruebas a diversos usuarios midiendo si a partir de ciertos estímulos requieren de la interacción de la máquina y guardando los datos resultantes. Posteriormente, estos datos se catalogan y se unen para crear una base de datos. Internamente este proceso se traduce en que durante la captura de los datos únicamente se ejecutan las dos primeras capas. Estas, obtienen, transforman y unen los datos generados durante los diversos experimentos y finalmente los empaquetan en una base de datos.

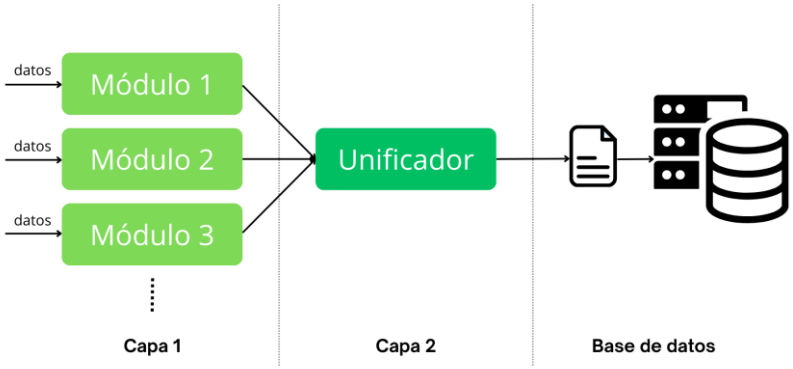


Figura 5. Guardado de datos en la Fase 1

b. Fase 2: partiendo de la base de datos, se formatean y normalizan los diferentes tipos catalogados y se separan en particiones de entrenamiento, validación y prueba para entrenar de forma correcta la red neuronal de la tercera capa.

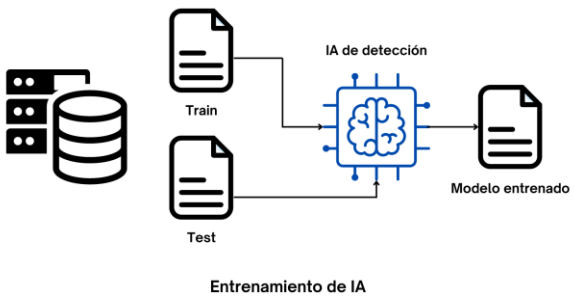


Figura 6. Entrenamiento de la IA de detección

La gran ventaja de esta arquitectura de subcapas es que en un mismo módulo se pueden varios clasificadores, optimizando así el uso de entradas y recursos del sistema. Por ejemplo, en el módulo de audio se puede disponer de un detector de emociones, de intencionalidad vocal, un traductor, etc.

2. Ejecución del modelo: este modo permite predecir en directo si el usuario requiere de la interacción de la máquina a partir de los diferentes datos percibidos por los módulos de la primera capa. Internamente, los datos van siendo capturados y tratados por la primera capa, unificados en un formato específico por la segunda, y finalmente, enviados a la tercera para generar la predicción. Cabe remarcar que para la ejecución de este modo se requiere antes haber ejecutado el modo de entrenamiento del clasificador (Modo 1).

Configuración asumida en el proyecto:

En el proyecto se ha asumido una arquitectura con una primera capa de tres módulos (audio, vídeo y dispositivos *smartwatch*), una segunda capa con guardado de ficheros por cada módulo con el objetivo de solucionar la asincronía, y una red neuronal simple en la tercera capa. Para su implementación se ha decidido usar el lenguaje de programación Python, por su disponibilidad de librerías de inteligencia artificial y su facilidad de uso. En las siguientes secciones se procederá con una descripción más detallada de cada componente y su implementación.

4. Módulo de audio

4.1. Recolector de datos de audio

Para la recolección de datos procedentes del audio del dispositivo se ha hecho uso de la librería PyAudio. Esta proporciona una interfaz para trabajar con audio a través de la biblioteca PortAudio, permitiendo a los desarrolladores de Python realizar tareas relacionadas con la entrada y salida de audio, como grabar y reproducir sonido. También resulta muy sencilla y versátil, y soporta la lectura y escritura de ficheros de audio de una forma rápida y eficiente.

La lectura del audio se ha realizado mediante una función llamada captura_audio(), la cual captura una grabación de audio del micrófono de duración RECORD_SECONDS(actualmente 30 segundos) y lo guarda como un fichero llamado OUTPUT_FILENAME(actualmente temp.wav) que servirá como un

fichero temporal sobre la cual la subcapa del procesador de datos de audio actuará. Es importante calibrar correctamente los dos parámetros para poder evitar posibles errores de desincronización.

4.2. Procesador de datos de audio

La implementación del procesador de datos de audio se compone de dos componentes principales: un transcriptor y traductor de audio a texto, también llamado *Speech to Text (STTE)* y un conversor de texto a emociones. A continuación, se procederá a detallar su funcionamiento e implementación.

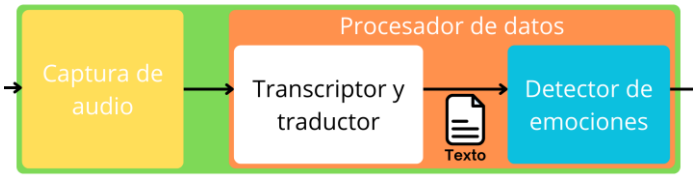


Figura 7. Arquitectura de procesador de datos de audio

4.2.1. Transcriptor y traductor

En cuanto a la implementación del primer componente se realizó una amplia investigación sobre las mejores posibles tecnologías y servicios para el proyecto. Estas debían contener modelos multilingüísticos altamente precisos, con una buena personalización y una alta velocidad de transcripción para soportar el modo en directo. A partir de estas restricciones se plantearon e implementaron dos modelos basados en el popular Whisper.

Whisper es un sistema de reconocimiento automático de voz (ASR) de OpenAI de código abierto y gratuito entrenado en 680.000 horas de datos supervisados multilingües y multitarea recopilados de la web [5]. Además, este modelo también permite realizar junto a la transcripción, la traducción de texto a diversos idiomas de una forma sencilla y altamente eficiente.

Internamente, Whisper emplea una arquitectura con un enfoque simple punto a punto basada en Transformer. Un Transformer es una arquitectura de red neuronal creado en 2017[6]. Esta, es líder en el procesamiento de lenguaje natural(NLP) debido a su alta capacidad de manejar secuencias de datos de forma altamente eficiente y efectiva. Algunas de las características más relevantes de esta arquitectura son su capacidad de paralelización, su escalabilidad y su capacidad de atención.

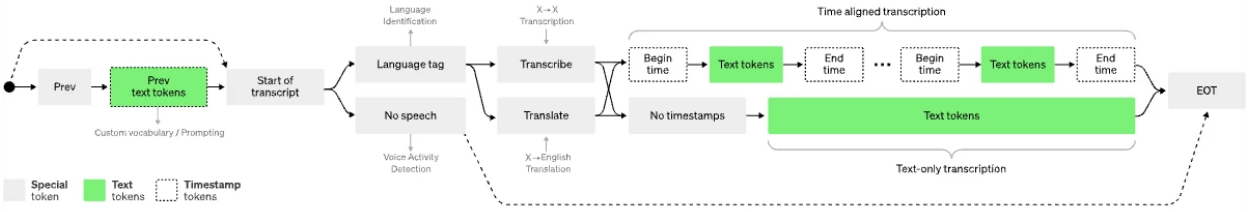


Figura 8. Arquitectura interna Whisper

A continuación, se procede a presentar y explicar las dos implementaciones basadas en Whisper de Open AI desarrolladas para el proyecto.

Implementación del modelo de transcripción de Whisper-JAX

Whisper-JAX[7] es una implementación optimizada del modelo de Whisper de OpenAI que funciona sobre JAX[8]. Esta, es una librería de Python desarrollada por Google, diseñada para ejecutar calculo numérico de alto rendimiento y aprendizaje automático a gran escala, que se caracteriza por la alta capacidad para realizar diferenciación automática, paralelización y la ejecución en hardware acelerado como GPUs y TPUs. Esta implementación sobre JAX, destaca por ser prácticamente compatible con todos los tipos de modelos oficiales de Whisper y por ser su API derivada más rápida disponible; llegando a multiplicar su rendimiento hasta 70 veces.

