

Neural Analytics: una herramienta software basada en redes neuronales de procesado de EEGs

Bachelor's Degree in Computer Engineering

Final Degree Project

Author:

Sergio Martínez Aznar

Supervisor(s):

Antonio Molina Picó

Academic Year:

2024/2025



UNIVERSITAT
POLITÈCNICA
DE VALENCIA

CAMPUS
D'ALCOI

Para aquellos que una vez soñaron con volar alto, y lo arriesgaron todo.

Resumen

Este proyecto presenta el desarrollo de un sistema de control domótico basado en interfaces cerebro-computadora (BCI). El objetivo principal es implementar un sistema no invasivo que permita la integración de señales cerebrales con dispositivos de iluminación inteligente, específicamente bombillas TP-Link Tapo.

El sistema utiliza la diadema Brainbit como dispositivo de adquisición de señales electroencefalográficas (EEG), procesando estas señales mediante una arquitectura de software que combina Deep Learning para procesamiento y una solución desarrollada en Rust para el consumo del modelo e interacción con el sistema de iluminación.

El marco teórico aborda el desarrollo de un modelo de clasificación basado en señales EEG, explorando técnicas de procesamiento de señales y aprendizaje automático para la interpretación precisa de patrones cerebrales. Además, se profundiza en los requisitos técnicos y normativos necesarios para el desarrollo de dispositivos médicos, con especial énfasis en el estándar IEC 62304 para procesos del ciclo de vida del software de dispositivos médicos, y la implementación de sistemas operativos en tiempo real (RTOS) para garantizar la fiabilidad y seguridad del sistema.

La solución propuesta integra tecnologías modernas de procesamiento de señales cerebrales con sistemas de domótica, creando una interfaz natural e intuitiva para el control del entorno doméstico. Este proyecto representa un paso hacia la democratización de las interfaces cerebro-computadora en aplicaciones cotidianas.

Palabras clave: Interfaz cerebro-computadora, BCI, Domótica, Aprendizaje profundo, Rust, EEG, RTOS, Certificación médica

Abstract

This project presents the development of a home automation control system based on brain-computer interfaces (BCI). The main objective is to implement a non-invasive system that enables the integration of brain signals with smart lighting devices, specifically TP-Link Tapo bulbs.

The system uses the Brainbit headband as an electroencephalographic (EEG) signal acquisition device, processing these signals through a software architecture that combines PyTorch for training deep learning models and Rust for the development of the inference engine.

The theoretical framework addresses the development of a classification model based on EEG signals, exploring signal processing and machine learning techniques for accurate interpretation of brain patterns. Additionally, it delves into the technical and regulatory requirements necessary for medical device development, with special emphasis on the IEC 62304 standard for medical device software life cycle processes, and the implementation of real-time operating systems (RTOS) to ensure system reliability and safety.

The proposed solution integrates modern brain signal processing technologies with home automation systems, creating a natural and intuitive interface for controlling the home environment. This project represents a step towards the democratization of brain-computer interfaces in everyday applications.

Keywords: Brain-computer interface, BCI, Home automation, Deep learning, Rust, PyTorch, EEG, RTOS, Medical certification

Índice general

Resumen	I
Abstract	III
Índice general	V
Índice de figuras	IX
1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	2
1.3. Metodología	3
1.3.1. Fase de Investigación	3
1.3.2. Fase de Desarrollo	3
1.3.3. Fase de Validación	3
I Marco Teórico	5
2. Normativa UNE-EN 62304	7
2.1. Objetivo y Alcance	7
2.2. Clasificación del Software	8
2.3. Cumplimiento y Aplicación durante el proyecto	8
3. Regiones implicadas del cerebro	11
3.1. Introducción a las Regiones Cerebrales Funcionales	11
3.2. El Lóbulo Temporal y la Memoria Visual	12
3.3. El Lóbulo Occipital y la Percepción del Color	12
3.4. Integración entre Memoria y Percepción	13
3.5. Aplicaciones en Interfaces Cerebro-Computadora	13
3.6. Aplicaciones durante este proyecto	13
4. Sistemas operativos en Tiempo Real	15
4.1. Taxonomía de Sistemas en Tiempo Real	16
4.1.1. Sistemas de Tiempo Real Estricto	16
4.1.2. Sistemas de Tiempo Real Flexible	17
4.1.3. Consideraciones de Implementación	17
4.2. Soluciones Comerciales para Hard Real-Time	18
4.2.1. VxWorks (Wind River Systems)	18

4.2.2. QNX Neutrino (BlackBerry)	19
4.2.3. Zephyr RTOS (Linux Foundation)	20
4.3. Soluciones Comerciales para Soft Real-Time	21
4.3.1. Wind River Linux (Wind River Systems)	21
4.3.2. Poky Linux (Proyecto Yocto)	22
4.4. Elección de RTOS para el Proyecto	23
4.4.1. Requisitos Temporales del Sistema	23
4.4.2. Consideraciones Técnicas	23
4.4.3. Aspectos Regulatorios y Económicos	23
5. Modelos de Deep Learning	25
5.1. Conceptos Fundamentales	25
5.1.1. Ventanas Temporales	25
5.1.2. One-Hot Encoding	26
5.2. Arquitectura del Modelo	26
5.2.1. Función de Activación ReLU	26
5.2.2. LSTM (Long Short-Term Memory)	27
5.2.3. Función Softmax	28
5.3. Evaluación del Modelo	28
5.3.1. Métricas de Evaluación	28
II Revisión Hardware & Software	31
6. BrainBit Headset	33
6.1. Introducción	33
6.2. Características Técnicas del Dispositivo	33
6.3. Metodologías para la Adquisición y Procesamiento	34
6.3.1. Captación de Señales	34
6.3.2. Acondicionamiento de Señales	34
6.3.3. Análisis mediante Aprendizaje Profundo	35
6.4. Campos de Aplicación	35
7. Raspberry Pi 4 Model B (8GB)	37
7.1. Introducción	37
7.2. Especificaciones Técnicas	37
7.2.1. Procesador y Memoria	37
7.2.2. Requerimientos de Energía	38
7.2.3. Interfaces y Conectividad	38
7.2.4. Consideraciones Térmicas	39
7.3. Elección de este dispositivo para el Proyecto	39
III Marco Práctico	41
8. Análisis Práctico	43
8.1. Requisitos funcionales y no funcionales	43
8.1.1. Requisitos Funcionales	43
8.1.2. Requisitos No Funcionales	45

8.2. Bibliotecas Usadas	46
8.2.1. Procesamiento de Señales EEG	46
8.2.2. Interfaz Gráfica y Visualización	46
8.2.3. Inteligencia Artificial y Procesamiento de Datos	46
8.2.4. Comunicación y Control de Dispositivos	46
8.2.5. Herramientas de Concurrencia y Asincronía	47
8.2.6. Arquitectura y Diseño del Sistema	47
8.2.7. Serialización y Estructuras de Datos	47
9. Planificación Temporal	49
9.1. Cronología del Desarrollo	49
9.1.1. Fase de Investigación (Enero 2025)	49
9.1.2. Adquisición de Hardware y Estructuración (Finales de Enero 2025)	50
9.1.3. Fase de Desarrollo (Febrero - Marzo 2025)	50
9.1.4. Fase de Refinamiento del Modelo (Abril 2025)	53
9.2. Distribución Temporal	54
9.3. Diagrama de Gantt	55
9.4. Conclusiones sobre la Planificación	55
10. Entrenamiento del modelo	57
10.1. Descripción de la arquitectura	57
10.1.1. Estructura de la red neuronal	57
10.1.2. Parámetros del modelo	59
10.2. Preprocesamiento de los datos	59
10.2.1. Adquisición y estructuración del dataset	59
10.2.2. Etapas de preprocesamiento	59
10.2.3. Implementación del dataset	60
10.3. Resultados del entrenamiento	60
10.3.1. Configuración del entrenamiento	60
10.3.2. Métricas de rendimiento	61
10.3.3. Análisis de resultados	62
10.3.4. Exportación del modelo	62
11. Implementación del Core	63
11.1. Arquitectura Hexagonal	63
11.1.1. Estructura General	63
11.1.2. Puertos y Adaptadores	64
11.2. Consumo del SDK de BrainFlow	66
11.2.1. Inicialización y Configuración	66
11.2.2. Adquisición de Datos	66
11.3. Patrón Model-View-Intent (MVI)	67
11.3.1. Componentes del Patrón MVI	67
11.3.2. Flujo de Datos	68
11.3.3. Manejador de Eventos	68
11.4. Interconexión con el sistema domótico	69
11.4.1. Adaptador para Bombillas Inteligentes	69
11.4.2. Integración con la Máquina de Estados	70
11.4.3. Preparación para una futura integración con Matter	72
11.5. Implementación de la interfaz gráfica	72

11.5.1. Estructura de la GUI	72
11.5.2. Integración con el Core	73
11.6. Conclusiones y Justificación de Decisiones Arquitectónicas	74
11.6.1. Alineación con los Requisitos del Proyecto	74
11.6.2. Beneficios Observados Durante el Desarrollo	75
11.6.3. Impacto en la Calidad del Software	75
12. Validación del Prototipo	77
12.1. Marco Normativo de Validación	77
12.1.1. Normativa Aplicable	77
12.1.2. Clasificación del Software	77
12.2. Estrategia de Pruebas	78
12.2.1. Herramientas y Entorno de Pruebas	78
12.3. Pruebas Unitarias	78
12.3.1. Estrategia de Pruebas Unitarias	78
12.3.2. Pruebas Unitarias del Core (<code>neural_analytics_core</code>)	80
12.3.3. Cobertura de Código	82
12.3.4. Validación Manual de la Interfaz (<code>neural_analytics_gui</code>)	84
12.4. Pruebas de Integración	85
12.4.1. Enfoque para las pruebas de integración	85
12.4.2. Descubrimientos con las pruebas de integración	85
12.5. Pruebas del Sistema	87
12.5.1. Los casos de prueba diseñados	87
12.6. Pruebas de Seguridad	88
12.6.1. Gestión de los datos del usuario	88
12.6.2. Seguridad eléctrica	88
12.7. Gestión de Anomalías	88
12.8. Matriz de Trazabilidad	89
12.9. Conclusión de la Validación	90
Conclusiones	93
12.10 Generación de Imágenes Embebidas	94
12.11 Trabajos Futuros	94
Agradecimientos	95
Anexo I: Manual de Operador para Captura de Datos de Entrenamiento	99
Anexo II: Manual de Usuario de la Aplicación Neural Analytics	103

Índice de figuras

4.1. Ecuación de sistemas de tiempo real estricto.	16
4.2. Ecuación de sistemas de tiempo real flexible.	17
5.1. Ejemplo de One-Hot Encoding para tres colores.	26
5.2. Ecuación de la función ReLU.	26
5.3. Ecuación de la función Softmax.	28
9.1. Diagrama de Gantt del proyecto Neural Analytics	55
10.1. Arquitectura del modelo de clasificación de señales EEG	58
10.2. Curvas de entrenamiento del modelo Neural Analytics	60
10.3. Matriz de confusión del modelo en el conjunto de validación	61
10.4. Curvas ROC para cada una de las clases	61
12.1. Verificación de impedancia y calidad de señal.	100
12.2. Adquisición de datos en tiempo real.	101
12.3. Pantalla de carga inicial de la aplicación.	103
12.4. Pantalla de bienvenida e instrucciones para colocarse la diadema.	104
12.5. Pantalla de calibración de electrodos.	104
12.6. Pantalla principal de la aplicación de inferencia.	105

Capítulo 1

Introducción

El presente trabajo aborda —desde una perspectiva integradora— el diseño y desarrollo de un sistema innovador de automatización del hogar que combina tecnología de interfaz cerebro-computadora (BCI) con dispositivos de iluminación inteligentes. El propósito fundamental del proyecto se centra en el desarrollo de una solución no invasiva que posibilite el control del entorno doméstico mediante la lectura e interpretación de ondas cerebrales, empleando específicamente dispositivos TP-Link Tapo como elementos de control.

La arquitectura implementada se sustenta en dos pilares fundamentales: por una parte, el procesamiento de señales electroencefalográficas (EEG) mediante técnicas de aprendizaje profundo y, por otra, el estricto cumplimiento de la normativa UNE-EN 62304 para dispositivos y/o software médicos. Para asegurar que el sistema responda en tiempo real —un aspecto de considerable importancia en esta aplicación—, se ha optado por la utilización de Poky Linux del Proyecto Yocto como sistema operativo base. Esta elección obedece a su apreciable flexibilidad y facilidad de personalización, además de facilitar una posible migración futura a Wind River Linux en caso de que se contemple la comercialización de este proyecto como producto médico.

1.1. Motivación

El desarrollo de interfaces cerebro-computadora y sus aplicaciones potenciales representa un campo de notable relevancia en la ingeniería biomédica contemporánea. Este proyecto configura una síntesis de múltiples disciplinas: la tecnología, los sistemas operativos en tiempo real, la medicina y la innovación. El trabajo con un sistema que integra procesamiento de señales cerebrales con control domótico permite la exploración de un campo que se proyecta como revolucionario para la interacción persona-máquina en los próximos años, abordándolo desde una perspectiva tanto técnica como práctica.

La decisión de operar con actuadores domóticos comunes, concretamente bombillas inteligentes, tiene como finalidad demostrar —de manera visual e intuitiva— el funcionamiento del sistema BCI, haciendo tangible una tecnología que con frecuencia puede percibirse como abstracta o de difícil acceso. Este enfoque práctico, adicionalmente, facilita de manera considerable la comprensión del sistema y su impacto potencial en contextos cotidianos.

El aspecto normativo del proyecto, si bien presenta desafíos técnicos, representa una valiosa oportunidad para la comprensión del proceso completo de desarrollo, desde una idea

innovadora hasta un producto viable en el mercado médico. El cumplimiento de la normativa UNE-EN 62304, la implementación de un sistema en tiempo real y el desarrollo de una metodología robusta enriquecen de forma significativa la comprensión sobre las implicaciones de desarrollar tecnología médica responsable y segura.

1.2. Objetivos

Este proyecto tiene como finalidad principal el desarrollo de un sistema de control domótico basado en interfaces cerebro-computadora, con la intención de incrementar la accesibilidad de esta tecnología en entornos cotidianos.

Para la consecución de esta meta, se han establecido los siguientes objetivos específicos:

- **Cumplimiento del estándar UNE-EN 62304:** La creación de una solución que se ajuste a la normativa UNE-EN 62304, garantizando así la seguridad y fiabilidad del software médico mediante una metodología de desarrollo rigurosa y bien documentada.
- **Implementar un clasificador de señales EEG:** El desarrollo e implementación de un modelo de aprendizaje profundo para la clasificación eficiente de señales EEG, utilizando PyTorch como framework principal y con un enfoque particular en la detección de patrones asociados a la visualización de colores.
- **Desarrollar un sistema de control BCI:** La construcción de un sistema funcional que permita controlar dispositivos de iluminación inteligente mediante señales EEG, empleando la diadema Brainbit como dispositivo para la adquisición de dichas señales.
- **Generar una imagen para la RPi4:** La creación y optimización de una imagen de sistema operativo personalizada basada en Poky Linux, asegurando un entorno de ejecución con garantías de tiempo real blando para el procesamiento de señales EEG.

Estos objetivos han sido definidos considerando tanto los aspectos técnicos como normativos del proyecto, manteniendo un equilibrio entre la innovación tecnológica y la viabilidad práctica en entornos reales.

1.3. Metodología

El desarrollo del proyecto sigue una metodología estructurada en varias fases que asegura el cumplimiento de los objetivos establecidos:

1.3.1. Fase de Investigación

- Un estudio exhaustivo de la literatura disponible sobre procesamiento de señales EEG.
- Un análisis detallado de los requisitos normativos para dispositivos médicos.
- Una evaluación de las tecnologías y los frameworks disponibles en el mercado actual.

1.3.2. Fase de Desarrollo

- La implementación del modelo de clasificación en PyTorch, llevada a cabo mediante múltiples iteraciones.
- El desarrollo del motor de inferencia en Rust, lenguaje seleccionado por su rendimiento y características de seguridad.
- La integración con el SDK de BrainFlow para la adquisición de señales EEG.
- La creación de la imagen personalizada de Poky Linux, con optimización de sus componentes.

1.3.3. Fase de Validación

- Pruebas rigurosas de rendimiento y fiabilidad efectuadas en distintos escenarios.
- La verificación del cumplimiento normativo, documentando cada aspecto relevante del proceso.
- Una evaluación de la usabilidad del sistema bajo condiciones reales de operación.

La vertiente práctica del proyecto incluye una descripción detallada del proceso de desarrollo, que abarca desde el entrenamiento del modelo de aprendizaje profundo hasta su integración con el resto de los componentes del sistema.

Parte I

Marco Teórico

Capítulo 2

Normativa UNE-EN 62304

La norma **UNE-EN 62304:2007** Asociación Española de Normalización (2016) corresponde a la versión española de la norma **IEC 62304:2006/A1:2015**, la cual ha sido adoptada como norma europea EN 62304:2006/A1:2015. Esta normativa establece los requisitos para los **procesos del ciclo de vida del software en dispositivos médicos**, asegurando su desarrollo, mantenimiento y gestión de riesgos de acuerdo con estándares internacionales —un aspecto de considerable importancia para cualquier implementación rigurosa en este campo—.