	OpenAI	Transformers	Whisper JAX	Whisper JAX
Framework	PyTorch	PyTorch	JAX	JAX
Backend	GPU	GPU	GPU	TPU
1 min	13.8s	4.54s	1.72s	0.45s
10 min	108.3s	20.2s	9.38s	2.01s
1 hour	1001.0s	126.1s	75.3s	13.8s

Tabla 1. Comparación de rendimientos de versiones de Whispers con Whisper-JAX

Para su inclusión en el proyecto se han debido instalar los "*drivers*" de CUDA y DNN de NVIDIA en el dispositivo de prueba junto a las versiones compatibles de JAX y Whisper-JAX para habilitar la aceleración de hardware y evitar posibles errores de compatibilidad. La sección de código correspondiente se sitúa en el método tr_audio. Su ejecución se puede dividir en tres partes:

1- La carga del modelo oficial de Whisper en Whisper-JAX: es importante hacerla únicamente dentro del método para que sólo se almacene en la memoria de la tarjeta gráfica una única vez.

2- La asignación de la tarea de transcribir y traducir al idioma inglés el texto del archivo temp.wav mediante la creación de un pipeline.

3- La unión de los diferentes segmentos del audio transcritos y traducidos en una única.

Una gran problemática que existe con este modelo es que, en comparación con muchas otras implementaciones, la alta velocidad de transcripción va vinculada a un excesivo uso de recursos del sistema y una gran dependencia de hardware especializado. Un ejemplo muy ilustrativo de este problema es la gran diferencia de asignación de memoria VRAM de la tarjeta gráfica; donde al cargar los parámetros de "large-v2" con el uso un modelo derivado más sencillo de Whisper, se puede llegar a consumir 3Gb de memoria de media mientras que Whisper-JAX puede llegar a intentar alojar hasta 21Gb. Por esta razón, sólo se recomienda hacer uso de este modelo de alta velocidad en el caso de necesitar transcripciones altamente veloces y disponer de los dispositivos especializados necesarios.

Implementación del Modelo de Faster-Whisper

Faster-Whisper es una reimplementación del modelo Whisper de OpenAI mediante el uso de CTranslate2. Esta, es una librería de C++ y Python especializada en la eficiencia de las inferencias mediante el uso de modelos de transformadores, resultando de esta forma altamente recomendable para la creación de modelos de IA ligeros y rápidos [9]. Algunas de las características que incluye esta librería para garantizar la eficiencia son el soporte de pesos con precisión reducida, la detección y asignación automática de código en la CPU, el uso de memoria dinámica, su ejecución paralela y asíncrona, la detección y adaptación a las diferentes arquitecturas de CPU o el reducido peso en el disco. Por estas razones, el modelo de Faster-Whisper progresa sobre el modelo base de Whisper, llegando a ser hasta 4 veces más rápido que el original, y a su vez consumiendo menos memoria.

Implementation	Precision	Beam size	Time	Max. GPU memory	Max. CPU memory
openai/whisper	fp16	5	4m30s	11325MB	9439MB
faster-whisper	fp16	5	54s	4755MB	3244MB
faster-whisper	int8	5	59s	3091MB	3117MB

Tabla 2. Rendimiento de Faster-Whisper en comparación con Whisper base

Por lo que hace a la integración de Faster-Whisper en el proyecto, para mejorar el rendimiento y consumo de memoria, al igual que en Whisper-JAX, se ha decidido ejecutar el modelo en la GPU, requiriendo la instalación de las versiones pertinentes de los drivers de NVIDIA. En cuanto al código implementado, al igual que en Whisper-JAX, se sitúa en la función tr_audio. En este, se pueden identificar las 2 siguientes secciones de código:

1- Carga del modelo: donde Faster-Whisper se carga dentro de la función con un modelo "*large-v2*" de OpenAI, la ejecución en los núcleos CUDA de la GPU y una precisión de coma flotante 16.

2- Creación de tarea: donde se carga el archivo de audio temp.wav y se crea la tarea de transcribir y traducir el archivo mediante segmentos de una cuarta parte de la duración de la grabación.

Se recomienda ampliamente el uso de esta librería en el proyecto ante Whisper-JAX ya que, menos en casos específicos, la diferencia de tiempo de transcripción no resulta problemática durante la ejecución del programa y la diferencia de uso de recursos es altamente notable, especialmente, en el uso de la memoria VRAM, donde el modelo más grande y preciso únicamente consume 3Gb.

4.2.2. Conversor de Texto a emociones

Como especifica su propio nombre, este componente se encarga de recibir el audio transcrito y traducido al inglés y clasificar de forma rápida y precisa su contenido en hasta 7 emociones diferentes. Estas emociones son: disgusto, neutral, sorpresa, enfado, tristeza, miedo y felicidad.

En cuanto a la implementación de este componente, se han desarrollado dos posibles opciones: crear y entrenar un modelo personalizado y usar un modelo pre-entrenado. A continuación, se profundizará con detalle cada una de estas opciones.

Modelo personalizado

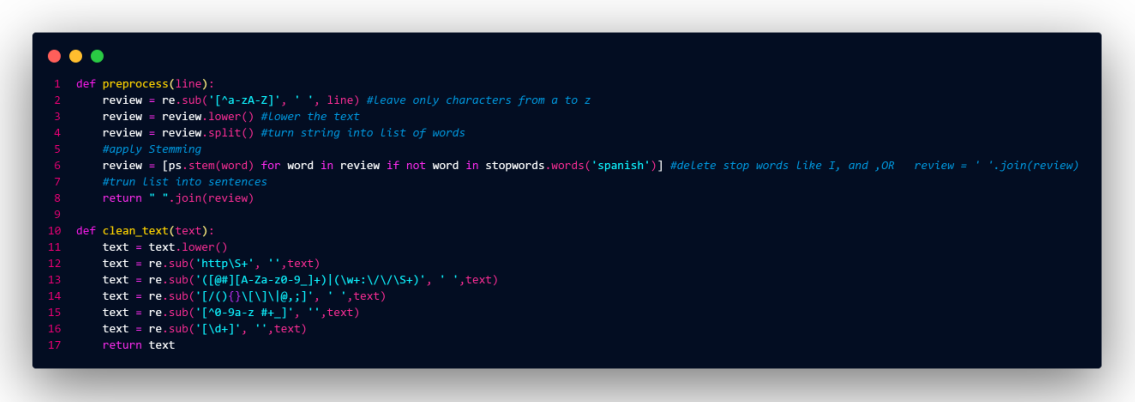
Esta opción se basa en la creación y entrenamiento de un modelo completamente nuevo. Para ello, primeramente, se ha realizado una búsqueda de posibles *datasets* ya clasificados de texto y su emoción adjunta. De esta búsqueda se han obtenido los dos mejores *datasets* compatibles son los requerimientos ya mencionados. Estos son:

- Emotion Detection from Text: es un popular *dataset* dedicado al procesamiento natural del lenguaje (NLP) compuesto por 40000 entradas de texto clasificadas en emociones procedentes de comentarios de la red social Twitter.

- Emotions dataset for NLP: es un *dataset* dedicado al procesamiento natural del lenguaje compuesto por 2000 entradas clasificadas en 6 diferentes emociones.

Con los d*atasets* ya acotados se ha procedido a su unión en uno de mayor tamaño y complejidad. Para conseguirlo, se han seguido los siguientes pasos:

1- Procesado individual de cada una de las entradas de los *datasets*: por cada entrada, se han eliminado todos los números y caracteres especiales mediante la función clean_text y la librería re y posteriormente, se ha reducido cada palabra a su raíz a partir del uso de un proceso de *Stemming*, implementado en la función preprocess.



```
def preprocess(line):
    review = re.sub('[^a-zA-Z]', ' ', line) #Leave only characters from a to z
    review = review.lower() #Lower the text
    review = review.split() #turn string into list of words
    #apply Stemming
    review = [ps.stem(word) for word in review if not word in stopwords.words('spanish')] #delete stop words like I, and ,OR   review = ' '.join(review)
    #trun list into sentences
    return " ".join(review)

def clean_text(text):
    text = text.lower()
    text = re.sub('http\S+', '',text)
    text = re.sub('([@#][A-Za-z0-9_]+)|(\w+:\/\/\S+)', ' ',text)
    text = re.sub('[/(){}\[\]\|@,;]', ' ',text)
    text = re.sub('[^0-9a-z #+_]', '',text)
    text = re.sub('[\d+]', '',text)
    return text
```

Código 1. Preproceso del dataset

2- Carga de los *datasets* resultantes en un DataFrame: cargamos los archivos de los *datasets* en la estructura de datos DataFrame de la librería Pandas. Esta estructura se puede percibir como una tabla formada por tantas filas como entradas y columnas como tipos datos de la entrada del d*ataset*.

3- Cambio de nombre de las columnas del DataFrame más pequeño (Emotions dataset for NLP) al del más grande.

4- Eliminación de columnas innecesarias de los DataFrames.

5- Unión de los dos DataFrames en uno.

6- Guardado del DataFrame resultante en un archivo.

Finalmente, se transforma el *dataset* final mediante el uso de un CountVectorizer de la librería Sklearn, y posteriormente, se entrena en una red neuronal simple. Esta red neuronal, creada a partir de la librería TensorFlow, está compuesta por tres capas: una capa para las entradas con 12 neuronas y una capa oculta con 8

neuronas que hacen uso de la función *ReLU* y una última capa de salida de 6 neuronas con una función *Softmax*.

Una vez entrenada y guardada la red neuronal, se carga en el componente de predicción de emociones y el texto pasa a predecir a la función *predict* del modelo para obtener su emoción.

Esta aproximación al diseño e implementación del predictor de emociones puede ser beneficiosa para casos en que la variedad del texto a recibir esté muy acotada, ya sea por la temática o el ámbito en que se esté capturando el audio. No obstante, en situaciones generales puede resultar ineficiente en comparación con otros modelos pre-entrenados.

Modelo pre-entrenado

En esta opción se ha realizado una búsqueda de un modelo pre-entrenado de detección de emociones, basado en texto, de gran precisión y eficiencia. El resultado de esta búsqueda es el modelo Emotion-english-distilroberta-base creado por Jochen Hartmann. Este, permite clasificar 7 emociones, está entrenado a partir de diversos *datasets* y su modelo deriva de distilRoBETa base, cuyo núcleo es BERT. Este último, se trata de un modelo de vanguardia de clasificación de texto creado por Google en 2018. Gran parte de la popularidad e importancia que ha logrado este, se debe a su gran facilidad de preentrenamiento y ajuste fino, su bidireccionalidad y a que utiliza la arquitectura Transformers. Algunas de sus ventajas más destacables son su versatilidad o su comprensión contextual profunda.

En comparación con la anterior opción, la implementación de este modelo se reduce a, mediante el uso de la librería de Transformers de Hugging Face, simplemente crear un pipeline con la referencia del clasificador de Jochen Harmann.

Por lo que hace a la integración con el proyecto, se puede dividir en las siguientes secciones situadas a continuación del componente de transcripción y traducción en el método tr_audio:

1- Crear el pipeline asociado al modelo pre-entrenado de Jochen Hartmann.

2- Obtener la lista de probabilidades por cada segmento transcrito y traducido del anterior componente.

3- Seleccionar la emoción con mayor probabilidad obtenida en la lista.

5. Módulo de video

5.1. Recolector de datos de video

Por lo que hace a la recolección de los datos del módulo de vídeo se ha hecho uso de la conocida librería de OpenCV. OpenCV es una biblioteca libre de visión artificial originalmente desarrollada por Intel. Esta, se ha utilizado en una gran cantidad de aplicaciones, y actualmente, se la sigue mencionando como la biblioteca más popular de visión artificial, detección de movimiento, reconocimiento de objetos o reconstrucción 3D a partir de imágenes entre otras aplicaciones [4].

Gran parte de su popularidad se debe a que es una librería libre, multiplataforma, bien documentada y explicada y con muchos años de desarrollo.

En cuanto a su implementación, al igual que en el módulo de audio, se encuentra en una función específica para realizar la captura de los diferentes *frames* de la webcam del dispositivo. Una parte importante del código es el ratio en segundos en que se capturan los *frames* conocido como FRAME_TIMER. Al igual que en los parámetros de la sección 4.1, este, se debe ajustar correctamente para descartar posibles errores de sincronización.

5.2. Procesador de datos de video

El procesador de datos de vídeo es el encargado de transformar cada una de las imágenes recibidas por el recolector de vídeo a información valorable para nuestra IA. En este caso, se ha decidido realizar una detección de emociones a partir de los diferentes rasgos faciales del usuario. Para ello, se han desarrollado una serie de componentes para obtener la mayor precisión posible y simultáneamente alcanzar un rendimiento óptimo. A continuación, se describirán detalladamente cada uno de ellos y los procesos que efectúan para conseguir dichos objetivos.

5.2.1. **Clasificador facial:** este componente es el encargado de detectar los diferentes rostros de cada imagen. Para conseguirlo, se ha hecho uso de un clasificador en cascada [11] que, actuando mediante el uso de clasificadores más simples como Haar o LBP llamados etapas, van eliminando sobre una imagen las regiones en las que no se ha detectado cierto patrón. En este caso, se ha utilizado un patrón por defecto capaz de detectar caras de forma frontal, utilizando el clasificador o etapa Haar, llamado haarcascade_frontalface_default.

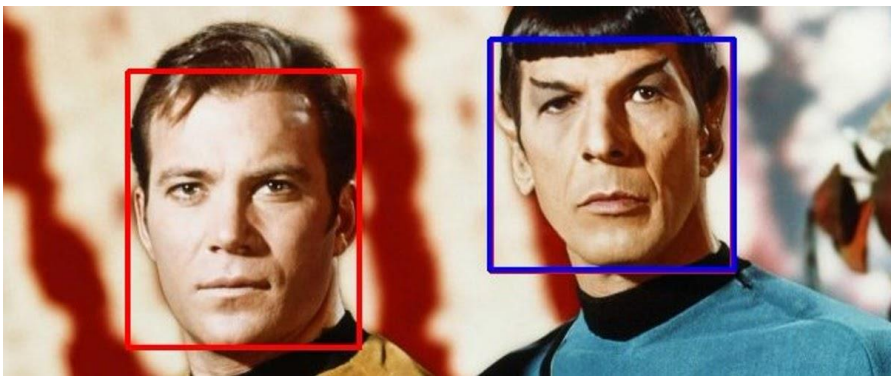


Figura 9. Detección de rostros en OpenCV

5.2.2. **Procesador de la imagen facial detectada:** una vez se ha detectado una cara, se procede a hacer una serie de transformaciones a la imagen original para obtener una imagen más reducida con el rostro del usuario centrado y con una resolución menor. Tal y como se puede apreciar en la FIGURA estos procesos consisten en: generar imágenes cuadradas donde se sitúen los diferentes rostros detectados y en disminuir la resolución de estas imágenes a un valor establecido, en el caso de la configuración actual del proyecto, 256 * 256 píxeles. Posteriormente, estas imágenes se guardarán para que el clasificador de emociones asíncrono las etiquete.

Figura 10. Imágenes postprocesadas

5.2.3. **Clasificador de Emociones:** este se encarga de identificar las diferentes emociones de las caras de las imágenes generadas en el anterior componente. Para conseguirlo se ha decidido hacer uso de la librería DeepFace [12].

DeepFace es una librería liviana y de código abierto diseñada para abordar el desafío del reconocimiento facial. Esta destaca por incorporar todos los modelos de vanguardia para el reconocimiento facial moderno como VFFFace, Facenet o OpenFace[13] de forma sencilla y rápida de implementar. Algunas de las funciones más destacadas que esta librería ofrece son la verificación facial, el reconocimiento de rostros, el análisis facial en tiempo real o el análisis de atributos faciales. Este último nos permite analizar, entre otros atributos como el género o la edad estimada del usuario, la emoción del usuario en un fotograma.