2.1. Objetivo y Alcance

El propósito de la UNE-EN 62304, desarrollado por la Asociación Española de Normalización (2016), es definir un marco normativo para la **gestión del ciclo de vida del software** en dispositivos médicos, con el fin de asegurar su seguridad y eficacia.

Esta norma resulta aplicable a:

- **Software que es un dispositivo médico en sí mismo.**
- **Software embebido en dispositivos médicos.**
- **Software utilizado en entornos médicos para diagnóstico, monitoreo o tratamiento.**

El estándar establece **procesos y actividades** que los fabricantes deben seguir, incluyendo:

- Planificación del desarrollo del software.
- Análisis de requisitos y arquitectura del software.
- Implementación, integración, pruebas y verificación.
- Gestión del mantenimiento y resolución de problemas.
- Gestión del riesgo asociado al software.
- Gestión de la configuración y cambios.

2.2. Clasificación del Software

La norma clasifica el software en **tres niveles de seguridad** según el riesgo que pueda representar para el paciente o el operador. Esta clasificación, aunque en una primera aproximación pudiera parecer simplificada, encierra relevantes matices que condicionan todo el proceso de desarrollo.

- **Clase A:** El software no puede causar daño en ninguna circunstancia.
- **Clase B:** El software puede contribuir a una situación peligrosa, pero el daño potencial es **no serio**.
- **Clase C:** El software puede contribuir a una situación peligrosa con **riesgo de daño serio o muerte**.

Tras el análisis de diversos casos de estudio y precedentes, se ha constatado que la determinación de la clase correcta requiere un enfoque crítico y no meramente formal.

2.3. Cumplimiento y Aplicación durante el proyecto

Dado el carácter riguroso, y la propia necesidad de garantizar el cumplimiento de la UNE-EN 62304 Asociación Española de Normalización (2016) en este trabajo, se seguirá un enfoque basado en la **gestión del ciclo de vida del software** y la evaluación de riesgos. Se adoptarán buenas prácticas de ingeniería de software y se documentarán las actividades necesarias para cumplir con los requisitos de seguridad y calidad establecidos por la normativa. En ocasiones, esto supone un esfuerzo adicional que podría percibirse como extenso, no obstante, se considera necesario.

En este proyecto, se utilizará un **modelo de desarrollo iterativo e incremental** inspirado en metodologías ágiles como Scrum, adaptado a las necesidades específicas del desarrollo de software médico. Después de una prueba inicial con un enfoque más tradicional, se identificaron limitaciones significativas que condujeron a una reconsideración de la aproximación. Este enfoque permitirá una mayor flexibilidad y capacidad de adaptación a los cambios, así como una entrega continua de valor.

Considerando que el sistema únicamente controla el encendido y apagado de una bombilla inteligente TP-Link Tapo de manera remota, se ha clasificado el software como de **Clase A**. Esta clasificación se justifica porque el software no puede causar daño al usuario en ninguna circunstancia, puesto que:

- La diadema BrainBit es un dispositivo no invasivo de lectura pasiva.
- El control se realiza sobre una bombilla doméstica de baja tensión.
- No hay interacción directa con sistemas críticos o vitales.

A pesar de esta clasificación de bajo riesgo —podría incluso argumentarse que resulta cautelosa en exceso— se mantendrán buenas prácticas de desarrollo y documentación para asegurar la calidad del software. La experiencia acumulada sugiere que la subestimación de los aspectos de calidad en fases tempranas suele traducirse en complicaciones posteriores de difícil resolución.

Dado que esta normativa constituye un **requisito esencial** para el desarrollo de software en el mercado sanitario español y europeo, su correcta implementación asegurará la viabilidad del producto en entornos clínicos y su aceptación por parte de los organismos reguladores.

Capítulo 3

Regiones implicadas del cerebro

3.1. Introducción a las Regiones Cerebrales Funcionales

El cerebro humano constituye un sistema altamente organizado en el que diferentes regiones operan de manera especializada pero interconectada para procesar la información y generar respuestas conductuales adaptadas. Como se indica en la obra "Principios de Neurociencia" de Kandel, Jessell y Schwartz (2001), el estudio de la neurociencia ha demostrado que la división funcional del cerebro permite el análisis de la forma en que los procesos perceptivos, cognitivos y emocionales emergen de la actividad neuronal distribuida. En el presente proyecto, el enfoque se centra en dos regiones clave para la percepción y la memoria del color: el lóbulo occipital y el lóbulo temporal.

Mediante la utilización de un sistema BCI basado en EEG, se emplean electrodos en ubicaciones específicas del sistema internacional 10-20: O1 y O2 en el lóbulo occipital, responsables del procesamiento primario de la información visual, y T3 y T4 en el lóbulo temporal, donde la información visual se asocia con memorias previas y respuestas emocionales. Resultó de particular interés constatar, a través de descripciones técnicas de diversos proveedores de interfaces cerebro-ordenador, cómo estos sensores ubicados estratégicamente capturan la actividad en estas regiones específicas.

Gracias a esta organización funcional —que no presenta la rigidez que se le atribuía hace algunas décadas— es posible estudiar la relación entre percepción e imaginación del color, explorando su impacto en la memoria y la experiencia subjetiva. La literatura previa consultada al respecto confirmó esta aproximación.

3.2. El Lóbulo Temporal y la Memoria Visual

El lóbulo temporal es de importancia fundamental en la memoria visual y la asociación semántica de colores. En ocasiones, su relevancia puede ser subestimada ante la preponderancia del lóbulo occipital en el discurso sobre la visión. Los electrodos T3 y T4 captan la actividad relacionada con:

- **Hipocampo:** Responsable de la consolidación de memorias visuales y asociaciones cromáticas.
- **Corteza temporal medial:** Procesa la identificación y categoría de los colores.
- **Amígdala:** Relaciona el color con respuestas emocionales, modulando el impacto afectivo de los colores percibidos o imaginados.
- **Corteza entorinal:** Conecta el hipocampo con otras áreas corticales, permitiendo que las asociaciones cromáticas se integren en experiencias más complejas.

Cuando una persona recuerda un color, T3 y T4 reflejan la activación de estos circuitos, lo que posibilita el análisis de la relación entre percepción visual y memoria. La interpretación de estas señales no siempre resulta directa. Estudios como los de Squire y Zola-Morgan (1991) han demostrado que lesiones en el lóbulo temporal pueden afectar la capacidad de recuperar memorias visuales, lo que refuerza su papel en el almacenamiento de información sensorial.

3.3. El Lóbulo Occipital y la Percepción del Color

El lóbulo occipital es la principal región del cerebro para el procesamiento de la información visual; posiblemente la más reconocida, aunque no por ello menos compleja. Los electrodos O1 y O2 capturan la actividad en:

- **Corteza visual primaria (V1):** Primer procesamiento del color y detección de longitudes de onda.¹
- **V4 (corteza visual asociativa):** Especializada en la interpretación y categorización de colores, estableciendo una conexión funcional con las áreas temporales para asignar significados semánticos.

Estudios como los de Brouwer y Heeger (2013) han demostrado que la actividad en V4 puede ser utilizada para clasificar diferentes colores percibidos o imaginados, lo que reforza la validez de los electrodos O1 y O2 para el análisis de patrones cromáticos en EEG. Conforme avanzaba la investigación, se constató que este respaldo científico era de gran importancia para fundamentar las decisiones de diseño adoptadas.

¹Es importante distinguir entre O1/O2, que son las *posiciones de los electrodos* según el sistema internacional 10-20, y V1/V4, que son *áreas funcionales del córtex visual* (designaciones de Brodmann) cuya actividad es registrada por dichos electrodos.

3.4. Integración entre Memoria y Percepción

El procesamiento del color en el cerebro, presentado de forma simplificada —con el fin de facilitar su comprensión— sigue un flujo distribuido:

1. O1/O2 detectan y analizan las características del color percibido o imaginado.
2. La información es enviada a T3/T4 para su comparación con recuerdos previos y asociaciones emocionales.
3. Se generan asociaciones semánticas y afectivas, determinando la experiencia subjetiva del color.

Este proceso no siempre ocurre de manera estrictamente lineal. Estudios como los de Rissman y Wagner (2012) han mostrado que patrones de activación en el lóbulo temporal pueden predecir si un individuo reconoce un color previamente visto, lo que refuerza la idea de que los recuerdos cromáticos tienen una representación neural clara. Después de la revisión de estos trabajos, resultó evidente la solidez del fundamento para el enfoque adoptado.

3.5. Aplicaciones en Interfaces Cerebro-Computadora

La integración de la percepción y la memoria del color en un sistema BCI presenta varias aplicaciones potenciales. Algunas de ellas consideradas con perspectivas favorables son:

- Diferenciar entre percepción real e imaginada de un color.
- Implementar sistemas BCI basados en selección cromática.
- Explorar la relación entre color, memoria y emociones para aplicaciones en neurotecnología.

De este modo, se puede explorar mediante una interfaz cerebro-computadora cómo la percepción y la memoria del color se integran en la experiencia subjetiva, permitiendo así su interpretación por un computador y que este pueda realizar acciones en función de la información recibida. Se asientan así las bases teóricas en relación al BCI de este proyecto.

3.6. Aplicaciones durante este proyecto

Los electrodos O1, O2, T3 y T4 capturan información fundamental sobre la percepción y la memoria del color. Su combinación permite el análisis de la reconstrucción mental de colores y su impacto en la experiencia subjetiva, conformando la base del sistema BCI desarrollado en este proyecto. Estas ubicaciones específicas fueron seleccionadas tras una revisión detallada de las especificaciones técnicas proporcionadas por varios fabricantes de dispositivos BCI, así como de la literatura académica relevante que ha sido citada a lo largo de este capítulo.

Capítulo 4

Sistemas operativos en Tiempo Real

Los sistemas operativos en tiempo real (RTOS) Siewert and Pratt (2016) representan una rama del software orientada a garantizar la ejecución de tareas en plazos temporales específicos. El concepto mismo de computación en tiempo real surge de la necesidad de procesar y responder a eventos externos con restricciones de tiempo bien definidas. En consecuencia, la corrección del sistema no solo depende de la lógica de los resultados, sino también del momento exacto en que estos se producen.

Las características que diferencian a un RTOS de sistemas operativos convencionales son:

- **Determinismo:** Propiedad fundamental donde, dada una entrada y un estado inicial, el sistema devolverá siempre la misma salida en tiempos predecibles.
- **Concurrencia:** Capacidad para gestionar varias tareas en espacios temporales limitados sin comprometer plazos críticos.
- **Interrupciones:** Mecanismos para responder a eventos externos de forma rápida y predecible.
- **Planificación:** Algoritmos que controlan el orden y tiempo de ejecución de tareas según su criticidad temporal.

En el campo de los sistemas embebidos, un RTOS actúa como intermediario entre el hardware y las operaciones de control. El ejemplo clásico de los controladores de vuelo en aeronaves resulta particularmente ilustrativo, donde cualquier fallo en los tiempos de respuesta puede tener efectos catastróficos. Estos sistemas también se encuentran en satélites artificiales, que suelen operar con varios RTOS en paralelo para controlar tanto las funciones del vehículo como los instrumentos científicos, trabajando de forma coordinada.

Al investigar las aplicaciones de RTOS, se han identificado numerosos sectores donde son vitales: en entornos industriales para el control de procesos críticos; en aviónica para sistemas de navegación (sujetos a certificaciones de alta exigencia); en defensa para sistemas de radar; y —aspecto relevante para este proyecto— en el sector médico, donde dispositivos como marcapasos, respiradores y bombas de infusión requieren respuestas predecibles para garantizar la seguridad del paciente.

4.1. Taxonomía de Sistemas en Tiempo Real

La clasificación básica de RTOS se fundamenta en la criticidad de sus restricciones temporales. Esta taxonomía, propuesta por Liu y Layland en 1973 Siewert and Pratt (2016), distingue principalmente entre sistemas estrictos (*hard real-time*) y flexibles (*soft real-time*). La comprensión de estas categorías evolucionó significativamente al observar aplicaciones reales, donde la línea divisoria entre ambas no siempre es tan nítida como se describe en la literatura teórica.

4.1.1. Sistemas de Tiempo Real Estricto

Los sistemas estrictos (**hard real-time**) no toleran ninguna desviación en sus plazos temporales. El incumplimiento de un plazo se considera un fallo crítico del sistema —una consideración de peso dadas sus aplicaciones en entornos críticos—. Su comportamiento se puede expresar matemáticamente como:

$$\forall t \in T : R(t) \leq D(t) \quad (4.1)$$

Figura 4.1: Ecuación de sistemas de tiempo real estricto.

donde $R(t)$ es el tiempo de respuesta y $D(t)$ el plazo máximo permitido.

Algunos de los casos de uso más comunes para estos sistemas son:

- **Control nuclear:** Donde la precisión temporal es crítica para la seguridad.
- **Sistemas ABS:** Responden en microsegundos para evitar accidentes.
- **Robótica quirúrgica:** Necesitan sincronización precisa durante operaciones.

La implementación requiere algoritmos **preemptivos** con prioridades estáticas, donde el tiempo máximo de ejecución debe ser predecible. Normalmente se utilizan Rate Monotonic (RM) o Earliest Deadline First (EDF). Durante las pruebas iniciales de implementación, se constató que garantizar un determinismo absoluto en hardware estándar es prácticamente imposible, lo que llevó a replantear algunas decisiones de diseño.

4.1.2. Sistemas de Tiempo Real Flexible

Los sistemas flexibles (**soft real-time**) toleran cierta variabilidad en sus plazos, funcionando con un modelo estadístico. Esto se puede representar como:

$$P(R(t) \leq D(t)) \geq p_{min} \quad (4.2)$$

Figura 4.2: Ecuación de sistemas de tiempo real flexible.

donde p_{min} es el nivel mínimo aceptable de cumplimiento.

Las aplicaciones más comunes son:

- **Streaming multimedia:** La pérdida ocasional de algunos frames no deteriora significativamente la experiencia.
- **Redes de monitorización:** Toleran retrasos esporádicos en la actualización de datos.
- **Trading algorítmico:** Se prioriza el rendimiento promedio sobre garantías absolutas.

Estos sistemas usan planificadores basados en **tiempo compartido** con prioridades dinámicas. Para la interfaz cerebro-computadora de este proyecto, dicho enfoque resultó más adecuado, ya que pequeñas variaciones en los tiempos no afectan de manera sustancial la experiencia. Tras un análisis detallado de los requisitos temporales, se evidenció que este modelo no solo cumplía con lo necesario, sino que ofrecía un mejor equilibrio entre complejidad y prestaciones.

4.1.3. Consideraciones de Implementación

La decisión entre sistemas estrictos o flexibles depende de varios factores:

- **Riesgos:** ¿Cuáles son las consecuencias si se incumple un plazo temporal?
- **Hardware disponible:** Limitaciones de procesamiento y memoria.
- **Presupuesto:** Balance entre garantías temporales y complejidad.
- **Regulaciones:** Requisitos de certificación según el ámbito de aplicación.

4.2. Soluciones Comerciales para Hard Real-Time

4.2.1. VxWorks (Wind River Systems)

VxWorks es una referencia en sistemas embebidos críticos, especialmente en aviación, aeroespacial y el sector médico. Al iniciar su estudio, la documentación asociada pudo parecer extensa. Sus características principales son:

Certificaciones y Normativas

- DO-178C Level A para sistemas aeroespaciales.
- IEC 62304 para dispositivos médicos.
- ISO 26262 ASIL D para automoción.

Características Técnicas

- **Kernel:** Microkernel determinista con latencias ≤ 50 ns.
- **Memoria:** MMU con protección y aislamiento.
- **Scheduler:** 256 niveles de prioridad y herencia.
- **IPC:** Comunicación con latencia determinista.
- **Multiproceso:** Soporte para SMP y AMP con aislamiento.

Al evaluar VxWorks, resultó notable su utilización en aplicaciones como rovers de Marte y dispositivos médicos certificados. No obstante, los costes de licencia para obtener un SDK personalizado se consideraron elevados para un proyecto en fase de prototipo.

4.2.2. QNX Neutrino (BlackBerry)

QNX Neutrino, adquirido por BlackBerry en 2010, destaca por su microkernel distribuido y fiabilidad. Resulta interesante que BlackBerry, tras una reorientación de su mercado de móviles, mantenga este producto tecnológico avanzado:

Arquitectura

- **Microkernel:** Núcleo de tamaño reducido, inferior a 100KB.
- **Servicios:** Arquitectura modular en espacio de usuario.
- **IPC:** Mensajería con mecanismo copy-on-write.
- **Recuperación:** Reinicio de componentes sin afectar al sistema.

Características Avanzadas

- **Tiempo Real:** Latencias garantizadas $\leq 100 \mu\text{s}$.
- **Seguridad:** Modelo de seguridad con ASLR.
- **Certificaciones:** IEC 61508 SIL3, IEC 62304 Clase C.