Por lo que concierne a la integración de esta librería en el proyecto, cabe remarcar que, debido a dependencias con las diferentes versiones de otras librerías, este se ha implementado como otro programa independiente que se ejecuta en paralelo junto al capturador y los anteriores componentes. A continuación, se explicará la el funcionamiento de la implementación del componente.

1- Se cargan las diferentes imágenes procesadas y guardadas por los anteriores componentes.

2- Por cada imagen se lanza la función de analizar de DeepFace indicándole como parámetro que únicamente queremos que nos analice la posible emoción detectada.

3- Del resultado de este análisis seleccionamos la emoción más probable o emoción dominante.

6. Módulo de *smartwatch*

Este módulo es el encargado de obtener los datos en tiempo real de un dispositivo *smartwatch* y procesarlos para obtener diferentes resultados. En este proyecto, se ha decidido hacer uso de los dispositivos de Fitbit por su fácil acceso al mercado de parte del consumidor medio y la sencillez que estos ofrecen para desarrollar herramientas para los mismos.

Fitbit es una marca de pulseras y relojes inteligentes enfocada principalmente en el seguimiento de la actividad física y salud. Esta, fue adquirida por Google el 1 de noviembre de 2019 y dispone de una gran cantidad de series de dispositivos de todo tipo de gamas; tanto asequibles para una primera entrada del púbico en el sector, como dispositivos punteros para consumidores de alto nivel.

Fitbit también incluye un sencillo y bien documentado portal de desarrollo [14], donde tanto los usuarios más avanzados como compañías disponen de diferentes herramientas como un simulador de sus dispositivos o una API. Esta última, permite acceder a diferentes tipos de datos en directo como los sensores del propio dispositivo del usuario o sus métricas y datos generados y guardados en su cuenta.

Por ello, se ha decidido desarrollar una aplicación compatible con gran parte de las series de Fitbit, que recupere en directo los datos del usuario y del dispositivo, y que posteriormente los procese y envíe a nuestro proyecto. A continuación, se explicarán las diferentes subcapas del módulo que han hecho posible esta función.

6.1. Recolector de datos de *smartwatch*

Por lo que hace a la recolección de datos de la API podemos distinguir dos tipos de datos:

- Las métricas pertenecientes a la cuenta del usuario: estas son visibles a través de la aplicación de Fitbit y se calculan internamente por sus servicios. Para acceder a ellas se deben realizar diferentes solicitudes POST y GET a la API de Fitbit correspondiente. Algunas de estas métricas pueden ser la calidad del sueño o de diferencia de ritmo cardíaco durante un intervalo de tiempo.

- Los datos del dispositivo inteligente: estos datos son los pertenecientes a los diferentes sistemas y sensores que contiene el dispositivo integrado. Para acceder a ellos, es necesario vincular el reloj a la cuenta con los permisos pertinentes de Fitbit e instalar la aplicación desarrollada. Estos datos se pueden conseguir de forma única o en directo. Algunos de los datos disponibles en la mayoría de los relojes es la frecuencia cardíaca, el giroscopio y el acelerómetro o intrínsecos al dispositivo como el porcentaje actual de su batería.

En cuanto al código empleado para la obtención de estos, también se puede agrupar en los diferentes tipos de datos. A continuación, se explicarán los diferentes procesos seguidos para obtener cada tipo:

1- Recolector de datos de la cuenta de usuarios: como se ha comentado previamente, esta se realiza mediante el uso de llamadas POST y GET a la API de Fitbit. Para ello, primeramente, se debe generar un token de autentificación, ya sea mediante la herramienta web que facilita la compañía o a través de una petición POST con los datos de identificación de la aplicación, su código seguro y un código cifrado. Una vez conseguido, y tal como se puede apreciar en la FIGURA, se llama a la función previamente desarrollada fetchData, pasándole como parámetro el tipo de dato a conseguir y haciendo uso del token generado para autentificarse. Los datos recabados mediante este método en el proyecto son la calidad del sueño y la diferencia de rimo cardíaco.

```
// Función para hacer una solicitud al API de Fitbit
async function fetchData(endpoint) {
  const url = `https://api.fitbit.com/1.2/${endpoint}`;
  const headers = {
    'Authorization': `Bearer ${accessToken}`
  };

  try {
    const response = await fetch(url, { headers });
    if (!response.ok) {
      throw new Error(`HTTP error! Status: ${response.status}`);
    }
    return await response.json();
  } catch (error) {
    console.error('Error fetching data:', error);
    return null;
  }
}
```

Código 2. Implementación de fetchData

2- Recolector de datos de sensores: en esta sección del programa se pretende obtener el ritmo cardíaco del reloj inteligente en tiempo real. Para obtenerlo, se comprueba si el dispositivo dispone del sensor y si la aplicación desarrollada cuenta con los permisos necesarios para acceder a él. Posteriormente, creamos una instancia del

sensor llamada hrm y le añadimos un *callback* que guarde el ritmo cardíaco en el texto de la variable t. Finalmente, se procede a iniciar el sensor del reloj.

6.2. Procesador de datos de *smartwatch*

El procesador de datos del *smartwatch* está compuesto por un único componente que, partiendo de los datos obtenidos en la anterior subcapa, calcula una aproximación del nivel de estrés del usuario, puntuando de 0(tranquilo) a 100(muy estresado) su nivel percibido. La implementación del componente se ha realizado en la función calcularNivelDeEstrés donde, a través de la asignación de ciertos pesos a variables y a posteriores operaciones sobre ellos, se ha obtenido el valor deseado. A continuación, se detallarán las asignaciones y cálculos empleados para obtener dicho objetivo.

1- Asignación de peso a los diferentes parámetros de la función. Estos pesos se han designado mediante el revisado de *papers*[15][16][17] sobre la influencia de dichos parámetros en el estrés junto a la comparación de los resultados con aplicaciones comerciales.

2- Multiplicación y suma de cada uno de los parámetros con su peso correspondiente.

7. Unión de módulos

La implementación de la capa de unión de los módulos se puede dividir en dos sistemas coexistentes y paralelos: el sistema de control del programa y el sistema de guardado y proceso de datos. A continuación, se procede a describir el funcionamiento e implementación de cada uno de ellos.

Sistema de control del programa

El sistema de control del programa es una pieza fundamental del funcionamiento del proyecto debido a que este se encarga de gestionar todas las ejecuciones de los diferentes componentes y capas, su sincronización y de sus errores. Por estas razones, la correcta definición de los órdenes de ejecución de los módulos y capas y sus parámetros necesarios resultan imprescindibles para alcanzar una ejecución satisfactoria y óptima. Internamente, y como se puede observar en la FIGURA, este se puede entender como la una línea de tiempo donde los diferentes módulos y componentes se sitúan. De esta forma, y para garantizar una ejecución sin errores, los módulos, componentes o tareas con dependencias con otros siempre se deben ejecutar secuencialmente. En el caso de no existir dependencias entre módulos o procesos, se podrán ejecutar en paralelo. Cabe remarcar también que un inicio del programa sólo estará sincronizado en el caso en que todos los módulos y procesos del proyecto se hayan cargado previamente en memoria, por lo que resulta necesario que cada módulo o proceso espere a la carga de los demás antes de iniciar su ejecución.