QNX presentó un atractivo considerable por su uso en dispositivos médicos. Su principal inconveniente no radicó tanto en el coste —BlackBerry ofrece licencias para desarrollo— sino en la limitada compatibilidad con Rust, lo que dificultaba la integración de las librerías utilizadas en este proyecto. Tras intentos de integración, se determinó que la inversión de tiempo adicional no resultaba justificable y se descartó esta opción.

4.2.3. Zephyr RTOS (Linux Foundation)

Zephyr es la alternativa open-source para sistemas embebidos críticos. Este proyecto, iniciado por Wind River y posteriormente donado a la Linux Foundation, ha experimentado un crecimiento rápido y cuenta con un ecosistema de desarrolladores activo. Se consideró una opción interesante para el prototipo:

Diseño y Arquitectura

- **Kernel:** Configurable como monolítico o microkernel.
- **Tamaño:** Desde 8KB hasta 512KB según configuración.
- **Scheduler:** Hasta 32 niveles de prioridad.
- **Certificación:** En proceso para IEC 61508 SIL 3/4.

Características Destacadas

- **Drivers:** Más de 350 controladores para periféricos.
- **Redes:** Soporte para protocolos IoT (BLE, Thread, LoRaWAN).
- **Seguridad:** Subsistema con aislamiento.
- **Desarrollo:** Herramientas de depuración avanzadas.

Aunque Zephyr resultaba atractivo y no implicaba costes económicos directos, presentaba también complicaciones relativas al soporte de librerías en Rust. Dadas sus características, se optó por explorar otras opciones que pudieran contar con el núcleo de Linux y así aprovechar el ecosistema de librerías existente.

4.3. Soluciones Comerciales para Soft Real-Time

4.3.1. Wind River Linux (Wind River Systems)

Wind River Linux es una solución comercial basada en Yocto para sistemas con requisitos temporales flexibles. Al ser descubierta, se percibió como una versión más accesible de VxWorks, con un enfoque contemporáneo:

Características Principales

- **Base:** Kernel Linux 5.10 LTS con parche PREEMPT_RT.
- **Certificaciones:** ISO 9001:2015 y precertificación IEC 62304.
- **Seguridad:** Monitorización de vulnerabilidades y mitigación.
- **Conformidad:** Documentación SBOM y Open Chain 2.1.

Capacidades Industriales

- **Soporte:** Mantenimiento garantizado por 5 años, extensible.
- **Actualizaciones:** Sistema OTA mediante OSTree.
- **Validación:** Más de 60.000 tests automatizados.
- **Servicios:** Soporte técnico y consultoría.

Inicialmente, se consideró seriamente Wind River Linux por su precertificación IEC 62304, un factor importante para dispositivos médicos. Sin embargo, los costes de licencia y soporte resultaron demasiado elevados para la fase actual del proyecto, por lo que se optó por una alternativa más económica para el diseño del prototipo.

4.3.2. Poky Linux (Proyecto Yocto)

Poky es la distribución de referencia del Proyecto Yocto para sistemas Linux embebidos con capacidades de tiempo real flexible. Al comenzar su utilización, se percibió como una herramienta muy flexible, aunque su curva de aprendizaje fue más pronunciada de lo esperado inicialmente:

Características Técnicas

- **Kernel:** Linux con parche PREEMPT_RT.
- **Tiempo Real:** Latencias configurables según necesidades.
- **Optimización:** Control fino sobre tamaño y rendimiento.
- **Personalización:** Capacidad para eliminar componentes innecesarios.

Consideraciones de Desarrollo

- **Mantenimiento:** Actualización manual de parches de seguridad.
- **Soporte:** Basado en comunidad, sin garantías comerciales.
- **Certificación:** Requiere proceso propio.
- **Validación:** Es necesario desarrollar pruebas específicas.

Durante el análisis, Poky demostró ser la opción con el mejor equilibrio entre capacidades, flexibilidad y costes. Permitió la creación de una imagen personalizada ajustada exactamente a los requisitos del proyecto. Aunque carece de certificaciones como Wind River Linux, su base en Yocto ofrece la posibilidad de migrar a una solución más robusta si el proyecto avanza hacia la comercialización.

4.4. Elección de RTOS para el Proyecto

Se eligió Poky Linux como sistema operativo para este proyecto por varios motivos:

4.4.1. Requisitos Temporales del Sistema

El proyecto requiere un sistema flexible (**soft real-time**) porque:

- La detección de patrones EEG para la identificación de colores no necesita latencias críticas.
- Un retraso en la respuesta no pone en peligro al usuario.
- El control de iluminación con TP-Link Tapo tolera cierta variabilidad.

Analizando el peor escenario (un retraso al cambiar la iluminación), se concluyó que las consecuencias no justificaban la complejidad de un sistema estricto.

4.4.2. Consideraciones Técnicas

Poky Linux presenta ventajas importantes para esta aplicación:

- **Flexibilidad:** Permite crear una imagen personalizada según los requisitos.
- **Compatibilidad:** Se integra de forma sencilla con las librerías consumidas durante el desarrollo del proyecto.
- **Tiempo Real:** El parche PREEMPT_RT proporciona las garantías temporales necesarias.

4.4.3. Aspectos Regulatorios y Económicos

Aunque Poky Linux no cuenta con precertificaciones como Wind River Linux, su base en Yocto facilita la migración a Wind River Linux si se requieren certificaciones para la comercialización. Esta decisión no fue inmediata; implicó semanas de análisis de ventajas y desventajas, incluyendo consultas con profesionales del sector médico. Finalmente, esta estrategia permite la optimización de costes iniciales y el mantenimiento de la flexibilidad en la fase actual, con la previsión de un camino hacia la certificación si el proyecto se convierte en un producto comercial.

Esta combinación de factores hace que Poky Linux sea la opción más adecuada para esta fase del proyecto, ofreciendo un buen equilibrio entre rendimiento, flexibilidad y costes. Como sucede con frecuencia en ingeniería, la solución óptima no siempre es la más avanzada tecnicamente, sino aquella que mejor se adapta al problema concreto que se está resolviendo.

Capítulo 5

Modelos de Deep Learning

A través de este capítulo, se describen los modelos de Deep Learning Raschka et al. (2022) implementados en el proyecto, así como los conceptos fundamentales y la arquitectura de cada uno. También se detallan las métricas de evaluación y el proceso de validación cruzada utilizado para evaluar su rendimiento.

La selección de estos modelos constituyó un proceso de análisis técnico detallado. Tras la evaluación de múltiples arquitecturas, se optó por aquellas que demostraron una capacidad superior para detectar patrones temporales en señales EEG, especialmente en ventanas cortas —un requisito fundamental para la clasificación en tiempo real demandada por el sistema—.

Aunque seguramente ya existen trabajos previos que abordan la clasificación de señales EEG mediante Deep Learning, en este proyecto se buscó una implementación desde cero, con el objetivo de comprender, y aprender, a fondo como se puede llegar a construir un modelo de clasificación adaptado a este contexto. Por tanto, se ha evitado el uso de modelos preentrenados o bibliotecas de alto nivel que abstraigan demasiado los detalles del proceso de entrenamiento y ajuste.

5.1. Conceptos Fundamentales

5.1.1. Ventanas Temporales

Las ventanas temporales en el procesamiento de señales EEG representan segmentos discretos de tiempo durante los cuales se recopilan datos. En este proyecto, estas ventanas capturan patrones de actividad cerebral asociados al pensamiento de diferentes colores. La longitud de la ventana se reveló como un parámetro de alta sensibilidad: si era demasiado corta, no capturaba suficiente información para la clasificación; si era excesivamente larga, introducía latencias consideradas inaceptables para una aplicación en tiempo real.

Tras pruebas exhaustivas con diferentes configuraciones, se determinó que ventanas de 62 muestras ofrecían el equilibrio más adecuado para el sistema. Este hallazgo difirió de las expectativas iniciales basadas en la literatura técnica, donde se suelen sugerir ventanas más extensas, pero fue validado empíricamente como la configuración de mayor eficiencia para esta implementación específica.

5.1.2. One-Hot Encoding

El One-Hot Encoding Raschka et al. (2022) es una técnica de preprocessamiento que transforma etiquetas categóricas (en este caso, colores) en vectores binarios. Por ejemplo, para tres colores:

Color	Vector One-Hot
Rojo	[1, 0, 0]
Verde	[0, 1, 0]
Azul	[0, 0, 1]

Figura 5.1: Ejemplo de One-Hot Encoding para tres colores.

Esta técnica es de gran utilidad cuando se trabaja con datos categóricos sin relación ordinal entre sí. A diferencia de la codificación de etiquetas ordinales, donde se asigna un valor numérico a cada categoría según un orden predefinido, One-Hot Encoding crea una nueva columna para cada categoría posible.

Por ejemplo, si existe una columna “color” con las opciones “rojo”, “verde” y “azul”, esta técnica la transforma en tres columnas nuevas. Cada fila tendrá un 1 en la columna de su color correspondiente y 0 en las demás.

Inicialmente, existió una indecisión entre utilizar esta técnica o una codificación ordinal más simple, pues en ciertos experimentos preliminares se habían observado comportamientos similares con ambas. Sin embargo, conceptualmente el One-Hot Encoding representa de manera más fidedigna la naturaleza de los datos —no existe una relación ordinal inherente entre los colores— por lo que se optó por implementar este enfoque, considerado más robusto.

5.2. Arquitectura del Modelo

5.2.1. Función de Activación ReLU

La función ReLU (Rectified Linear Unit) se consolidó como un componente esencial del modelo implementado, debido a características que la hacen especialmente adecuada:

$$f(x) = \max(0, x) \tag{5.1}$$

Figura 5.2: Ecuación de la función ReLU.

ReLU es una función de activación no lineal que aborda el problema del desvanecimiento del gradiente, presente en funciones como tanh o sigmoide. Este problema ocurre cuando, por ejemplo, para valores de entrada grandes ($z_1 = 20$ y $z_2 = 25$), las funciones tanh y sigmoide producen salidas prácticamente idénticas ($\sigma(z_1) \approx \sigma(z_2) \approx 1,0$) debido a su comportamiento asintótico.

Las principales ventajas que condujeron a la selección de ReLU son:

- **Gradiente Constante:** Para valores positivos de entrada, la derivada es siempre 1, lo que evita el desvanecimiento del gradiente.

- **Eficiencia Computacional:** Su implementación es simple y rápida, requiriendo solo una comparación con cero.
- **No Linealidad:** A pesar de su simplicidad, mantiene la capacidad para aprender funciones complejas.
- **Activación Dispersa:** Produce activaciones dispersas, ya que cualquier entrada negativa se convierte en cero.

Durante la fase de experimentación, se probaron diferentes funciones de activación, incluyendo Leaky ReLU y ELU, pero no se observaron mejoras significativas en el rendimiento que justificaran la complejidad adicional. La implementación de ReLU estándar resultó ser la opción más práctica y eficiente para el caso de uso específico de este proyecto.

5.2.2. LSTM (Long Short-Term Memory)

Las LSTM fueron diseñadas para superar el problema del desvanecimiento del gradiente, frecuente en redes neuronales recurrentes (RNN) estándar. Este problema tiene lugar por la multiplicación repetida de gradientes durante la retropropagación a través del tiempo (BPTT), lo que provoca que los gradientes se vuelvan extremadamente pequeños (desvanecimiento) o grandes (explosión).

La primera aproximación al proyecto utilizaba RNNs convencionales, pero pronto se encontraron limitaciones al procesar secuencias temporales largas. Las LSTM ofrecían una solución efectiva a este problema, aunque con una mayor complejidad computacional —un factor que inicialmente generó cierta preocupación, dado el requisito de ejecución en tiempo real—.

Para una mejor comprensión de este problema, considérese una RNN con una sola unidad oculta. La derivada de la función de pérdida respecto a la entrada neta posee un factor multiplicativo que puede volverse muy pequeño o muy grande según el peso recurrente. Si este peso es menor que 1, el gradiente se desvanece; si es mayor que 1, explota.

Las LSTM abordan esta cuestión mediante celdas de memoria que mantienen información durante períodos prolongados. Cada celda incluye tres tipos de puertas: olvido, entrada y salida.

- **Puerta de Olvido (Forget Gate):** Decide qué información descartar de la memoria. Se calcula mediante:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (5.2)$$

- **Puerta de Entrada (Input Gate):** Decide qué nueva información almacenar. Se calcula como:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (5.3)$$

- **Valor Candidato (Candidate Value):** Representa la nueva información potencial. Se calcula con:

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \quad (5.4)$$

- **Puerta de Salida (Output Gate):** Decide qué parte de la memoria se usará para la salida. Se calcula así:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (5.5)$$

La celda de memoria se actualiza de la siguiente forma:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \quad (5.6)$$

Y la salida se calcula como:

$$h_t = o_t \cdot \tanh(C_t) \quad (5.7)$$

Estas ecuaciones pueden parecer complejas en una primera instancia, pero la intuición subyacente es relativamente clara: las LSTM aprenden a controlar qué información recordar, actualizar u olvidar en cada paso temporal. Al implementar esta arquitectura, fue posible capturar dependencias temporales en las señales EEG que resultaron fundamentales para distinguir patrones asociados a diferentes colores.

5.2.3. Función Softmax

La función Softmax es una versión suavizada de argmax; en lugar de proporcionar un único índice de clase, ofrece la probabilidad de cada una. Esto permite calcular probabilidades significativas en configuraciones multiclase.

En Softmax, la probabilidad de que una muestra con entrada neta z pertenezca a la clase i se calcula con un término de normalización en el denominador, que suma las funciones lineales ponderadas exponencialmente:

$$p(z) = \sigma(z) = \frac{e^{z_i}}{\sum_{j=1}^M e^{z_j}} \quad (5.8)$$

Figura 5.3: Ecuación de la función Softmax.

Las probabilidades resultantes suman 1, como es de esperar. También es destacable que la etiqueta predicha es la misma que al aplicar argmax a la salida logística.

Durante el desarrollo del modelo, se consideró brevemente la utilización de una capa sigmoide final con entrenamiento independiente para cada clase (enfoque one-vs-all), pero la implementación con Softmax resultó más depurada y directa, además de proporcionar interpretaciones probabilísticas más intuitivas de las predicciones.

5.3. Evaluación del Modelo

5.3.1. Métricas de Evaluación

Para evaluar el rendimiento del modelo, se utilizó un conjunto de métricas complementarias:

- **Accuracy:** Proporción de predicciones correctas sobre el total. Aunque es una métrica intuitiva, no siempre refleja el rendimiento real cuando las clases están desbalanceadas.

- **Matriz de Confusión:** Visualización detallada de aciertos y errores por clase. Esta herramienta resultó de particular utilidad para identificar patrones específicos de confusión entre colores, lo que permitió ajustar el preprocesamiento de señales para mejorar la discriminación en casos problemáticos.
- **ROC-AUC:** Área bajo la curva ROC para evaluación multiclase. Esta métrica se consideró especialmente valiosa por su robustez ante el desbalanceo de clases, un problema que se manifestó en algunas sesiones de recopilación de datos donde ciertos colores mostraban frecuencias de aparición variables.

La combinación de estas métricas proporcionó una visión integral del rendimiento del modelo en diferentes escenarios y condiciones, guiando el proceso iterativo de mejora hasta alcanzar resultados considerados satisfactorios para la aplicación en tiempo real.

Parte II

Revisión Hardware & Software

Capítulo 6

BrainBit Headset

6.1. Introducción

La elección del dispositivo EEG adecuado para este proyecto resultó ser un proceso más complejo de lo anticipado inicialmente. Después de una revisión de varias opciones disponibles, la decisión recayó en el **BrainBit Neurotechnology Systems LLC** (2024) por razones que trascienden las meras especificaciones técnicas.

El proceso de selección se centró en identificar un dispositivo que ofreciera tanto precisión como practicidad. No resultaba útil un equipo que solo funcionara de manera óptima en condiciones de laboratorio —categoría en la que se encuentran numerosos dispositivos en el mercado— sino que se requería un sistema que pudiera ser empleado en condiciones reales, sin una complejidad de configuración excesiva en cada uso. En última instancia, la decisión se redujo a encontrar un equilibrio adecuado entre calidad científica y facilidad de uso, una convergencia que no siempre se materializa en el ámbito de las interfaces cerebro-computadora.

6.2. Características Técnicas del Dispositivo

El BrainBit es un casco EEG portátil que utiliza **electrodos secos**. Esta característica en particular simplificó considerablemente los procedimientos, ya que la no utilización de gel conductor agiliza el proceso general —especialmente cuando es necesario repetir mediciones varias veces al día y se busca minimizar la preparación de los electrodos en cada ocasión—.

Las especificaciones de mayor relevancia para el proyecto fueron:

- **Canales EEG:** 4 canales (T3, T4, O1, O2).
- **Frecuencia de muestreo:** 250 Hz.
- **Interfaz de comunicación:** Bluetooth Low Energy (BLE).
- **Tiempo de uso continuo:** Hasta 12 horas.
- **Filtro de ruido integrado:** Este componente evitó la necesidad de implementar una cantidad significativa de procesamiento adicional.

- **Ubicación de electrodos:** Sistema 10-20 estándar, con sensores en **O1** y **O2** para actividad occipital.