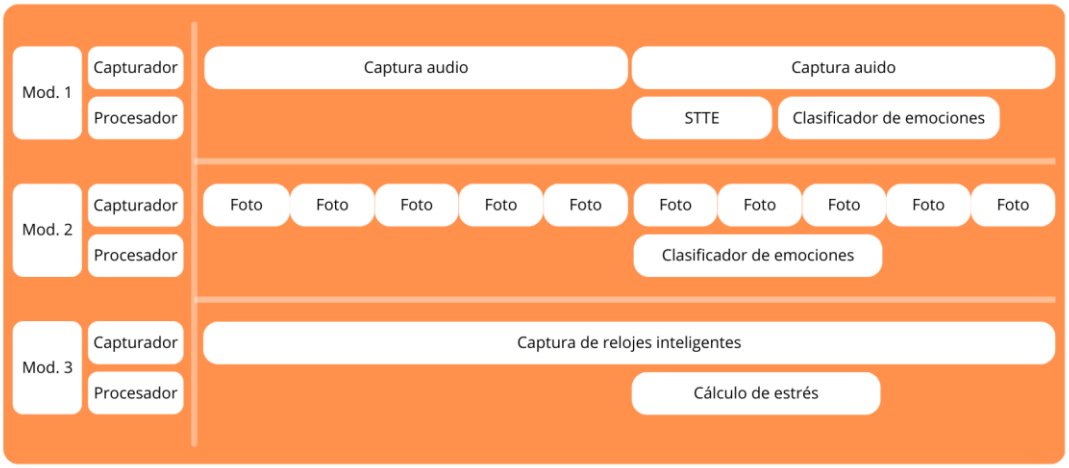


Figura 11. Ejemplo de linea de tiempo

En cuanto a la integración e implementación de este sistema en proyecto nos encontramos con una grave problemática relacionada con el lenguaje de programación usado; Python.

Problemática de Python

Esta problemática con el popular lenguaje de programación está relacionada con el uso de los hilos. Teóricamente, Python es un lenguaje con soporte de paralelismo mediante el empleo de hilos sobre diferentes funciones del programa, pero realmente, este no los paraleliza, sino que los ejecuta de forma secuencial y sólo detiene o transfiere su ejecución a otro hilo si el actual se pausa o finaliza. Esto, se debe al GIL. El GIL o *Global Interpreter Lock* es un mecanismo de Python que gestiona el acceso simultáneo de los hilos a los objetos del lenguaje. Este, con el propósito de asegurar la seguridad de los hilos y simplificar el manejo de la memoria, actúa como un *mutex* pausando los hilos para proteger los objetos e impidiendo de esta forma que se ejecute el código del programa de forma simultánea.

Frente a esta problemática y la necesidad de paralelismo para el correcto funcionamiento de la arquitectura de capas y módulos sólo existe una posible solución, el *multiprocessing*. El *multiprocessing* o multiproceso se trata de crear instancias separadas del programa con su propio GIL independiente. De esta forma, al iniciar dichas instancias independientes, cada una ejecuta su porción de código correspondiente de forma paralela, dejando la pausa o ejecución del proceso al cargo del sistema operativo. La principal desventaja de esta implementación es que, al crear procesos independientes, se dificulta gravemente o se pierde la capacidad de usar variables compartidas, por lo que hay que desarrollar otros sistemas como la comunicación por fichero o mensajes.

Con la principal problemática solventada, se ha decidido que cada una de las subcapas de los módulos se ejecute en un proceso independiente. De esta forma, al permitir que mientras se estén capturando los datos de un módulo, se procese la captura anterior de forma simultánea, augmentamos la eficiencia del programa. Análogamente, mientas los módulos están capturando y procesando los datos, la IA de predicción estará en paralelamente en ejecución analizando los anteriormente generados.

En cuanto a la implementación del lanzador inicial de los procesos, los diferentes procesos han sido lanzados en un orden específico y con cierto período de tiempo de diferencia para que, como se ha comentado anteriormente, se obligue a que todos ellos empiecen su ejecución de forma simultánea. Otra forma de conseguir esto diferente al uso de sleeps podría ser la espera y el intercambio de mensajes entre procesos para saber en qué estado están. Esta opción no ha sido implementada para fomentar la sencillez y la prevención de errores del programa.

```python
# Python de DeepFace
    subprocess.Popen(batch_file + f"\t {str(FRAME_TIMER)} {str(BATCH_SECONDS)}")

    # Captura de vídeo
    video_mod_process = multiprocessing.Process(target=video_module)
    video_mod_process.start()
    print("Iniciado el mod de video")
    time.sleep(1)

    # Módulo de audio
    # Captura de audio
    audio_cap_process = multiprocessing.Process(target=captura_audio)
    audio_cap_process.start()
    print("Iniciada grabadora de audio")

    # Tiempo de grabar + 1 segundo de guardar - 5 segundos de carga de los modelos
    time.sleep(RECORD_SECONDS - 4)

    # Preproceso de audio
    audio_trans_process = multiprocessing.Process(target=tr_audio)
    audio_trans_process.start()
    print("Iniciado transcripción y traducción de audio")


    # Espera de la función principal
    audio_cap_process.join()
    audio_trans_process.join()
    video_mod_process.join()
```

Código 3. Lanzamiento sincronizado del programa principal

Por lo que hace a la implementación del sistema de control en los procesos se han añadido dos secciones al código de la función: si es necesaria, una al principio que espera a que los demás procesos se preparen y otra sección al final que pausa su ejecución hasta resolver las dependencias con otros procesos.

Sistema de guardado y proceso de los datos

Este sistema de guardado y proceso de los datos es el encargado tanto de guardar los resultados generados por los diferentes módulos, como de unificarlos y posteriormente transformarlos al formato interpretable por la IA de detección. Para el correcto desempeño del programa, la lectura y el guardado de estos datos debe ser altamente ligera y eficiente, por lo que el empaquetado mediante formatos populares como JSON no son óptimos. Por ello, se ha creado un formato personalizado, de extensión CCS, con únicamente los campos necesarios para el programa. A continuación, se procede a describir detalladamente su composición.

El formato CCS está compuesto por dos secciones principales separadas por el símbolo del punto al inicio de la línea. En la primera sección se especificarán los diferentes

parámetros del módulo como por ejemplo, el intervalo de tiempo en el que se ha realizado la captura. Estos parámetros estarán formateados, de forma de que al inicio de la línea se le asignará el nombre del parámetro, seguido por el símbolo de los dos puntos, y finalizado por el valor de dicho parámetro. La segunda sección corresponde a la línea de tiempo de la captura. Cada línea de esta sección estará compuesta por el instante de tiempo en que se realizó la captura, un símbolo de resta y el valor procesado en ese instante.

Código 4. Ejemplo simple de un archivo CCS

En cuanto a la integración del guardado en el proyecto, cada subcapa de proceso de los módulos se encarga de confeccionar y guardar su fichero CCS con la información de la captura procesada. Para ello, al inicio del preproceso se escriben sus parámetros en el CCS y a medida que este va iterando y clasificando, los datos se van añadiendo al fichero.

Por lo que respecta a la unión de los datos, esta se realiza en la llamada a la IA de detección y se divide en tres etapas:

1- Abrir los ficheros CCS en modo lectura para no bloquear a las subcapas de proceso.

2- Guardar de cada una los datos del timeline desde el último punto que analizó la IA de detección en la anterior iteración, hasta el último momento procesado en

la actual, y posteriormente, hacer la media de cada uno de los mismos datos en módulo.

3- Juntar todos los datos en una array unidimensional.

8. IA de detección

Al tratarse de la pieza encargada de la predicción del estado del usuario, se puede afirmar que esta inteligencia artificial es el núcleo principal del proyecto, por lo que su correcto desarrollo, entrenamiento y ajuste es esencial para asegurar la correcta funcionalidad del programa. Por ello, se ha decidido hacer uso de diversas técnicas, métricas y tipos de modelos de IA artificial para cumplir dicho objetivo. A continuación, se describen y muestran los procesos seguidos para preparar su entrenamiento y los diferentes tipos y métricas resultantes de las IA implementadas.

1- Carga i división del *dataset* de vectores unidimensionales etiquetados en una partición de entrenamiento del 80% de *train* y 20% de *test* mediante la función read_csv de pandas y la función functi propia.

2- Entrenamiento de la IA mediante la función train_model. Esta función requiere como parámetros las particiones de *train* y el tipo de modelo de IA a entrenar. En esta función se han implementado los siguientes modelos:

a. Random Forest(RF): construye un conjunto de árboles de decisión entrenados de forma independiente utilizando subconjuntos de vectores.

b. K-Nearest Neighbors(KNN): clasifica una nueva instancia basándose en las clases de sus K vecinos más cercanos en el espacio de características.

c. Red neuronal simple: red neuronal compuesta por una capa de entrada y 3 capas ocultas con 32 neuronas cada una y haciendo uso de la función *Selu*. Finalmente, esta tiene una capa de salida de dos neuronas con la función *Softmax*.