La configuración de los electrodos O1 y O2, junto con los electrodos T3 y T4, fue un factor determinante para la selección del dispositivo. Su posicionamiento sobre la región occipital y temporal se ajustaba de manera precisa a los requerimientos para la detección de patrones visuales relacionados con colores. No siempre es sencillo encontrar un dispositivo que disponga de los electrodos necesarios en las ubicaciones exactas —en ocasiones, la disposición de los sensores por parte de los fabricantes puede parecer arbitraria—.

6.3. Metodologías para la Adquisición y Procesamiento

El desarrollo práctico siguió un protocolo estructurado en tres fases, aunque es preciso admitir que, en la práctica, fue necesario un grado de improvisación mayor al planeado inicialmente.

6.3.1. Captación de Señales

La colocación correcta de los electrodos **O1** y **O2** sobre la región occipital demostró ser más crítica de lo supuesto. Aunque teóricamente parecía un procedimiento simple, requirió varias sesiones experimentales hasta definir un protocolo que funcionara consistentemente. Existe un componente de frustración al constatar que procedimientos descritos como triviales en la literatura científica pueden demandar una considerable destreza en su aplicación práctica.

El SDK proporcionado por el fabricante resulta adecuado para una primera toma de contacto, pero pronto se evidenció la necesidad de una solución más personalizada, dadas las exigencias del proyecto. Las herramientas básicas de captura en tiempo real incluidas no cubrían la totalidad de los requisitos específicos, ya que estas se orientaban a modelos básicos para inferir, por ejemplo, el nivel de estrés del paciente. Consecuentemente, se desarrolló una interfaz propia que se integrara de manera más efectiva con el sistema de procesamiento y clasificación diseñado, una medida necesaria para alcanzar la funcionalidad deseada.

Uno dei problemi che generò mayor preocupación fue garantizar la estabilidad en la conexión de los electrodos. Para abordar esta cuestión, se desarrolló una interfaz de calibración que muestra en tiempo real la calidad del contacto de cada electrodo. Esta herramienta resultó de gran utilidad durante las sesiones experimentales, y los detalles de su implementación se encuentran en el Capítulo 11.

6.3.2. Acondicionamiento de Señales

Aunque el BrainBit ya incluye un sistema de filtrado de calidad notable, las señales EEG requirieron algunos procesamientos adicionales. Afortunadamente, la calidad base de las señales proporcionadas por el dispositivo es elevada, lo que permitió mantener estos procesamientos en un nivel de complejidad relativamente bajo —una circunstancia favorable, ya que el procesamiento de señales EEG puede convertirse rápidamente en una tarea intrincada—.

El enfoque principal del acondicionamiento fue la normalización de amplitudes y una segmentación temporal adecuada para el análisis posterior. Aunque puedan parecer pasos básicos, fueron esenciales para asegurar la consistencia de los datos de entrada al sistema. En ocasiones, las tareas aparentemente más simples son las que requieren mayor tiempo para su perfeccionamiento.

6.3.3. Análisis mediante Aprendizaje Profundo

En esta fase, se implementaron modelos de **aprendizaje profundo** especializados en el procesamiento de series temporales EEG. Estos modelos fueron entrenados para identificar patrones específicos relacionados con la visualización mental de colores —concretamente rojo y verde—.

Los fundamentos teóricos de estos modelos se explican en detalle en el Capítulo 5, y todo lo concerniente al entrenamiento práctico se describe en el Capítulo 10.

6.4. Campos de Aplicación

Las posibilidades de aplicación de esta tecnología son considerablemente amplias, aunque muchas se encuentran aún en fase experimental —o directamente en un terreno donde la viabilidad práctica es incierta—:

- Desarrollo de interfaces cerebro-máquina para asistir a personas con diversidad funcional. Esto podría habilitar nuevas formas de comunicación e interacción con la tecnología, si bien se reconoce que existe un camino considerable por recorrer hasta alcanzar aplicaciones verdaderamente prácticas.
- Sistemas de control en entornos estériles —tanto médicos como industriales— donde el contacto físico podría comprometer la asepsia o la seguridad. En este ámbito se percibe un potencial de aplicación más inmediato.
- Aplicaciones en realidad aumentada y virtual, que permitirían formas de interacción más naturales e intuitivas. No obstante, surge la reflexión sobre la conveniencia de controlar todos los aspectos mediante el pensamiento, o si esta aspiración se debe meramente a la disponibilidad tecnológica.

Durante el desarrollo del proyecto, la consideración de estas aplicaciones potenciales contribuyó a mantener la motivación al enfrentar obstáculos técnicos que parecían de difícil resolución. La perspectiva del impacto real que este tipo de interfaces podría tener —incluso en casos muy específicos— justifica el tiempo y esfuerzo invertidos en su desarrollo y optimización.

Capítulo 7

Raspberry Pi 4 Model B (8GB)

7.1. Introducción

La elección de la Raspberry Pi 4 Model B Raspberry Pi Foundation (2020) para este proyecto no fue una decisión inmediata. Inicialmente, se habían considerado otras opciones con mayor potencia —incluso se evaluaron algunos mini-PCs x86—, pero finalmente este ordenador de placa única resultó convincente por razones eminentemente prácticas. El equilibrio entre rendimiento, consumo y facilidad de desarrollo demostró ser adecuado para los requerimientos del proyecto, una conclusión que, es preciso admitir, se consolidó solo después de varias semanas de trabajo y algunos desafíos iniciales.

El factor determinante para la selección fue el descubrimiento de que incorpora un procesador ARM de 64 bits con un funcionamiento notablemente eficiente (superior al esperado, con toda sinceridad), una cantidad de memoria RAM considerable —especialmente en el modelo de 8GB, que fue el elegido tras una deliberación entre este y el de 4GB—, y la práctica totalidad de las interfaces requeridas por el diseño. No obstante, la mayor seguridad provino de la verificación de que su arquitectura ARM cuenta con un soporte sólido por parte de los principales fabricantes. Este aspecto era fundamental, dadas experiencias previas con incompatibilidades en otros proyectos, una situación que se deseaba evitar.

7.2. Especificaciones Técnicas

El modelo de 8GB de la Raspberry Pi 4 se basa en una arquitectura de hardware que ofreció una relación prestaciones-precio valorada positivamente, aunque se reconoce que las expectativas iniciales eran moderadas:

7.2.1. Procesador y Memoria

- CPU: Quad-Core ARM Cortex-A72 (64 bits) a 1.5GHz —si bien puede alcanzar hasta 1.8GHz en ciertas condiciones, un detalle que se conoció posteriormente—.
- GPU: VideoCore VI compatible con OpenGL ES 3.0 —suficiente para las necesidades del proyecto, sin esperar un rendimiento gráfico excepcional—.
- Memoria RAM: 8 GB LPDDR4 SDRAM. Este fue el factor que motivó la inversión adicional en este modelo específico, especialmente tras experiencias previas de

limitaciones de memoria en otros desarrollos.

7.2.2. Requerimientos de Energía

La Raspberry Pi 4 Model B requiere una fuente de alimentación de 5V y 3A a través de USB-C. Para configuraciones que incluyan dispositivos USB adicionales, se recomienda el uso de una fuente con mayor capacidad —una lección aprendida tras algunos problemas de estabilidad inesperados durante las primeras pruebas—. Considerar este aspecto es fundamental, especialmente en un proyecto donde la estabilidad es un requisito principal. De hecho, se invirtió una considerable cantidad de tiempo en la depuración de un supuesto problema de software que, en realidad, se debía a una fuente de alimentación con corriente insuficiente para el sistema completo.

7.2.3. Interfaces y Conectividad

- Red:

- Gigabit Ethernet (compatible con PoE mediante un módulo adicional, aunque esta opción no fue implementada —quizás su consideración más detenida hubiera sido pertinente—).
- Wi-Fi 802.11 b/g/n/ac de doble banda (2.4 GHz y 5.0 GHz). Su funcionamiento general es adecuado, aunque la potencia de la señal podría ser superior.
- Bluetooth 5.0 con BLE, que resultó idóneo para la conexión con el dispositivo BrainBit —uno de los componentes cuya integración funcionó de manera óptima desde el inicio—.

- Almacenamiento:

- Ranura para tarjeta microSD. Se recomienda encarecidamente la inversión en una tarjeta de alta calidad —las opciones de bajo coste generaron problemas de rendimiento y algunos incidentes relacionados con la corrupción de datos—.

- Puertos USB:

- 2 puertos USB 3.0.
- 2 puertos USB 2.0.

- Vídeo y Audio:

- 2 puertos micro-HDMI con soporte hasta 4K a 60Hz.
- Salida de audio analógico y vídeo compuesto mediante conector TRRS de 3.5 mm.

- Expansión:

- Conector GPIO de 40 pines compatible con modelos anteriores.
- Conector CSI para cámaras.
- Conector DSI para pantallas.

7.2.4. Consideraciones Térmicas

El sistema de gestión térmica de la Raspberry Pi 4 reduce automáticamente la frecuencia y el voltaje del procesador durante períodos de baja demanda, lo cual opera eficientemente para el ahorro de energía y la prevención del sobrecalentamiento. Ahora bien, bajo cargas intensivas y, especialmente, en condiciones de temperatura ambiente elevada, es necesario añadir disipación adicional, como disipadores o ventiladores. Esta necesidad se confirmó de manera directa cuando el sistema comenzó a reiniciarse durante las primeras sesiones de pruebas intensivas, y tomó cierto tiempo identificar que se trataba de un problema térmico y no de software.

7.3. Elección de este dispositivo para el Proyecto

El modelo de 8GB de la Raspberry Pi 4 demostró ser una solución versátil y compacta para los requerimientos del desarrollo, aunque es preciso reconocer que hubo momentos en los que se sopesó la posibilidad de optar por una plataforma de mayor potencia. Disponer de 8GB de RAM proporcionó el margen suficiente para ejecutar aplicaciones complejas sin una preocupación constante por las limitaciones de memoria, y el rendimiento general se mostró adecuado para procesos en tiempo real —al menos para las exigencias específicas de este proyecto—.

El aspecto más valorado finalmente fue la compatibilidad con sistemas operativos en tiempo real y las distribuciones Linux con las que ya se tenía familiaridad. La posibilidad de utilizar herramientas conocidas aceleró considerablemente el desarrollo, una ventaja no anticipada en su totalidad al inicio, pero que resultó de gran importancia cuando los plazos comenzaron a ser más ajustados.

Parte III

Marco Practico

Capítulo 8

Análisis Práctico

8.1. Requisitos funcionales y no funcionales

La definición de los requisitos del sistema resultó ser un proceso más complejo de lo anticipado inicialmente. Se había considerado que consistiría simplemente en elaborar una lista de las funcionalidades deseadas para el sistema; sin embargo, se constató que muchos requisitos no se manifiestan hasta que se inicia la implementación y surgen problemas no previstos. Configurando así un proceso marcadamente iterativo, donde cada fase del desarrollo revelaba nuevas necesidades.

8.1.1. Requisitos Funcionales

Los requisitos funcionales definen esencialmente las operaciones que el sistema debe realizar. A continuación, se presentan aquellos identificados a partir del análisis del código final:

- **RF-01:** El sistema debe permitir la conexión y desconexión con el dispositivo EEG (electroencefalograma).
- **RF-02:** El sistema debe capturar los datos crudos (raw) de los canales T3, T4, O1 y O2 del dispositivo EEG.
- **RF-03:** El sistema debe obtener datos de impedancia del dispositivo EEG para verificar la calidad de la señal.
- **RF-04:** El sistema debe permitir el cambio entre distintos modos de trabajo del dispositivo EEG.
- **RF-05:** El sistema debe implementar un modelo de inferencia para predecir el color en el que está pensando el usuario.
- **RF-06:** El sistema debe distinguir entre al menos dos colores (rojo y verde) y un estado 'desconocido'.
- **RF-07:** El sistema debe controlar la activación y desactivación de una bombilla inteligente.
- **RF-08:** El sistema debe proporcionar una interfaz gráfica que permita visualizar el estado de la conexión del dispositivo EEG.

- **RF-09:** El sistema debe permitir visualizar la señal EEG en tiempo real.
- **RF-10:** El sistema debe mostrar al usuario las predicciones realizadas por el modelo de inferencia.

8.1.2. Requisitos No Funcionales

La definición de los requisitos no funcionales representó el aspecto más intrincado. En esta categoría se incluyen temas como el rendimiento, la seguridad y el cumplimiento de normativas —aspectos que con frecuencia se omiten en las fases iniciales por ser menos evidentes—. Algunos de estos requisitos no habían sido siquiera considerados hasta el inicio de las primeras pruebas del sistema, momento en el cual se observó que ciertos componentes no operaban según lo esperado. Los requisitos finalmente identificados son:

- **RNF-01: Normativa:** El sistema debe cumplir con la norma UNE-EN 62304:2007 para software de dispositivos médicos.
- **RNF-02: Tiempo Real:** El sistema debe operar en tiempo real blando para asegurar una respuesta adecuada a los cambios en la señal EEG.
- **RNF-03: Fiabilidad:** El sistema debe validar la calidad de las señales EEG mediante los datos de impedancia antes de realizar predicciones.
- **RNF-04: Portabilidad:** El sistema debe poder ejecutarse en una Raspberry Pi.
- **RNF-05: Seguridad:** El sistema debe garantizar la privacidad y seguridad de los datos biométricos del usuario.
- **RNF-06: Interoperabilidad:** El sistema debe integrarse con dispositivos domóticos estándar (bombillas inteligentes).
- **RNF-07: Mantenibilidad:** El sistema debe seguir un diseño hexagonal (puertos y adaptadores) para facilitar su mantenimiento y pruebas.
- **RNF-08: Usabilidad:** La interfaz gráfica debe ser intuitiva y proporcionar retroalimentación clara sobre el estado del sistema.
- **RNF-09: Escalabilidad:** La arquitectura debe permitir la inclusión de nuevos tipos de predicciones o dispositivos de salida.
- **RNF-10: Rendimiento:** El sistema debe ser capaz de procesar y analizar señales EEG con una latencia mínima.

8.2. Bibliotecas Usadas

La selección de las bibliotecas adecuadas requirió un tiempo considerable de investigación. Inicialmente, se pensó que sería suficiente con identificar las opciones más populares; sin embargo, se constató que muchas tecnologías que parecían idóneas en teoría, posteriormente no operaban correctamente con los requisitos de tiempo real o no compilaban para la arquitectura ARM. Este fue un caso donde la teoría y la práctica presentaron divergencias significativas.

El proyecto utiliza una combinación de bibliotecas para diferentes propósitos. Algunas fueron seleccionadas desde el inicio, mientras que otras se incorporaron a medida que surgían necesidades específicas no previstas. A continuación, se organizan según su función principal:

8.2.1. Procesamiento de Señales EEG

- **BrainFlow**: Biblioteca para la adquisición y procesamiento de datos de dispositivos de electroencefalografía (EEG). Permite la comunicación con el dispositivo BrainBit y la captura de datos en tiempo real.

8.2.2. Interfaz Gráfica y Visualización

- **slint**: Framework para la creación de interfaces gráficas, con soporte para Rust y con características de alta eficiencia.
- **plotters**: Biblioteca para la creación de gráficos y visualizaciones en Rust, utilizada para mostrar las señales EEG en tiempo real.

8.2.3. Inteligencia Artificial y Procesamiento de Datos

- **PyTorch**: Framework de aprendizaje profundo utilizado para el entrenamiento del modelo de clasificación de señales EEG.
- **ONNX**: Formato estándar para la representación de modelos de aprendizaje automático que permite la interoperabilidad entre diferentes frameworks.
- **tract-onnx**: Biblioteca en Rust para la ejecución de modelos ONNX, utilizada para las inferencias en tiempo real.
- **ndarray**: Biblioteca para el procesamiento de arrays multidimensionales en Rust, utilizada para el preprocesamiento de datos.

8.2.4. Comunicación y Control de Dispositivos

- **tapo**: Cliente en Rust para controlar dispositivos inteligentes Tapo, utilizado para la bombilla inteligente que responde a las señales cerebrales del usuario.
- **presage**: Biblioteca de gestión de eventos y mensajería para la comunicación entre componentes.

8.2.5. Herramientas de Concurrencia y Asincronía

- **tokio**: Runtime asíncrono para Rust que facilita la programación concurrente, esencial para manejar múltiples flujos de datos en tiempo real.
- **async-trait**: Permite la definición de traits asíncronos en Rust.

8.2.6. Arquitectura y Diseño del Sistema

- **statig**: Biblioteca para la implementación del patrón máquina de estados en Rust, utilizada para gestionar el ciclo de vida de la aplicación.
- **once_cell**: Para la implementación de singlettons en Rust, utilizado en la gestión de recursos compartidos.

8.2.7. Serialización y Estructuras de Datos

- **serde**: Framework de serialización/deserialización para Rust, utilizado para el intercambio de datos entre componentes.
- **chrono**: Biblioteca para el manejo de fechas y tiempos en Rust.

Capítulo 9

Planificación Temporal

9.1. Cronología del Desarrollo

La organización del desarrollo del proyecto siguió una estructura metodológica dividida en fases claramente diferenciadas. Se implementó una planificación temporal que permitió el desarrollo progresivo de todos los componentes del sistema, considerando las dependencias técnicas entre módulos y los requisitos normativos específicos. La metodología adoptada facilitó el control de la complejidad técnica y garantizó el cumplimiento de los objetivos establecidos dentro del cronograma propuesto.

9.1.1. Fase de Investigación (Enero 2025)

Durante enero se establecieron las bases teóricas y técnicas del proyecto mediante una fase de investigación exhaustiva. Se realizó el análisis de las tecnologías disponibles y se evaluaron los recursos necesarios para el desarrollo:

- **Estudio de arquitecturas de redes neuronales:** Se efectuó un análisis detallado de diferentes arquitecturas de aprendizaje profundo, con un enfoque particular en las redes LSTM (Long Short-Term Memory) por su adecuación para el procesamiento de secuencias temporales como las señales EEG. Se evaluaron las características específicas que hacían estas arquitecturas apropiadas para el procesamiento de datos neurofisiológicos.
- **Evaluación de dispositivos EEG:** Se compararon diferentes dispositivos disponibles en el mercado, analizando criterios como precisión, número de canales, facilidad de integración y compatibilidad con bibliotecas de software existentes. Se seleccionó el dispositivo BrainBit por ofrecer un equilibrio adecuado entre funcionalidad, costo y disponibilidad de documentación técnica.
- **Investigación de bibliotecas de adquisición de datos:** Se evaluaron múltiples bibliotecas para la captura de datos EEG, seleccionando finalmente BrainFlow debido a su compatibilidad con diversos dispositivos y la robustez de su documentación técnica, factores de alta relevancia para el desarrollo de aplicaciones médicas.
- **Estudio de normativas aplicables:** Se analizaron las regulaciones y estándares pertinentes para dispositivos médicos, prestando especial atención a la norma

UNE-EN 62304:2007 para software de dispositivos médicos. Este análisis permitió establecer los requisitos de desarrollo y documentación correspondientes.

- **Estudio de plataformas para implementación:** Se evaluaron diferentes opciones de hardware para la implementación del sistema, analizando capacidad de procesamiento en tiempo real y adecuación para aplicaciones médicas. Esta evaluación determinó los requisitos técnicos para la selección posterior del hardware.

9.1.2. Adquisición de Hardware y Estructuración (Finales de Enero 2025)

Completada la fase de investigación, se procedió a la adquisición del hardware necesario y la estructuración del proyecto. Esta fase se centró en materializar las decisiones técnicas tomadas durante la investigación y establecer la arquitectura base del sistema:

- **Adquisición del dispositivo BrainBit:** Se obtuvo el dispositivo EEG seleccionado como fuente principal de datos neurofisiológicos para el proyecto. La adquisición se realizó considerando los criterios técnicos establecidos durante la fase de investigación.
- **Obtención de la Raspberry Pi 4:** Se seleccionó esta plataforma como sistema de implementación principal debido al equilibrio entre potencia de procesamiento, portabilidad y adecuación para el desarrollo de prototipos médicos.
- **Adquisición de bombillas inteligentes Tapo:** Se obtuvieron los dispositivos de control domótico necesarios para implementar las respuestas del sistema a las señales cerebrales procesadas. Esta selección se basó en la facilidad de integración y compatibilidad con los protocolos de comunicación requeridos.
- **Definición de la arquitectura del software:** Se diseñó la estructura general del proyecto adoptando una arquitectura hexagonal (puertos y adaptadores) para garantizar modularidad, testabilidad y facilidad de mantenimiento. Esta decisión arquitectónica se fundamentó en los requisitos normativos de la UNE-EN 62304.
- **Planificación de componentes del sistema:** Se definieron los módulos constituyentes del proyecto: neural_analytics_data para la captura de datos, neural_analytics_model para entrenamiento e inferencia, neural_analytics_core para la lógica central, y neural_analytics_gui para la interfaz de usuario. Esta modularización facilitó el desarrollo paralelo y la validación independiente de componentes.

9.1.3. Fase de Desarrollo (Febrero - Marzo 2025)

Durante febrero y marzo se implementaron todos los componentes del sistema siguiendo la arquitectura previamente definida. Esta fase constituyó el núcleo del desarrollo técnico, caracterizada por la implementación simultánea de múltiples módulos y su posterior integración:

Desarrollo del Programa de Extracción de Datos (neural_analytics_data)

Se desarrolló este módulo como primer componente del sistema, estableciendo la interfaz entre el dispositivo de adquisición y el resto del sistema. El desarrollo de este módulo presentó varios desafíos técnicos relacionados con la captura fiable de datos EEG:

- **Integración con BrainFlow:** Se implementó la interfaz con la biblioteca BrainFlow para la captura de datos del dispositivo BrainBit. Se desarrolló un sistema de comunicación robusto que garantiza la adquisición continua y fiable de señales EEG.
- **Diseño de protocolos de captura:** Se desarrollaron rutinas estructuradas para la captura de datos EEG durante tareas de imaginación de colores. Se establecieron protocolos temporales precisos para garantizar la consistencia de las muestras y facilitar el posterior entrenamiento del modelo.
- **Implementación de procesamiento de señales:** Esta etapa implicó una inmersión en un área de conocimiento que resultaba novedosa. Se desarrollaron funciones para filtrar y preprocesar las señales raw, si bien es preciso admitir que inicialmente existía una comprensión limitada del proceso. Se invirtió un tiempo considerable en el estudio de transformadas de Fourier y filtros digitales —conceptos teóricamente notables pero de aplicación práctica compleja—.
- **Almacenamiento y etiquetado de datos:** Se implementó un sistema para almacenar y etiquetar automáticamente los datos capturados. La automatización de este proceso se consideró de alta importancia para evitar errores manuales. No obstante, lograr un etiquetado fiable requirió un esfuerzo superior al estimado.

Desarrollo del Programa de Entrenamiento del Modelo (neural_analytics_model)

Esta fase presentó desafíos imprevistos. Durante la resolución de problemas en la extracción de datos, surgió la idea de desarrollar el módulo de entrenamiento en paralelo. En retrospectiva, la simultaneidad de estas dos tareas complejas podría parecer cuestionable, pero el resultado no fue tan caótico como podría haberse previsto.

Una estrategia funcional consistió en probar el pipeline de entrenamiento con los primeros datos extraídos, incluso si eran parciales. Esto permitió la detección temprana de problemas —como la constatación de que el preprocesamiento eliminaba información crucial para la clasificación—. Son aspectos que frecuentemente solo se descubren al trabajar con datos reales.

El módulo de entrenamiento se convirtió en un entorno de experimentación para diversas configuraciones:

- **Diseño de arquitectura LSTM:** Esta tarea implicó adentrarse en un terreno poco explorado previamente. Fue necesario diseñar una red neuronal basada en capas LSTM, optimizada específicamente para clasificar patrones en señales EEG. Inicialmente, la definición del punto de partida presentó dificultades. Se dedicaron varias semanas a la revisión de literatura científica y a la prueba de configuraciones, en algunos casos de forma exploratoria, hasta comprender los parámetros de mayor impacto. Hubo períodos en los que una arquitectura parecía óptima, para luego descubrir su bajo rendimiento con conjuntos de datos diferentes.
- **Implementación en PyTorch:** Se eligió PyTorch por su reputación de ser más intuitivo que TensorFlow. Sin embargo, la adaptación a su sintaxis desde otros frameworks representó una curva de aprendizaje pronunciada. Las primeras semanas fueron intensas; tareas conceptualmente simples requerían horas de esfuerzo debido a la falta de familiaridad con el manejo de tensores. Hubo ocasiones en que se

dedicaron noches enteras a lograr que un entrenamiento básico se ejecutara sin errores.

- **Rutinas de entrenamiento y validación:** Este fue uno de los aspectos más laboriosos del proceso. Se desarrollaron procedimientos para entrenar el modelo de manera eficiente y validarlos con diferentes conjuntos de datos. No obstante, conseguir la estabilidad del entrenamiento se convirtió en un desafío recurrente. El modelo exhibía variabilidad en su comportamiento —en ocasiones convergía adecuadamente en pocas épocas, mientras que en otras no lograba aprender patrones coherentes a pesar de los ajustes de parámetros—. Esta situación generaba frustración.
- **Exportación a formato ONNX:** Esta tarea requirió más esfuerzo del previsto. Se implementó la conversión para utilizar el modelo posteriormente en Rust, pero la compatibilidad entre PyTorch y ONNX demostró ser más problemática de lo que la documentación sugería. Ciertas operaciones que funcionaban correctamente en PyTorch no eran interpretadas adecuadamente por ONNX. Se invirtieron varios días en reescribir partes del modelo para asegurar una exportación exitosa.

Desarrollo del Core del Sistema (`neural_analytics_core`)

Esta sección se erigió como el componente central y más demandante del proyecto. Consumió una cantidad de tiempo considerablemente mayor a la estimada, a pesar de que las previsiones iniciales ya contemplaban un margen amplio.

La correcta implementación de la arquitectura hexagonal se convirtió en un foco de atención principal durante semanas. Hubo períodos en los que la reflexión sobre puertos, adaptadores y diagramas de arquitectura ocupaba gran parte del pensamiento cotidiano.

Se estableció la arquitectura núcleo del sistema implementando los principios de diseño hexagonal para garantizar la separación entre lógica de negocio e infraestructura. Se desarrolló un sistema modular que permite la interoperabilidad eficiente entre componentes:

- **Implementación de puertos y adaptadores:** Se definieron interfaces claras para todos los componentes externos (puertos) y sus implementaciones concretas (adaptadores), siguiendo estrictamente los principios de la arquitectura hexagonal. Esta implementación garantiza la flexibilidad del sistema y facilita futuras extensiones o modificaciones.
- **Desarrollo del dominio central:** Se implementó la lógica de negocio central de forma completamente independiente de las infraestructuras externas. Esta separación permite el desarrollo, testing y mantenimiento de cada componente de forma aislada, cumpliendo con los requisitos normativos de trazabilidad.
- **Sistema de eventos:** Se desarrolló un mecanismo de comunicación basado en eventos utilizando la biblioteca presage para Rust. Este sistema permite el desacoplamiento efectivo entre componentes y facilita la gestión del flujo de información a través del sistema.
- **Máquina de estados:** Se implementó una máquina de estados para gestionar el ciclo de vida de la aplicación y las transiciones entre diferentes modos operativos. Esta implementación garantiza el comportamiento predecible del sistema y facilita la gestión de estados de error y recuperación.

- **Servicio de inferencia:** Se desarrolló un servicio para la ejecución del modelo en tiempo real utilizando tract-onnx para inferencia en Rust. Esta implementación garantiza el rendimiento requerido para el procesamiento en tiempo real de señales EEG.
- **Control de dispositivos domóticos:** Se implementó la integración con dispositivos inteligentes a través de la biblioteca tapo. Esta integración permite el control remoto efectivo de dispositivos de iluminación basado en las predicciones del modelo.

Desarrollo de la Interfaz Gráfica (`neural_analytics_gui`)

Se desarrolló la interfaz gráfica del sistema utilizando tecnologías modernas para crear una experiencia de usuario eficiente y accesible. El desarrollo se centró en proporcionar visualización en tiempo real de las señales EEG y control intuitivo del sistema:

- **Diseño de interfaz con Slint:** Se utilizó el framework Slint para crear una interfaz moderna y eficiente. Esta elección tecnológica permitió el desarrollo de una interfaz responsive con renderizado de alto rendimiento, adecuada para el procesamiento en tiempo real.
- **Visualización de señales EEG:** Se implementó la representación gráfica de las señales en tiempo real utilizando la biblioteca plotters. Esta funcionalidad permite el monitoreo continuo de la calidad de las señales y facilita la calibración del sistema.
- **Integración con el core:** Se estableció la comunicación bidireccional con el núcleo del sistema para la visualización de estados y resultados en tiempo real. Esta integración garantiza la sincronización efectiva entre el frontend y el backend del sistema.
- **Interfaz para calibración:** Se desarrollaron vistas específicas para la calibración del dispositivo y verificación de impedancias. Estas funcionalidades son críticas para garantizar la calidad de las señales EEG y la fiabilidad de las predicciones.
- **Visualización de predicciones:** Se implementó un sistema para mostrar las predicciones del modelo en tiempo real de manera clara y comprensible. La interfaz incluye indicadores de confianza y visualización de estados del sistema para facilitar la interpretación de resultados.

9.1.4. Fase de Refinamiento del Modelo (Abril 2025)

Durante abril se implementó una fase intensiva de refinamiento del modelo de aprendizaje profundo. Esta fase se caracterizó por la optimización sistemática de todos los parámetros del modelo y la ampliación significativa del dataset de entrenamiento:

- **Ampliación del dataset:** Se organizaron sesiones adicionales de captura de datos EEG para aumentar significativamente tanto el tamaño como la diversidad del dataset disponible. Se incluyeron grabaciones de múltiples usuarios en diferentes condiciones temporales para mejorar la capacidad de generalización del modelo.
- **Diversificación de casos de uso:** Se expandieron los escenarios de prueba para incluir variaciones en los enfoques mentales utilizados por diferentes usuarios durante las tareas de imaginación de colores. Esta diversificación permitió al modelo adaptarse a diferentes estrategias cognitivas individuales.

- **Reentrenamiento del modelo:** Se realizaron múltiples iteraciones de reentrenamiento del modelo LSTM utilizando los datos ampliados. Se optimizaron sistemáticamente los hiperparámetros para maximizar la precisión del modelo manteniendo el rendimiento en tiempo real.
- **Validación cruzada:** Se implementaron técnicas rigurosas de validación cruzada para verificar la consistencia del rendimiento del modelo con diferentes subconjuntos de datos. Esta metodología garantiza la fiabilidad de las métricas de rendimiento obtenidas.
- **Ajuste de umbrales de confianza:** Se refinaron los mecanismos para determinar cuándo una predicción debe clasificarse como "desconocida." en lugar de forzar una clasificación incorrecta. Esta optimización mejoró significativamente la fiabilidad percibida del sistema durante operación en condiciones de incertidumbre.

9.2. Distribución Temporal

La distribución temporal del proyecto refleja la complejidad técnica de cada fase y las dependencias entre componentes. Los tiempos de desarrollo se organizaron de manera eficiente para optimizar el uso de recursos:

Fase	Período	Duración
Investigación	Enero 2025	4 semanas
Adquisición y Estructuración	Finales de Enero 2025	1 semana
Desarrollo del Programa de Extracción	Febrero 2025	2 semanas
Desarrollo del Programa de Entrenamiento	Febrero 2025	2 semanas
Desarrollo del Core del Sistema	Febrero - Marzo 2025	4 semanas
Desarrollo de la Interfaz Gráfica	Marzo 2025	2 semanas
Pruebas Iniciales	Finales de Marzo 2025	2 semanas
Refinamiento del Modelo	Abril 2025	4 semanas

Cuadro 9.1: Distribución temporal del desarrollo del proyecto Neural Analytics

9.3. Diagrama de Gantt

A continuación, se presenta la representación visual de cómo se solaparon las diferentes fases. Este tipo de diagramas permite observar la superposición temporal de las tareas, un aspecto que no resulta tan evidente al examinar únicamente las fechas:

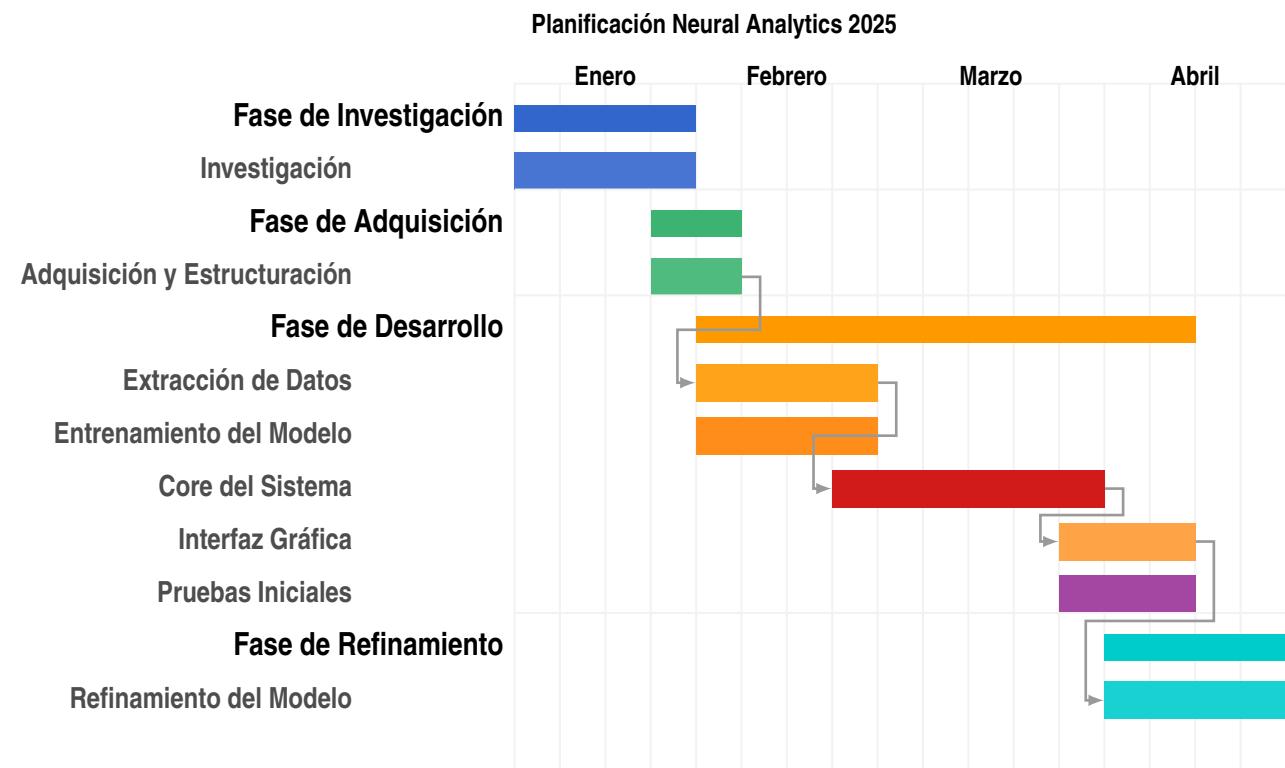


Figura 9.1: Diagrama de Gantt del proyecto Neural Analytics

9.4. Conclusiones sobre la Planificación

La evaluación retrospectiva de la planificación temporal del proyecto permite identificar aspectos exitosos y áreas de mejora para futuros desarrollos similares. La metodología implementada resultó adecuada para alcanzar los objetivos establecidos dentro del marco temporal propuesto.