Posteriormente, el modelo de inteligencia artificial seleccionado se entrena utilizando el optimizador de *Adam* y la función de pérdida de *sparse_categorical_crossentropy* durante 30 *epochs* con un *batch_size* de 10.

3- Guardado del modelo entrenado.

9. Pruebas y resultados

Previamente a la definición del tipo de pruebas y su resultado, se van a definir la configuración completa del modelo empleado y las características del dispositivo utilizado para llevarlas a cabo.

Modelo empleado

El modelo empleado está compuesto por una primera capa con un módulo de video que utiliza deepFace de procesador, un módulo de audio con Faster-Whisper, un módulo de *smartwatch* con Fitbit, una segunda capa con control por "*timers*" y guardado por ficheros CCS y una última capa con el modelo de IA más preciso.

Dispositivo utilizado

El dispositivo utilizado para llevar a cabo estas pruebas es un convertible dos en uno Surface Pro 8, con un i7-1185G7 de 4 núcleos, 16Gb de memoria RAM, 512Gb de almacenamiento SSD y una gráfica externa 4060Ti de 8Gb de memoria de vídeo.

Caso de uso y Pruebas

Con la configuración usada ya establecida, se va a proceder con la descripción del caso de uso planteado y las pruebas empleadas. Para simplificar las pruebas del modelo se ha planteado un caso de uso concreto; la comprensión de una app o un documento. Las pruebas de este caso consistían en mostrar y permitir interactuar a 7 sujetos con diferentes recursos de variada dificultad de comprensión y ámbito, como pueden ser documentos de investigación o sencillas interfaces de aplicaciones.

Estos sujetos presentaban una variación de edad y estudios entre sí para fomentar una mayor amplitud del modelo. Al finalizar el período de la prueba, estos debían notificar si habían entendido el recurso mostrado y aportar una pequeña explicación para validar la certeza de la contestación.

Tras finalizar las pruebas con los sujetos se obtuvo un *dataset* con 1000 entradas etiquetadas para entrenar la IA. A continuación, se muestran los resultados de entrenar los diferentes modelos de IA mencionados en el anterior punto con este *dataset*.

Modelo RF

Por lo que hace al resultado del entrenamiento del modelo de Random Forest, se ha obtenido los siguientes resultados:

Accuracy	0.89
Precision	0.90
Recall	0.89
F1	0.89

Tabla 3. Resultados del modelo RF

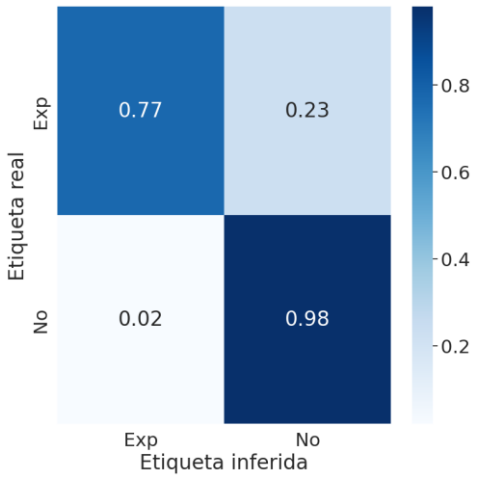


Gráfico 1. Precisión del modelo RF

Modelo KNN

Por lo que hace al resultado del entrenamiento del modelo de K-Nearest Neighbors, se han obtenido los siguientes resultados:

Accuracy	0.78
Precision	0.78
Recall	0.78
F1	0.78

Tabla 4. Resultados del modelo KNN

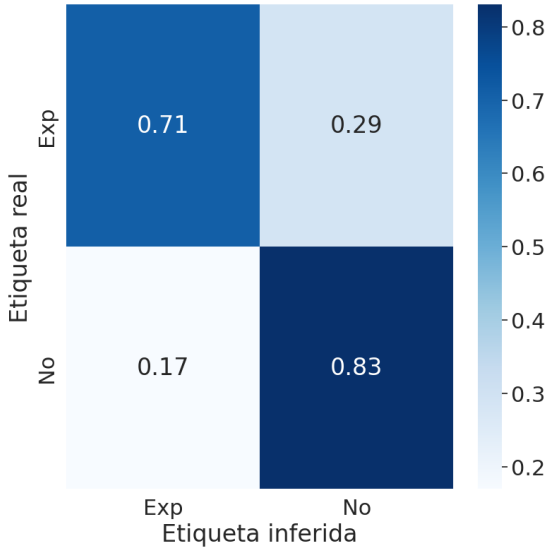


Gráfico 2. Precisión del modelo KNN

Modelo Red Neuronal

Por lo que hace al resultado del entrenamiento del modelo de Red Neuronal simple, se han obtenido los siguientes resultados:

Accuracy	0.84
Precision	0.84
Recall	0.84
F1	0.84

Tabla 5. Resultados del modelo RN

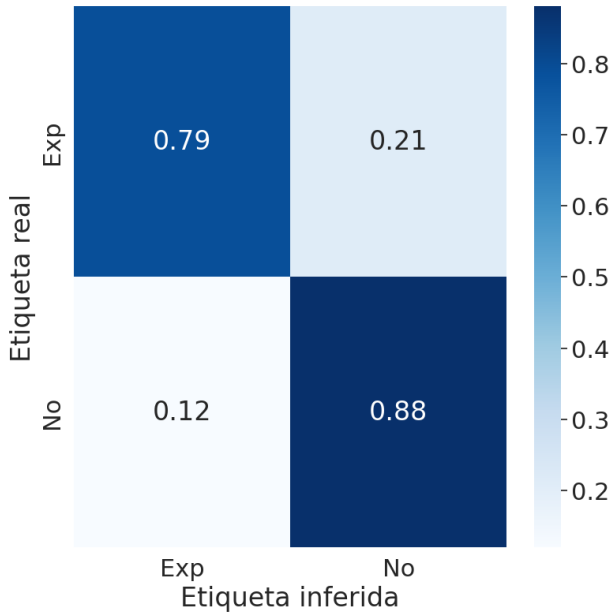


Gráfico 3. Precisión del modelo RN

A partir de estos resultados, se puede concluir que el modelo con mejor precisión de los tres planteados es Random Forest, con una capacidad estimada de predecir de forma correcta si el usuario requiere o no de la intervención de la máquina el 89% de las veces.

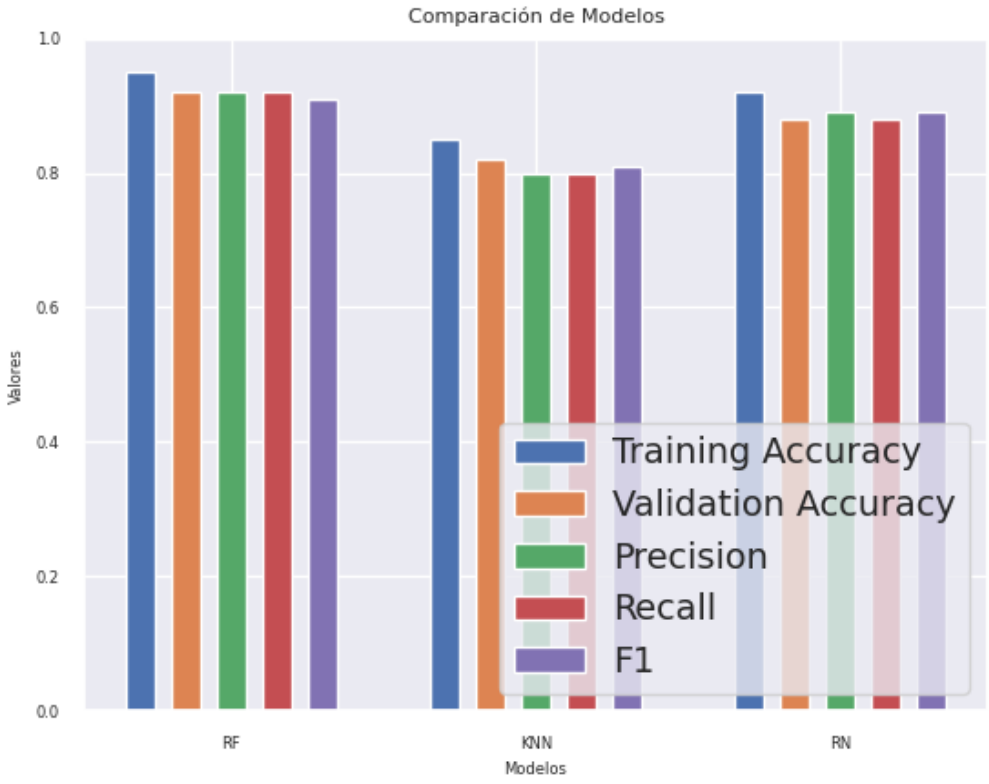


Gráfico 4. Comparación de valores de los modelos

10. Conclusiones finales

Este proyecto tiene como objetivo abordar uno de los campos más innovadores y desafiantes del campo de la inteligencia artificial moderna; el desarrollo de un modelo capaz de entender y reaccionar ante las necesidades y el estado del ser humano.