- **Duración de la fase de desarrollo core:** El desarrollo del núcleo del sistema, siguiendo estrictamente los principios de arquitectura hexagonal, requirió una inversión temporal considerable. Esta inversión se justificó por los beneficios obtenidos en términos de mantenibilidad, testabilidad y facilidad para realizar pruebas exhaustivas posteriores.
- **Paralelización de tareas:** La estructuración del desarrollo en componentes modulares desde el inicio permitió el desarrollo simultáneo de varios módulos. Esta estrategia optimizó significativamente el tiempo total de desarrollo y facilitó la gestión de la complejidad del proyecto.
- **Importancia de la fase de refinamiento:** La fase de abril demostró ser de mayor criticidad de lo inicialmente estimado. La ampliación del dataset con muestras

diversas y representativas incrementó significativamente el rendimiento del modelo, justificando la inversión temporal adicional requerida.

- **Iteración continua:** El enfoque iterativo resultó fundamental, especialmente durante la etapa de refinamiento del modelo. Cada incorporación de datos permitió ajustes incrementales que mejoraron gradualmente el rendimiento del sistema.
- **Áreas de mejora identificadas:** Para futuros desarrollos similares, se recomienda ampliar significativamente la fase de pruebas con usuarios reales para obtener mayor retroalimentación sobre la usabilidad del sistema. También se sugiere continuar expandiendo el dataset con muestras más diversas y representativas.

La implementación de la arquitectura hexagonal, aunque inicialmente más costosa en tiempo de desarrollo, proporcionó una base sólida para cumplir efectivamente con los requisitos normativos y facilitar futuras extensiones del sistema.

La dedicación de un mes completo al refinamiento intensivo del modelo fue fundamental para alcanzar niveles de precisión adecuados para un dispositivo de uso médico. La experiencia obtenida sugiere que esta fase crítica debería considerarse con mayor peso temporal en planificaciones futuras de proyectos similares.

Capítulo 10

Entrenamiento del modelo

Tras completar el desarrollo de la arquitectura del sistema, se procedió al entrenamiento del modelo para la clasificación de señales EEG. Esta fase representó un hito de alta relevancia para el proyecto, donde se validarían los fundamentos teóricos y la viabilidad práctica de la aproximación propuesta.

Este capítulo documenta de manera sistemática el proceso de entrenamiento del modelo, incluyendo las decisiones arquitectónicas, la metodología experimental y los resultados obtenidos. El desarrollo de esta fase presentó diversos desafíos técnicos que requirieron adaptaciones iterativas en la aproximación inicial.

10.1. Descripción de la arquitectura

La selección de la arquitectura neuronal constituyó una decisión técnica fundamental del proyecto. Se optó por una arquitectura híbrida que combina redes LSTM (Long Short-Term Memory) con capas densas, elección fundamentada en la capacidad demostrada de las LSTM para procesar secuencias temporales complejas como las señales EEG.

El sistema se diseñó para procesar y clasificar secuencias temporales de datos neurofisiológicos, utilizando ventanas de 62 puntos temporales como entrada base.

10.1.1. Estructura de la red neuronal

Se implementó una arquitectura híbrida específicamente diseñada para procesar señales EEG de los cuatro canales seleccionados: T3, T4, O1 y O2. El sistema realiza una clasificación en tres categorías: RED, GREEN y TRASH, manteniendo un enfoque simplificado que facilita la validación inicial del concepto.

La arquitectura final se estableció tras un proceso iterativo de experimentación con diferentes configuraciones de capas y parámetros de red.

Los componentes principales son:

- **Capa LSTM:** Una capa LSTM con 64 unidades que captura patrones temporales en las señales. La configuración utiliza `batch_first=True` para optimizar la forma de entrada (`batch_size`, `seq_length`, `features`).
- **Capas densas:** Después de la LSTM, se encuentran varias capas:

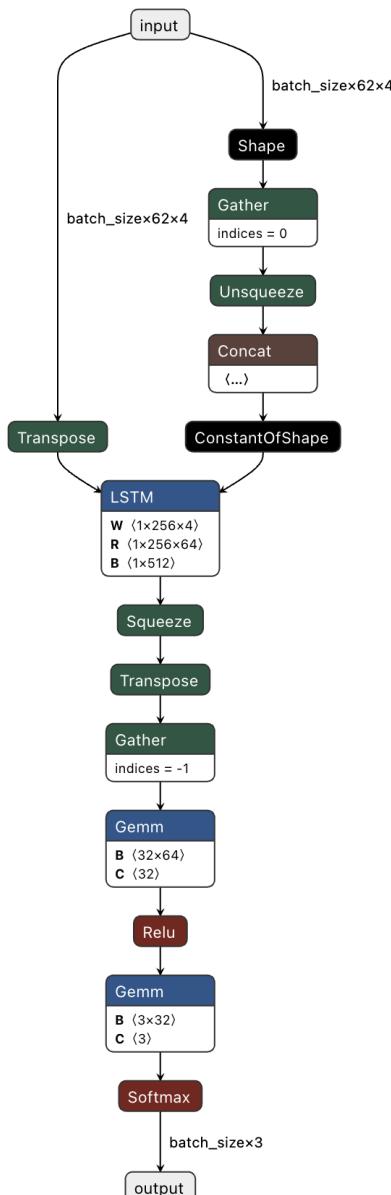


Figura 10.1: Arquitectura del modelo de clasificación de señales EEG

- Primera capa densa: Reduce de 64 a 32 unidades.
- Activación ReLU: Añade no-linealidad.
- Segunda capa densa: Proyecta a 3 neuronas de salida.
- Softmax: Normaliza las salidas como probabilidades.

El flujo de datos se describe a continuación:

1. Ingresá una secuencia de 62 puntos, cada uno con 4 características.
2. La LSTM procesa esta secuencia y extrae 64 características por punto.
3. Se toma el último estado de la secuencia.
4. Este estado atraviesa las capas densas con activación ReLU.

5. La capa final con Softmax proporciona la probabilidad para cada clase.

10.1.2. Parámetros del modelo

Los principales parámetros son:

- **INPUT_SIZE = 4:** Corresponde a los cuatro canales.
- **HIDDEN_SIZE = 64:** Unidades en la capa LSTM.
- **NUM_CLASSES = 3:** Las tres categorías de clasificación.
- **WINDOW_SIZE = 62:** Tamaño de la ventana para las secuencias.
- **BATCH_SIZE = 64:** Número de muestras por lote durante el entrenamiento.

10.2. Preprocesamiento de los datos

El preprocesamiento constituye una fase crítica para garantizar la calidad de las entradas al modelo. Se implementaron múltiples etapas de procesamiento, desde la captura inicial hasta la generación de ventanas deslizantes optimizadas.

10.2.1. Adquisición y estructuración del dataset

El dataset presenta la siguiente estructura:

- **Organización por clases:** Archivos CSV almacenados en directorios según la clase:
 - `/red/`: Datos registrados mientras el usuario piensa en el color rojo.
 - `/green/`: Datos registrados mientras el usuario piensa en el color verde.
 - `/trash/`: Datos que no se corresponden con las categorías anteriores.
- **Formato:** Cada archivo CSV contiene mediciones de los canales T3, T4, O1 y O2 en columnas.

10.2.2. Etapas de preprocesamiento

El proceso se encuentra encapsulado en la función `neural_analytics_preprocessor` y realiza las siguientes operaciones:

1. **Normalización:** Escala los canales EEG al rango [0,1].
2. **Extracción de etiquetas:** Obtiene la clase a partir del nombre del directorio.
3. **Codificación one-hot:** Convierte las etiquetas categóricas a vectores binarios:
 - `red`: [1, 0, 0]
 - `green`: [0, 1, 0]
 - `trash`: [0, 0, 1]
4. **Ventanas deslizantes:** Para cada archivo CSV, crea ventanas con solapamiento.

10.2.3. Implementación del dataset

Se desarrolló una clase `NeuralAnalyticsDataset` que hereda de `Dataset`, propia de PyTorch. Esta implementación:

- Recorre el directorio y procesa los archivos CSV.
- Aplica el preprocessamiento definido.
- Almacena las ventanas y sus etiquetas correspondientes.
- Convierte los datos a tensores de PyTorch.
- Implementa los métodos `__len__` y `__getitem__`.

Durante la inicialización, se implementa una división automática en conjuntos de entrenamiento (80 %) y validación (20 %) utilizando la función `train_test_split`.

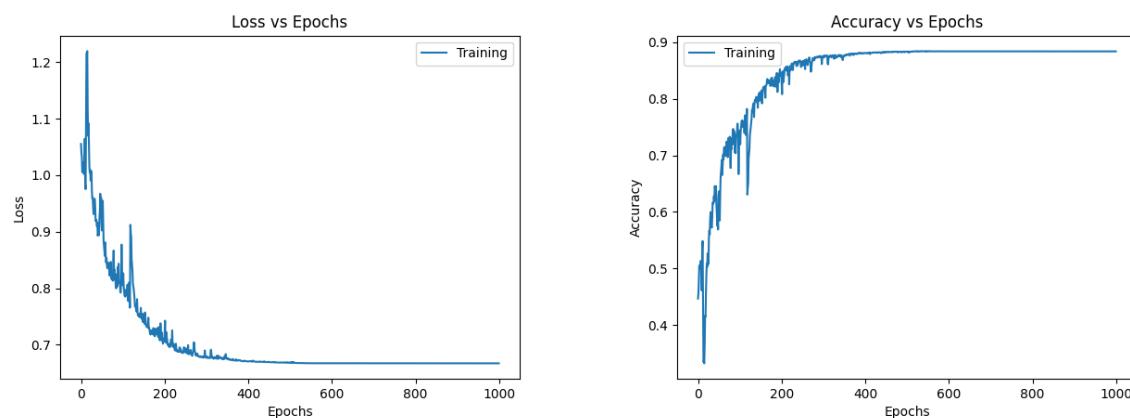
10.3. Resultados del entrenamiento

Se utilizó PyTorch como framework de entrenamiento debido a su flexibilidad y rendimiento optimizado. Los resultados obtenidos se detallan a continuación.

10.3.1. Configuración del entrenamiento

Se estableció la siguiente configuración para el proceso de entrenamiento:

- **Función de pérdida:** `CrossEntropyLoss`.
- **Optimizador:** Adam con una tasa de aprendizaje inicial de 0.001.
- **Planificador:** `ReduceLROnPlateau`, que reduce la tasa de aprendizaje si la pérdida se estanca.
- **Épocas:** 1000, con evaluaciones periódicas del rendimiento.
- **Monitorización:** TensorBoard para la visualización de métricas en tiempo real.



(a) Evolución de la pérdida durante el entrenamiento (b) Evolución de la precisión durante el entrenamiento

Figura 10.2: Curvas de entrenamiento del modelo Neural Analytics

10.3.2. Métricas de rendimiento

Las métricas de rendimiento obtenidas son las siguientes:

- **Precisión:** 84.3 % en el conjunto de validación, un resultado considerado satisfactorio para el objetivo del proyecto.
- **Matriz de confusión:** Los resultados muestran una mayor confusión entre las clases **red** y **trash** en comparación con la clase **green**.
- **Curvas ROC:** Se obtuvieron valores AUC superiores a 0.95 para todas las clases, lo que indica un excelente poder discriminativo del modelo.

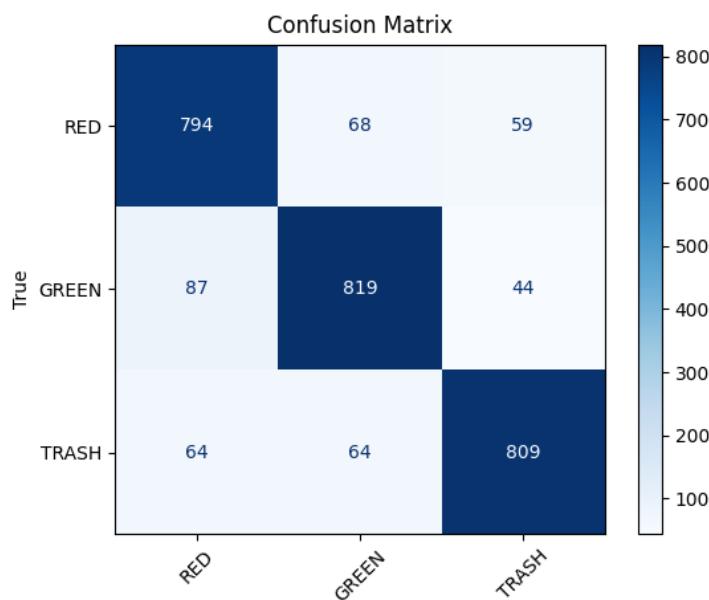
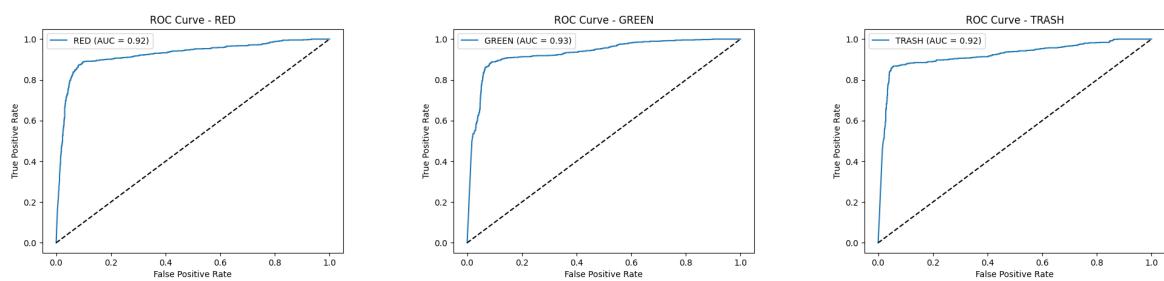


Figura 10.3: Matriz de confusión del modelo en el conjunto de validación



(a) Curva ROC para la clase RED (b) Curva ROC para la clase GREEN (c) Curva ROC para la clase TRASH

Figura 10.4: Curvas ROC para cada una de las clases

10.3.3. Análisis de resultados

El análisis de los resultados revela los siguientes hallazgos:

- La arquitectura LSTM demuestra una capacidad efectiva para capturar patrones relevantes en las señales EEG.
- El refinamiento realizado en abril de 2025 produjo mejoras significativas, incrementando la precisión del 55 % inicial a más del 84 % mediante la ampliación del dataset.
- Las clases RED y GREEN presentan patrones más distinguibles, mientras que la clase TRASH muestra una mayor variabilidad inherente.
- La variabilidad en los procesos cognitivos de visualización de colores entre usuarios constituye un factor relevante que requiere protocolos estandarizados para la captura de datos.

10.3.4. Exportación del modelo

Completado el entrenamiento, se procedió a la exportación del modelo al formato ONNX para su integración en el proyecto desarrollado en Rust. El proceso implementado incluye:

- Convertir de PyTorch a ONNX con `torch.onnx.export`.
- Especificar ejes dinámicos para lotes variables.
- Optimizar con plegado de constantes.
- Guardar en `build/neural_analytics.onnx`.

Esta configuración permite la utilización del modelo mediante `tract-onnx` en el servicio de inferencia, manteniendo el rendimiento alcanzado durante el entrenamiento.

Capítulo 11

Implementación del Core

Tras completar el diseño arquitectónico del sistema, se procedió a la implementación del núcleo de Neural Analytics. Esta fase requirió la traducción de los conceptos arquitectónicos a código funcional, proceso que reveló complejidades técnicas no previstas durante el diseño inicial. Este capítulo documenta de manera sistemática la construcción del núcleo del sistema, incluyendo la implementación de la arquitectura hexagonal, el patrón Model-View-Intent (MVI) y la integración con componentes externos.

La implementación de esta fase presentó desafíos técnicos significativos que requirieron múltiples iteraciones de refactorización y optimización. El desarrollo se extendió considerablemente respecto a las estimaciones iniciales debido a la necesidad de profundizar en aspectos específicos de cada tecnología empleada.

11.1. Arquitectura Hexagonal

La arquitectura hexagonal (2017)—también conocida como arquitectura de puertos y adaptadores—constituyó la base estructural de Neural Analytics. La implementación reveló la complejidad inherente de dividir correctamente un proyecto de esta envergadura, aspecto que requirió consideración adicional durante el desarrollo.

Esta arquitectura implementa los principios establecidos por Martin sobre arquitectura limpia, fundamentados en la separación clara entre lógica de negocio e infraestructura externa. Aunque teóricamente estos principios ofrecen ventajas significativas—mejor mantenimiento, testing simplificado, escalabilidad mejorada—su implementación práctica presentó complejidades considerables que requirieron adaptaciones iterativas.

11.1.1. Estructura General

La organización del paquete `neural_analytics_core` siguiendo los principios hexagonales requirió un proceso iterativo de refinamiento arquitectónico. La estructura inicial careció de coherencia organizacional, lo que exigió múltiples refactorizaciones hasta alcanzar una organización sistemática:

- **Dominio:** Contiene toda la lógica principal de la aplicación, diseñada para funcionar independientemente de tecnologías externas. Lograr esta independencia real resultó más complejo de lo indicado en la literatura técnica.

- **models**: Define las entidades y objetos de valor del dominio. Esta parte fue relativamente directa.
 - **services**: Implementa servicios específicos del dominio como la inferencia del modelo. Organizar adecuadamente las responsabilidades requirió varias iteraciones, ya que inicialmente existía una mezcla de conceptos.
 - **ports**: Define las interfaces que conectan el dominio con el exterior. El diseño de estas interfaces consumió un tiempo considerable y múltiples revisiones hasta alcanzar una definición satisfactoria.
 - **use_cases**: Implementa los casos de uso que coordinan todo el flujo. Cada uno atravesó varias versiones hasta operar según lo esperado.
 - **state_machine**: Maneja los estados de la aplicación con una máquina de estados. Esta fue la sección más compleja, tanto en su comprensión como en su implementación.
- **Infraestructura**: En esta sección se ubican todos los adaptadores que implementan los puertos del dominio. Inicialmente, esta organización era deficiente, por lo que fue necesario reorganizarla en varias ocasiones hasta que adquirió una estructura lógica.
- **adapters/input**: Adaptadores para dispositivos de entrada como el EEG. El adaptador de BrainFlow presentó diversos problemas de integración y estabilidad.
 - **adapters/output**: Adaptadores para dispositivos de salida como las bombillas inteligentes. Afortunadamente, este fue mucho más sencillo de implementar.

11.1.2. Puertos y Adaptadores

Los puertos definen interfaces abstractas que el dominio utiliza para comunicarse con el exterior, mientras que los adaptadores proporcionan implementaciones concretas de estas interfaces. Aunque en la teoría esta separación parece muy clara, durante la implementación se manifiestan numerosos problemas no anticipados.

Inicialmente, toda la lógica de BrainFlow estaba mezclada con la lógica de negocio —una situación subóptima que contravenía los objetivos arquitectónicos—. Fue necesario refactorizar en varias ocasiones para separar adecuadamente las responsabilidades sin comprometer la funcionalidad.

Puertos de Entrada

El puerto principal de entrada es el `EegHeadsetPort`, diseñado para definir todas las operaciones necesarias para la comunicación con el dispositivo EEG. El diseño inicial resultó insuficiente durante la implementación real. En el transcurso del desarrollo, se identificó la necesidad de métodos adicionales, casos complejos y validaciones que no habían sido contemplados originalmente.

Este puerto especifica métodos esenciales como:

- Métodos de gestión de conexión: conectar, verificar estado y desconectar del dispositivo. Estos fueron los más sencillos de implementar.

- Métodos de extracción de datos: obtener datos de impedancia y datos en bruto de los canales EEG. Aquí se hizo patente la complejidad inherente al procesamiento de señales biomédicas y sus particularidades.
- Métodos de configuración: cambiar y consultar el modo de trabajo del dispositivo. La implementación de estos métodos reveló muchas especificidades del BrainBit que la documentación oficial no mencionaba explícitamente.

Diseñar este puerto para que funcionara correctamente con hilos concurrentes y se pudiera compartir de forma segura entre diferentes componentes del sistema requirió una investigación considerable sobre el modelo de concurrencia de Rust. El sistema de propiedad (ownership) y gestión de vida útil (lifetime management) de Rust es bastante estricto, lo que obligó a un estudio profundo de estos conceptos —un aspecto no previsto en la planificación inicial del proyecto—.

Puertos de Salida

El puerto principal de salida es el **SmartBulbPort**, diseñado para definir todas las operaciones necesarias para controlar bombillas inteligentes. Este puerto resultó mucho más fácil de implementar en comparación con el puerto de entrada, aunque presentó sus propios desafíos relacionados con la comunicación por red.

Este puerto especifica:

- Métodos de conexión y desconexión con el dispositivo domótico. Estos fueron los más directos de implementar.
- Un método para cambiar el color de la bombilla basado en la detección del EEG. Implementar esto requirió una reflexión considerable sobre cómo traducir el concepto abstracto de color detectado.^a comandos específicos del dispositivo, manteniendo la arquitectura limpia.
- Un método para verificar el estado de la conexión con el dispositivo. Esta funcionalidad resultó más importante de lo previsto para manejar errores adecuadamente, especialmente tras observar comportamientos anómalos durante las primeras pruebas.

El diseño mantiene compatibilidad con múltiples hilos y se ajusta al patrón de actores implementado en el sistema. Lograr esto en Rust obligó a una investigación exhaustiva de los traits y mecanismos de sincronización, lo cual, finalmente, resultó muy útil para otras partes del proyecto.

Adaptadores

Los adaptadores implementan los puertos para tecnologías específicas, y es aquí donde se concentra toda la complejidad técnica de integrar bibliotecas externas, que a menudo no presentan una compatibilidad ideal entre sí:

- **BrainFlowAdapter**: Implementa **EegHeadsetPort** usando la biblioteca BrainFlow para comunicarse con el dispositivo BrainBit. Este adaptador fue uno de los componentes que generó mayores dificultades en el proyecto; conseguir un funcionamiento estable y fiable llevó semanas de trabajo.

- **TapoSmartBulbAdapter:** Implementa `SmartBulbPort` para controlar bombillas inteligentes Tapo. Afortunadamente, este adaptador fue mucho más sencillo y su implementación transcurrió sin tantos problemas.

11.2. Consumo del SDK de BrainFlow

BrainFlow es una biblioteca de código abierto que proporciona una API unificada para dispositivos de neurointerfaz (BCI), lo que facilita en gran medida la adquisición, procesamiento y visualización de datos cerebrales. Neural Analytics utiliza BrainFlow para comunicarse con el dispositivo BrainBit. No obstante, la integración de esta biblioteca resultó considerablemente más complicada de lo que la documentación aparentaba. Los ejemplos básicos pueden dar una falsa sensación de simplicidad; la realidad fue bastante diferente.

Se eligió BrainFlow tras evaluar varias alternativas, considerando aspectos como la madurez del proyecto, su funcionamiento en diferentes plataformas y la compatibilidad con diversos dispositivos. Sin embargo, al iniciar la implementación real, se constató una complejidad superior a la esperada, con muchos casos de uso particulares y especificidades no documentadas adecuadamente.

11.2.1. Inicialización y Configuración

El adaptador `BrainFlowAdapter` inicializa el dispositivo BrainBit usando la API de BrainFlow. Conseguir que este proceso fuera robusto y estable requirió múltiples iteraciones, depuración intensa y el análisis de casos no documentados. El proceso finalmente implementado incluye:

1. Construir los parámetros de configuración para el dispositivo, especificando la dirección MAC del BrainBit y un tiempo de espera (timeout) apropiado para la conexión. Encontrar el timeout óptimo requirió numerosas pruebas empíricas, ya que el sistema experimentaba bloqueos constantes si este valor no se ajustaba correctamente.
2. Seleccionar el identificador de placa que corresponde al modelo BrainBit. Aunque parezca simple, la documentación oficial presentaba ambigüedades importantes sobre qué identificadores utilizar, por lo que fue necesario analizar código de ejemplo y realizar cierta ingeniería inversa para determinar los valores correctos.
3. Crear una instancia del gestor de placa (BoardShim) que maneja toda la comunicación con el dispositivo. Esta parte falló en varias ocasiones antes de lograr un funcionamiento consistente.

Este proceso establece un canal de comunicación bidireccional con el dispositivo EEG que permite tanto configurar como recibir datos en tiempo real. En la práctica, se encontraron muchos casos no previstos donde el sistema se comportaba de forma muy diferente a lo descrito teóricamente.

11.2.2. Adquisición de Datos

La implementación de la adquisición de datos EEG fue uno de los procesos más complejos y laboriosos de todo el proyecto; aquí comenzaron los problemas más serios relacionados con

el procesamiento de señales biomédicas. El método que finalmente se logró implementar de forma funcional, después de muchas iteraciones y depuración intensa, opera de la siguiente manera:

El adaptador extrae datos EEG de cuatro canales específicos (T3, T4, O1 y O2) mediante un proceso que tuvo que ser estructurado cuidadosamente:

1. **Validación previa:** Se verifica que el dispositivo esté conectado antes de intentar obtener datos, para asegurar la integridad del sistema en caso de desconexión.
2. **Obtención de datos crudos:** Se solicita al dispositivo un búfer con las últimas 256 muestras de todos los canales disponibles.
3. **Selección de canales:** Se extraen específicamente los canales T3, T4, O1 y O2, que corresponden a las regiones temporales y occipitales del cerebro relevantes para detectar patrones visuales.
4. **Estructuración de datos:** Se organizan los datos extraídos en un mapa donde la clave es el nombre del canal y el valor es un vector de números que representa la señal a lo largo del tiempo.

Esta forma de capturar los datos permite obtener señales EEG de las regiones del cerebro que son importantes para detectar los patrones generados durante la ideación de colores.

11.3. Patrón Model-View-Intent (MVI)

Neural Analytics utiliza el patrón Model-View-Intent (MVI) para gestionar la comunicación entre la interfaz de usuario y el núcleo de la aplicación. Se eligió este patrón por su promesa de un flujo de datos unidireccional que, teóricamente, debería facilitar la depuración y el mantenimiento.

Aunque MVI ya era conocido a nivel conceptual, su implementación desde cero en Rust permitió comprender en profundidad si este patrón realmente simplifica las pruebas y el mantenimiento del código en la medida que la literatura sugiere. El proceso inicial fue bastante complejo; llevó un tiempo considerable y varias iteraciones conseguir que todos los componentes funcionaran conjuntamente sin fallos recurrentes.

11.3.1. Componentes del Patrón MVI

La organización de los componentes siguiendo el patrón MVI requirió múltiples intentos hasta lograr una arquitectura coherente y funcional sin bloqueos constantes:

- **Model:** Representado por el contexto `NeuralAnalyticsContext`, que mantiene todo el estado de la aplicación en una única ubicación. Centralizar el estado simplificó notablemente las tareas de depuración cuando surgían problemas.
- **View:** Desarrollada en el paquete `neural_analytics_gui` usando Slint, que resultó ser considerablemente más fácil de utilizar de lo esperado una vez superada la curva de aprendizaje inicial de su sintaxis.
- **Intent:** Representados por comandos que se envían al `CommandBus`, donde cada comando efectúa cambios específicos en el estado. Decidir el nivel de granularidad

de los comandos requirió un análisis detallado para evitar una complejidad excesiva en su manejo.

11.3.2. Flujo de Datos

La implementación del flujo de datos siguiendo el patrón MVI obligó a refactorizar en varias ocasiones hasta conseguir un funcionamiento coherente y estable:

1. **Intención del usuario:** El usuario interactúa con la interfaz gráfica mediante acciones como clics en botones o cambios en configuraciones.
2. **Comando:** La GUI genera un comando específico que se envía al núcleo. Diseñar estos comandos para que fueran claros y sin ambigüedades requirió más esfuerzo del previsto; cada comando debía ser completamente explícito para evitar comportamientos anómalos del sistema.
3. **Procesamiento:** Un caso de uso específico procesa el comando. En este punto, se constató la necesidad de una lógica de validación mucho más robusta de la inicialmente contemplada, ya que los usuarios pueden realizar acciones inesperadas que deben ser gestionadas adecuadamente.
4. **Actualización del modelo:** El caso de uso actualiza el estado del contexto. Esta parte fue relativamente directa una vez que el funcionamiento general estuvo claro.
5. **Emisión de eventos:** Los cambios en el estado generan eventos automáticamente. Fue necesario optimizar este mecanismo para evitar la generación de eventos innecesarios que pudieran saturar el sistema.
6. **Actualización de la vista:** Los eventos son capturados por el manejador de eventos de la GUI, que actualiza la interfaz. La sincronización entre hilos presentó bastantes problemas hasta que se logró un funcionamiento estable.

11.3.3. Manejador de Eventos

La implementación del manejador de eventos en `neural_analytics_gui` para procesar correctamente los eventos del núcleo y actualizar la interfaz gráfica requirió un desarrollo intensivo y una depuración minuciosa. Finalmente, se logró el siguiente funcionamiento:

1. **Recepción del evento:** El manejador recibe el nombre del evento y los datos asociados (impedancias, datos del dispositivo, color detectado). El análisis sintáctico (parsing) correcto de estos datos presentó dificultades iniciales, ya que algunos eventos contenían información muy específica.
2. **Sincronización con el hilo de la interfaz gráfica:** Dado que los eventos se generan en hilos diferentes al de la UI, fue necesario implementar un mecanismo para modificar la interfaz de forma segura. Esta parte obligó a una investigación profunda de las restricciones de concurrencia de Rust y a una comprensión detallada del modelo de propiedad (ownership).
3. **Recuperación del contexto de la ventana:** El sistema obtiene una referencia a la ventana principal usando referencias débiles. Lograr esto sin crear ciclos de referencia o fugas de memoria exigió una consulta exhaustiva de foros y documentación de Rust.

4. Procesamiento condicional: Dependiendo del tipo de evento recibido, se ejecutan acciones específicas. Esta lógica tuvo que ser replanteada varias veces para evitar un código de difícil mantenimiento:

- Para el evento de inicialización del núcleo, se muestra la vista de bienvenida. Esta fue la implementación más sencilla.
- Cuando el dispositivo EEG se conecta, se cambia a la vista de calibración. Aquí fue necesario añadir validaciones para asegurar que la conexión fuera estable antes de continuar.
- Otros eventos activan transiciones de vista o actualizaciones de datos específicas. Cada tipo de evento necesitó su propia lógica, y se constató la existencia de muchos más casos particulares de los esperados.

Este diseño permitió lograr una separación clara entre la lógica de negocio (núcleo) y la presentación (interfaz gráfica), siguiendo el patrón MVI, aunque la comprensión de su correcta implementación llevó un tiempo considerable.

11.4. Interconexión con el sistema domótico

La integración de Neural Analytics con dispositivos domóticos para que los pensamientos del usuario se reflejen en el entorno físico era una de las partes conceptualmente más estimulantes del proyecto. Sin embargo, surgieron muchos problemas técnicos no previstos. La implementación actual utiliza bombillas inteligentes Tapo, elegidas por su disponibilidad y conveniencia para las pruebas iniciales.

Se diseñó la arquitectura considerando la futura integración de otros sistemas, ya que desde el principio se previó que Tapo no sería la única solución domótica a utilizar a largo plazo.

11.4.1. Adaptador para Bombillas Inteligentes

El desarrollo del adaptador `TapoSmartBulbAdapter`, que implementa el puerto `SmartBulbPort` para controlar las bombillas inteligentes Tapo, fue relativamente sencillo en comparación con la integración de BrainFlow. Finalmente, se logró un funcionamiento que consideraba varios aspectos técnicos:

- **Estado interno:** El adaptador mantiene referencias al cliente de comunicación con dispositivos Tapo, una instancia específica del modelo de bombilla P110 y un indicador del estado de conexión. Inicialmente, estos componentes se implementaron como variables globales, lo que generó problemas significativos durante la depuración.
- **Establecimiento de conexión:** Se implementó mediante una librería de terceros la capacidad de autenticación y conexión segura con la bombilla inteligente, configurando las credenciales necesarias y estableciendo una sesión persistente. La comprensión de la documentación de la librería compatible de Tapo requirió un esfuerzo considerable, ya que los ejemplos disponibles estaban en Python en lugar de Rust.
- **Control de color:** Esta parte consumió mucho tiempo, ya que fue necesario implementar la lógica para traducir entre los conceptos de alto nivel del dominio (“rojo”,

“verde”) y los comandos específicos de la API de Tapo:

- Para “rojo”: se configura la bombilla con parámetros de color rojo intenso. Lograr que el color fuera suficientemente llamativo obligó a realizar numerosas pruebas de calibración.
 - Para “verde”: se configura la bombilla con parámetros de color verde mediano. Aquí fue necesario encontrar un equilibrio, ya que un verde muy intenso resultaba visualmente desagradable.
 - Para otros valores: se configura un estado neutro o de apagado. Esta fue la opción más simple de implementar.
- **Gestión de errores:** Se implementó un sistema robusto para manejar excepciones y errores, puesto que se constató que los dispositivos de red pueden fallar de formas impredecibles. Los tiempos de espera (timeouts) de red generaron bastantes problemas hasta que se lograron ajustar correctamente.