Por ello, se ha diseñado una arquitectura general altamente eficiente, modular y escalable que hace uso de varias técnicas y tecnologías de vanguardia de la inteligencia artificial, capaz de comprender de forma dinámica la situación y las necesidades del usuario.

De esta forma, se puede concluir que el resultado del proyecto ha sido satisfactorio, ya que se han conseguido cumplir todos los objetivos planteados en su inicio, presentando de esta forma una posible pauta a seguir en los futuros modelos de interacción humano-máquina.

11. Relación del trabajo desarrollado con los estudios cursados

En este punto se va a citar y detallar las asignaturas con mayor grado de relación con las habilidades y tecnologías necesarias para el desarrollo del proyecto. Estas asignaturas y su relación con el proyecto son:

1- Sistemas de almacenamiento y recuperación de información (SAR): esta asignatura ha sido de gran utilidad para la comprensión e implementación de las diferentes metodologías posibles de recuperación y tratamiento de datos altamente utilizada al largo del proyecto.

2- Sistemas Inteligentes (SIN): en esta asignatura se han impartido los principios de la IA que han servido para entender el concepto, funcionamiento y tipos de clasificadores implementados en el proyecto.

3- Percepción (PER): al ser una continuación de SIN, Percepción ha sido fundamental para la comprensión del funcionamiento de las redes neuronales y los tipos de modelos de la IA junto al significado y utilidad de cada uno de sus parámetros.

4- Aprendizaje automático (APR): esta asignatura avanza sobre los conocimientos mostrados en Percepción y explica cómo crear, entrenar, realizar ajuste fino y los tipos más relevantes de redes neuronales mayoritariamente empleadas durante el desarrollo del proyecto.

5- Computación paralela (CPA): tal y como dice su nombre, CPA ha sido de gran utilidad para crear e implementar el sistema de control de la segunda capa del modelo.

6- Tecnología de Sistemas de Información en la Red (TSR): esta asignatura ha sido esencial para la comprensión del lenguaje de programación JavaScript, utilizado en el módulo del reloj inteligente.

12. Trabajo futuro

Dado que este proyecto se sitúa en el marco de la interacción humano-máquina como un cimiento sobre el cual construir futuros nuevos modelos e integraciones especializadas, se pueden realizar una gran variedad de ampliaciones, mejoras y aplicaciones de este en muchos sectores de la inteligencia artificial híbrida. Algunos de estos posibles trabajos son:

1- Uso de un modelo de IA basado en el Machine Learning: este podría, partiendo de un modelo base pre-entrenado, adaptarse y reaccionar de una forma más natural y dinámica a las diferentes acciones y estados del usuario.

2- Módulo de seguimiento ocular: este trabajo se centraría en el desarrollo de un módulo que a través del *tracking* de los ojos del usuario sobre la pantalla pueda inferir mediante el uso de mapas de calor qué zonas del monitor son observadas durante periodos de tiempo más prolongados.

3- Módulo de seguimiento del cursor: consiste en la creación de un módulo capaz de identificar las zonas de la pantalla más visitadas por el cursor del dispositivo.

4- Clasificador de entonación: se trata de una adición en la subcapa de procesado del módulo de audio, en la que se pueda detectar la forma del habla del usuario, infiriendo a partir de esta el estado emocional del mismo. Otra posible función del clasificador, es reforzar las emociones detectadas mediante la clasificación de la transcripción de texto. Un ejemplo de este refuerzo, sería la inversión de la emoción generada por la transcripción, en el caso de detectar un tono sarcástico.

5- Detección simultánea de varios usuarios: esta consiste en identificar a los diferentes usuarios e inferir sus datos de forma independiente para generar una predicción de la necesidad de interacción humano-máquina individualizada.

6- Integración y ejecución del proyecto en las nuevas unidades de procesamiento neuronal (NPU) incluidas en la reciente arquitectura de Copilot+PC: eliminar la necesidad de hardware más potente y caro como las gráficas y usar las nuevas unidades centradas en el proceso de inteligencia artificial de la reciente y popular arquitectura de Copilot+PC.

13. Bibliografía

[1] Noticia sobre el crecimiento de la IA

https://www.europapress.es/portaltic/empresas/noticia-crecimiento-exponencial-ia-desbordado-todas-previsiones-espana-closerstill-media-20230904133919.html

[2] Estadísticas clave de uso de ChatGPT

https://www.primeweb.com.mx/chatgpt-usuarios-estadisticas#:~:text=y%20mucho%20m%C3%A1s.-,Estad%C3%ADsticas%20clave%20de%20ChatGPT,llegar%20a%20la%20misma%20cifra.

[3] Noticia del instituto tecnológico IMMUNE sobre la IA híbirda
https://immune.institute/blog/inteligencia-artificial-hibrida/

[4] Entrada de Wikipedia de OpenCV

https://es.wikipedia.org/wiki/OpenCV

[5] Página oficial del modelo Whisper de OpenAI

https://openai.com/index/whisper/

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser y Illia Polosukhin, <<Attention isa ll you need>>, 12 Jun 2017

[7] GitHub de Whisper-JAX

https://github.com/sanchit-gandhi/whisper-jax

[8] GitHub de JAX

https://github.com/google/jax#installation

[9] GitHub de Faster-Whisper

https://github.com/SYSTRAN/faster-whisper

[10] GitHub de CTranslate2

https://github.com/OpenNMT/CTranslate2/

[11] Documentación OpenCV sobre el *cascade classifier*
https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html

[12] Página oficial DeepFace

https://viso.ai/computer-vision/deepface/

[13] Documentación de PyPi sobre DeepFace.

https://pypi.org/project/deepface/0.0.24/

[14] Portal de desarrollador de Fitbit.

https://dev.fitbit.com/

[15] Entrada sobre la relación del estrés y la salud cardíaca.

https://www.heart.org/en/healthy-living/healthy-lifestyle/stress-management/stress-and-heart-health

[16] Carmen Schiweck, Ali Gholamrezaei, Maxim Hellyn, Thomas, Elske Vrieze y Stephan Claes << Exhausted Heart Rate Responses to Repeated Psychological Stress in Women With Major Depressive Disorder>>, 18 April 2022.

[17] National Sleep Foundation's, <<Teens' Sleep and Mental Health Are Strongly Connected>>, 2024

[18] Página web oficial de las ODS

https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/

ANEXO

ANEXO 1: OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	**Alto**	**Medio**	**Bajo**	**No Procede**
ODS 1. **Fin de la pobreza.**				**X**
ODS 2. **Hambre cero.**				**X**
ODS 3. **Salud y bienestar.**	**X**			
ODS 4. **Educación de calidad.**		**X**		
ODS 5. **Igualdad de género.**				**X**
ODS 6. **Agua limpia y saneamiento.**				**X**
ODS 7. **Energía asequible y no contaminante.**				**X**
ODS 8. **Trabajo decente y crecimiento económico.**		**X**		
ODS 9. **Industria, innovación e infraestructuras.**	**X**			
ODS 10. **Reducción de las desigualdades**.	**X**			
ODS 11. **Ciudades y comunidades sostenibles**.				**X**
ODS 12. **Producción y consumo responsables.**			**X**	
ODS 13. **Acción por el clima.**				**X**
ODS 14. **Vida submarina.**				**X**
ODS 15. **Vida de ecosistemas terrestres.**				**X**
ODS 16. **Paz, justicia e instituciones sólidas.**				**X**
ODS 17. **Alianzas para lograr objetivos.**				**X**

Reflexión sobre la relación del TFG con los ODS y con el/los ODS más relacionados.

Los Objetivos de Desarrollo Sostenibles (ODS)[18] son una serie de 17 objetivos globales interconectados creados por la Asamblea General de las Naciones Unidas(AG-ONU) en 2015 y diseñados para ser un plan para alcanzar un futuro mejor y más sostenible para todos.

Por lo que hace a los objetivos fuertemente relacionados con este proyecto, nos podemos encontrar con los siguientes cinco:

1- Salud y bienestar: la aplicación del proyecto y sus tecnologías en el sector de la salud puede llegar a beneficiosos al poder reconocer el momento cuando personas con problemas de salud físico o mental pueden requerir de la asistencia humana o computarizada. Un ejemplo de una aplicación de este proyecto en la salud, puede ser la detección de la necesidad de generación de mensajes de soporte o notificación a un personal de ayuda, para personas con ansiedad, depresión o un trastorno mental.

2- Educación de calidad: la implementación de un modelo basado en el proyecto en el sector de la educación puede ser altamente positivo tanto para el alumnado como para el personal docente. Esto, debido a que la detección de la incertidumbre de un estudiante ante cierto contenido impartido puede impulsar una educación adaptada a cada tipo de alumno, augmentando de esta forma su potencial y nivel de estudios y disminuyendo la tasa de abandono escolar.

3- Trabajo decente y crecimiento económico: la detección de las necesidades de un empleado durante el período laboral puede ayudar a distribuir la carga de trabajo dependiendo del estado del trabajador, disminuyendo la posible ansiedad causada por el exceso de trabajo pesado en momentos de cansancio y, aumentando d esta forma su productividad, aumentando consecuentemente el crecimiento económico.

4- Industria, innovación e infraestructuras: la definición de un modelo base de interacción humano-máquina puede fomentar tanto la aparición de industria y proyectos que implementen estos avances como posibles investigaciones que, partiendo del mismo, profundicen en la creación de nuevos modelos, tecnologías o arquitecturas mejoradas.

5- Reducción de las desigualdades: este proyecto permite ofrecer una detección de las necesidades adaptada completamente al estado y perfil de usuario.

ANEXO 2: IMPLEMENTACIÓN EXTERNA DE LA LIBRERÍA DEEPFACE

Como se he comentado, en la descripción de la implementación del detector de emociones de la subcapa de procesador de video (punto 5.2.3), la librería DeepFace genera diversas dependencias de versiones con otras librerías. Más específicamente, estas son keras, numpy y las librerías que contienen. Puesto que incluso que con la compilación de la librería completamente desde cero con versiones teóricamente compatibles no se solucionó la problemática, se decidió separar su ejecución del programa principal. Esta decisión, creó una serie de desafíos y dificultades a superar en el ámbito del entorno de ejecución y la sincronización con el programa principal. A continuación, se van a describir estas problemáticas y su solución.

Entorno de ejecución

Debido a la incompatibilidad de versiones de las librerías se optó por la creación de un entorno virtual nuevo de Python separado del principal. Para reducir el gasto de memoria y evitar posibles errores al ejecutar múltiples entornos virtuales de forma simultánea, en este entorno, se realizó la instalación de únicamente las librerías básicas requeridas por DeepFace. Con el entorno ya creado, se implementó una vía mediante la cual el programa principal pudiera lanzar el subprograma en el entorno y con los parámetros adecuados. Esta solución, consiste en el desarrollo de un archivo BAT, que al inicio de la ejecución del programa principal lance el subprograma de detección de emociones mediante DeepFace. El formato BAT es un script de texto que contiene una serie de comandos que se ejecutan secuencialmente en la línea de comandos de Windows (CMD). A continuación, se describirán los diferentes comandos en orden que permiten la llamada de ejecución del subprograma desde el programa principal.

1- Guardar los argumentos enviados por el programa principal en las variables arg1 y arg2.

2- Desactivar posibles entornos virtuales activos.

3- Situar la consola de comandos en la carpeta del ejecutable del subprograma.

4- Activar el entorno virtual del subprograma.

5- Ejecutar el subprograma con los parámetros guardados en arg1 y args2

6- Desactivar el entorno virtual del subprograma

```
@echo off
set arg1=%1
set arg2=%2
REM Activar el ambiente de Conda deseado
call conda deactivate
call cd Face_Mod
call conda activate tr

REM Lanzar el script de Python
python Face.py %arg1% %arg2%

call conda deactivate

REM Mantener la ventana de comandos abierta
REM cmd /k
```

Código 5. Archivo BAT del subprograma

Sincronización con el programa principal

Como especifica su propio nombre, este problema trata de sincronizar la ejecución de la aplicación principal con el subprograma. Para hacerlo, simplemente se obliga al subprograma a esperar a la carga de los modelos y la captura del primer conjunto de imágenes. Una vez terminados estos procesos ya se puede reanudar de forma normal su ejecución.

Una vez solucionadas estas problemáticas ya no existe ningún obstáculo que impida la finalización del desarrollo del subprograma tal y como se menciona en el punto 5.2.3.

ANEXO 3: RECOPILACIÓN DEL CÓDIGO

1.

```python
def preprocess(line):
    review = re.sub('[^a-zA-Z]', ' ', line) #Leave only characters from a to z
    review = review.lower() #lower the text
    review = review.split() #turn string into list of words
    #apply Stemming
    review = [ps.stem(word) for word in review if not word in stopwords.words('spanish')] #delete stop words like I, and ,OR   review = ' '.join(review)
    #trun list into sentences
    return " ".join(review)

def clean_text(text):
    text = text.lower()
    text = re.sub('http\S+', '',text)
    text = re.sub('([@#][A-Za-z0-9_]+)|(\w+:\/\/\S+)', ' ',text)
    text = re.sub('[/(){}\[\]\|@,;]', ' ',text)
    text = re.sub('[^0-9a-z #+_]', '',text)
    text = re.sub('[\d+]', '',text)
    return text
```

Preproceso de dataset

2.

```javascript
// Función para hacer una solicitud al API de Fitbit
async function fetchData(endpoint) {
  const url = `https://api.fitbit.com/1.2/${endpoint}`;
  const headers = {
    'Authorization': `Bearer ${accessToken}`
  };

  try {
    const response = await fetch(url, { headers });
    if (!response.ok) {
      throw new Error(`HTTP error! Status: ${response.status}`);
    }
    return await response.json();
  } catch (error) {
    console.error('Error fetching data:', error);
    return null;
  }
}
```

Implementación de fetchData

3.

```python
# Python de DeepFace
    subprocess.Popen(batch_file + f"\t {str(FRAME_TIMER)} {str(BATCH_SECONDS)}")

    # Captura de vídeo
    video_mod_process = multiprocessing.Process(target=video_module)
    video_mod_process.start()
    print("Iniciado el mod de video")
    time.sleep(1)

    # Módulo de audio
    # Captura de audio
    audio_cap_process = multiprocessing.Process(target=captura_audio)
    audio_cap_process.start()
    print("Iniciada grabadora de audio")

    # Tiempo de grabar + 1 segundo de guardar - 5 segundos de carga de los modelos
    time.sleep(RECORD_SECONDS - 4)

    # Preproceso de audio
    audio_trans_process = multiprocessing.Process(target=tr_audio)
    audio_trans_process.start()
    print("Iniciado transcripción y traducción de audio")


    # Espera de la función principal
    audio_cap_process.join()
    audio_trans_process.join()
    video_mod_process.join()
```

Lanzamiento sincronizado del programa principal

4.

```
time:1.0
.timeline
0.0 - neutral
1.0 - sad
2.0 - sad
3.0 - neutral
4.0 - neutral
5.0 - neutral
6.0 - surprise
7.0 - surprise
8.0 - angry
```

Ejemplo de archivo con formato CCS