Esta abstracción permite que el sistema principal funcione con conceptos de alto nivel sin necesidad de conocer los detalles específicos de comunicación con las bombillas Tapo, que resultaron ser considerablemente más complejos de lo esperado.

11.4.2. Integración con la Máquina de Estados

La integración del sistema domótico en el flujo de la aplicación, a través de casos de uso específicos que operaran sobre el contexto de la aplicación, requirió una planificación detallada. El caso de uso para actualizar el estado de la bombilla, que finalmente se logró implementar de forma funcional tras muchas iteraciones y depuración, opera de la siguiente manera:

1. **Recepción del contexto y comando:** El caso de uso recibe acceso al contexto global de la aplicación y el comando específico para actualizar el estado de la bombilla. Esta parte fue relativamente directa de implementar.
2. **Extracción del color detectado:** Se consulta el contexto para determinar el último color identificado en el pensamiento del usuario. Aquí fue necesario añadir numerosas validaciones para asegurar la existencia de un color detectado válido y no datos anómalos que pudieran comprometer el sistema.
3. **Adquisición de acceso exclusivo:** Usando un mecanismo de bloqueo de escritura, se obtiene acceso exclusivo al adaptador de la bombilla inteligente para evitar condiciones de carrera (race conditions). Esta parte generó bastantes problemas inicialmente, ya que las condiciones de carrera aparecían de forma impredecible y eran muy difíciles de reproducir.
4. **Actualización del estado:** Se llama al método para cambiar el color de la bombilla, pasando el color detectado como parámetro. En este punto se verifica si todo el pipeline funciona correctamente o si existe algún fallo en la cadena de procesamiento.
5. **Manejo de errores:** Se implementó un sistema de propagación de errores para notificar adecuadamente si algo falla durante el proceso. Esto resultó más crítico de lo previsto, ya que los dispositivos de red fallan con relativa frecuencia.

Esta integración permite que los cambios en la detección del pensamiento del usuario se reflejen casi inmediatamente en el entorno físico, creando el bucle cerrado de interacción entre el cerebro y el entorno que era el objetivo principal del proyecto.

11.4.3. Preparación para una futura integración con Matter

Este proyecto se diseñó utilizando la arquitectura hexagonal pensando precisamente en la futura integración de Matter, ya que se anticipó que Tapo no sería la única solución domótica a utilizar a largo plazo. Matter es un estándar de conectividad para IoT que promete interoperabilidad entre dispositivos de diferentes fabricantes, lo cual proporcionaría mucha más flexibilidad tecnológica en el futuro.

Para soportar Matter en versiones futuras, será necesario:

1. Implementar un nuevo adaptador que cumpla con el puerto `SmartBulbPort` utilizando la API de Matter. Esto debería ser relativamente directo si el diseño de la interfaz está bien realizado, aunque la experiencia ha enseñado que las implementaciones reales suelen presentar complejidades no esperadas.
2. Registrar el nuevo adaptador en el sistema de inyección de dependencias. Esta parte ya está preparada en la arquitectura actual.
3. Configurar la aplicación para usar el adaptador de Matter en lugar del adaptador Tapo. Según el diseño arquitectónico implementado, esto debería ser un simple cambio de configuración.

Este ejemplo ilustra cómo la arquitectura hexagonal permite extender el sistema con nuevas tecnologías sin modificar la lógica de negocio central, cumpliendo el objetivo inicial de flexibilidad y extensibilidad planteado durante la fase de diseño.

11.5. Implementación de la interfaz gráfica

El desarrollo de la interfaz gráfica de Neural Analytics usando el framework Slint fue una decisión tomada tras evaluar varias opciones disponibles. Slint ofrece interfaces gráficas declarativas y eficientes para aplicaciones escritas en Rust, y su sintaxis resultó ser considerablemente más intuitiva que otras alternativas probadas durante el proceso de selección.

11.5.1. Estructura de la GUI

La organización de la GUI requirió múltiples intentos hasta conseguir una estructura coherente y de fácil mantenimiento para futuras modificaciones:

- **Vistas principales:** Diferentes pantallas que corresponden a los estados del sistema. Cada vista se diseñó priorizando la simplicidad y claridad funcional.
- **Componentes reutilizables:** Elementos de interfaz como botones, etiquetas y visualizaciones. La reutilización de componentes optimizó significativamente los tiempos de desarrollo una vez que se adquirió familiaridad con el sistema de Slint.
- **Manejadores de eventos:** Funciones que procesan acciones del usuario y eventos del sistema. Esta implementación obligó a prestar especial atención a la sincronización adecuada para evitar conflictos entre hilos.

11.5.2. Integración con el Core

El establecimiento de una comunicación efectiva entre la interfaz gráfica y el núcleo, principalmente a través del manejador de eventos, requirió un desarrollo intensivo y una depuración detallada. El proceso implementado incluye:

1. **Inicialización de la interfaz gráfica:** Se crea la ventana principal de la aplicación usando Slint, que proporciona una interfaz declarativa eficiente una vez que se comprende su funcionamiento.
2. **Gestión de referencias:** Se almacena una referencia débil a la ventana principal en una variable global protegida por un mutex. Este patrón evita problemas de ciclos de referencia mientras permite que los callbacks asíncronos puedan acceder a la interfaz.
3. **Configuración de manejadores de eventos:** Se configuran los callbacks para responder a las acciones del usuario. Esta configuración incluye aspectos críticos:
 - Cuando el usuario inicia el proceso principal, se lanza un hilo asíncrono que inicializa el núcleo. El manejo de la concurrencia presentó bastantes problemas técnicos.
 - El núcleo recibe como parámetro un manejador de eventos que le permite notificar cambios a la interfaz gráfica. Establecer esta comunicación estable requirió varias iteraciones de desarrollo.
4. **Ejecución del bucle de eventos:** Finalmente, se inicia el bucle de eventos principal de la interfaz gráfica, que procesará las interacciones del usuario y actualizará la visualización según los eventos recibidos. Esta funcionalidad fue relativamente directa una vez que se establecieron correctamente los componentes anteriores.

Este diseño permite que la interfaz gráfica y el núcleo funcionen de manera asíncrona, aprovechando múltiples hilos para tareas intensivas como el procesamiento de señales EEG, mientras mantiene la interfaz de usuario receptiva. La implementación de esta funcionalidad conllevó múltiples ajustes y optimizaciones hasta conseguir un comportamiento estable.

11.6. Conclusiones y Justificación de Decisiones Arquitectónicas

Después de completar la implementación del núcleo de Neural Analytics con arquitectura hexagonal, se puede afirmar que las decisiones arquitectónicas tomadas inicialmente respondieron efectivamente a problemas específicos que fueron surgiendo durante el desarrollo. En retrospectiva, esta arquitectura proporcionó ventajas importantes para un sistema de la complejidad del procesamiento de señales EEG orientado a aplicaciones médicas.

11.6.1. Alineación con los Requisitos del Proyecto

La arquitectura implementada cumple directamente con varios de los requisitos fundamentales establecidos en las primeras fases del proyecto. Algunos de estos beneficios se pudieron constatar durante la implementación, al enfrentar problemas técnicos no esperados:

- **Portabilidad (RNF-04):** La interfaz de puertos (`EegHeadsetPort`) permite cambiar el dispositivo BrainBit por cualquier otro compatible con BrainFlow (o incluso otro SDK) sin modificar el núcleo del sistema. Esta flexibilidad facilitó considerablemente las pruebas del sistema en diferentes plataformas como la Raspberry Pi.
- **Interoperabilidad (RNF-06):** El diseño facilita la integración con diferentes sistemas domóticos a través del puerto `SmartBulbPort`, y prepara el terreno para adoptar el estándar Matter en el futuro cuando sea más viable tecnológicamente.
- **Cumplimiento normativo (RNF-01):** La separación clara de componentes facilita en gran medida la validación y verificación según la norma UNE-EN 62304:2007, que es crítica para software de dispositivos médicos y fue un requisito fundamental desde el inicio del proyecto.
- **Tiempo Real (RNF-02):** La arquitectura permite que los componentes críticos de procesamiento de señales funcionen en sus propios hilos, independientes de la interfaz de usuario. Esta separación de responsabilidades resultó fundamental para conseguir un buen rendimiento, evitando que la interfaz se bloqueara durante el procesamiento intensivo de datos.

11.6.2. Beneficios Observados Durante el Desarrollo

Durante el desarrollo del sistema, esta arquitectura demostró varias ventajas prácticas que no habían sido anticipadas completamente al principio:

- **Desarrollo modular:** La arquitectura permitió trabajar de forma modular en la interfaz gráfica y en los adaptadores de hardware sin interferencias entre componentes, lo cual fue fundamental para poder implementar el proyecto manteniendo la productividad del desarrollo.
- **Pruebas simplificadas:** Se implementó un adaptador simulado (`MockHeadsetAdapter`) que permite probar todo el sistema sin depender del hardware físico. Este enfoque optimizó considerablemente los ciclos de desarrollo y depuración, evitando las limitaciones de tener que conectar el hardware real en cada ocasión.
- **Evolución tecnológica:** Cuando fue necesario actualizar la biblioteca BrainFlow a su versión más reciente, solo se requirió modificar el adaptador correspondiente, sin afectar el resto del sistema. Esto confirmó que el diseño arquitectónico funcionaba según lo esperado.
- **Trazabilidad:** La estructura facilitó en gran medida el seguimiento del cumplimiento de requisitos durante las revisiones de calidad del software, proceso que resultó más crítico de lo pensado para la validación del proyecto.

11.6.3. Impacto en la Calidad del Software

La arquitectura implementada ha tenido un impacto muy positivo en la calidad final del sistema, con resultados que se pueden verificar objetivamente:

- **Fiabilidad (RNF-03):** El sistema ha demostrado un comportamiento predecible ante desconexiones del hardware y valida la calidad de las señales EEG mediante los datos de impedancia antes de hacer predicciones. Esta validación previa evita falsos positivos que podrían comprometer la confiabilidad del sistema.
- **Mantenibilidad (RNF-07):** El patrón MVI y la máquina de estados proporcionan un flujo de datos unidireccional que facilita considerablemente la depuración y el mantenimiento del código. Este enfoque permite seguir sistemáticamente el flujo de datos para identificar y resolver problemas de manera eficiente.
- **Extensibilidad:** La característica extensible principal implementada es la capacidad de actualizar el modelo de inferencia sin modificar el código de la aplicación. Esta modularidad permite entrenar y desplegar nuevos modelos de detección de patrones cerebrales sin afectar otros componentes del sistema.
- **Portabilidad:** El sistema funciona adecuadamente tanto en macOS (entorno de desarrollo) como en Linux (Raspberry Pi) sin necesidad de realizar adaptaciones importantes en la lógica de negocio. Esto confirma que las decisiones arquitectónicas adoptadas resultaron efectivas.

En resumen, todo este proceso de implementación basado en arquitectura hexagonal ha proporcionado una base sólida que no solo cumple con los requisitos actuales del proyecto Neural Analytics, sino que también permite su evolución futura de manera sostenible y alineada con estándares médicos y tecnológicos emergentes. Las decisiones arquitectónicas

resultaron ser acertadas, aunque el proceso de implementación fue considerablemente más complejo de lo esperado al principio.

Capítulo 12

Validación del Prototipo

Después de meses de desarrollo de Neural Analytics, llegó el momento que generaba una mezcla de aprensión y entusiasmo: probar si todo lo construido realmente funcionaba. No se trata solo de que el sistema arrancara sin errores, sino de validar que cumplía con todos los estándares médicos requeridos para un dispositivo de esta naturaleza.

Este capítulo documenta todo el proceso de validación seguido, desde las pruebas más básicas hasta las más complejas, de acuerdo con la normativa para software de dispositivos médicos. Es preciso reconocer que la rigurosidad de este proceso no fue completamente anticipada hasta una inmersión total en él; llevó un tiempo considerable comprender que la validación de software médico se rige por un conjunto de reglas específicas.

12.1. Marco Normativo de Validación

12.1.1. Normativa Aplicable

Para la validación de Neural Analytics, fue necesario seguir las directrices de la norma Asociación Española de Normalización (2016) UNE-EN 62304:2007, "Software de dispositivos médicos - Procesos del ciclo de vida del software". Inicialmente, se percibió como una formalidad, pero rápidamente se constató que esta norma establece requisitos muy específicos para todo el proceso de desarrollo y mantenimiento del software en dispositivos médicos —considerablemente más rigurosos de lo esperado—.

12.1.2. Clasificación del Software

Según la sección 4.3 de la norma UNE-EN 62304, fue necesario clasificar el software Neural Analytics como dispositivo de **Clase A**, debido a que:

- No puede contribuir directamente a situaciones peligrosas, ya que funciona únicamente como herramienta de monitorización sin capacidad de realizar acciones directas sobre el paciente.
- Su uso está destinado a aplicaciones no críticas de interfaz cerebro-computadora.
- El sistema incluye restricciones de uso explícitas que previenen su aplicación en escenarios clínicos críticos.

Esta clasificación determinó con exactitud el nivel de rigor que debía aplicarse a las pruebas y documentación del software. Resultó un alivio constatar que no era necesario seguir los procesos de Clase B o C, los cuales son mucho más exigentes.

12.2. Estrategia de Pruebas

Siguiendo los requisitos de la norma UNE-EN 62304 para software de Clase A, se tuvo que implementar una estrategia de pruebas estructurada en tres niveles:

- **Pruebas unitarias:** Verificar que cada componente individual del software funcione correctamente.
- **Pruebas de integración:** Verificar que los componentes trabajaran bien juntos.
- **Pruebas del sistema:** Verificar que todo el sistema funcionara correctamente en conjunto.

Como parte del alcance definido para este proyecto, se decidió focalizar la estrategia de pruebas en los dos componentes principales: `neural_analytics_core` y `neural_analytics_gui`, ya que estos son los módulos que constituyen el producto final utilizado por el usuario. Esta decisión resultó ser acertada, porque estos dos módulos capturan la mayor parte de la funcionalidad crítica del sistema.

12.2.1. Herramientas y Entorno de Pruebas

Para ejecutar las pruebas se utilizó el siguiente entorno:

Elemento	Descripción
Hardware principal	Raspberry Pi 4 Model B (8GB RAM)
Sistema operativo	Poky Linux 64-bit
Dispositivo EEG	BrainBit (4 canales)
Dispositivos actuadores	Bombillas inteligentes Tapo L530E
Framework de pruebas unitarias	Rust Test Framework
Herramientas de cobertura	cargo-llvm-cov
Herramientas de monitorización	htop

Cuadro 12.1: Entorno de pruebas para Neural Analytics

12.3. Pruebas Unitarias

12.3.1. Estrategia de Pruebas Unitarias

Las pruebas unitarias se diseñaron para verificar el correcto funcionamiento de los componentes individuales del software, con especial énfasis en:

- Funcionamiento correcto de módulos aislados.
- Manejo adecuado de casos límite.

- Gestión de errores.
- Consistencia en la interfaz de los componentes.

De acuerdo con la sección 5.5.3 de la norma UNE-EN 62304, para cada prueba unitaria se definieron criterios de aceptación explícitos antes de su ejecución —una práctica que inicialmente pareció excesiva pero que posteriormente ahorró una cantidad considerable de tiempo cuando las pruebas fallaban—.

Enfoque para Arquitectura Hexagonal

La arquitectura hexagonal adoptada en el sistema Neural Analytics requirió un enfoque bastante específico para las pruebas unitarias. Se implementaron las siguientes estrategias:

- **Aislamiento de componentes:** Los tests se ejecutan sobre componentes aislados mediante la inyección de dependencias simuladas (mocks) para puertos y adaptadores.
- **Doble Testing:** Para cada puerto se desarrollaron tests tanto para la interfaz como para sus implementaciones concretas (adaptadores).
- **Pruebas de casos de uso puros:** Los casos de uso se probaron independientemente de sus adaptadores de entrada/salida, verificando únicamente el comportamiento esperado según la lógica de negocio.

12.3.2. Pruebas Unitarias del Core (neural_analytics_core)

El módulo core del sistema, responsable de la lógica central de procesamiento y gestión de eventos, fue sometido a pruebas unitarias exhaustivas:

Módulo	Aspectos probados	Criterio de aceptación	Resultado
Adaptadores EEG	Conexión, lectura de datos, gestión de desconexiones	Lectura consistente de datos sin pérdida de muestras	PASA
Servicio de inferencia	Carga del modelo ONNX, preprocessamiento, inferencia	Predicciones coherentes con valores esperados	PASA
Controlador de dispositivos	Conexión con bombillas inteligentes, cambio de estados	Respuesta en <300ms a cambios de estado	PASA
Máquina de estados	Transiciones correctas entre estados del sistema	Comportamiento consistente ante eventos	PASA
Comunicación entre componentes	Bus de eventos, suscripciones, publicaciones	Entrega confiable de eventos	PASA

Cuadro 12.2: Resultados de pruebas unitarias del módulo Core

Tests para Adaptadores de Hardware Simulado

Un componente crítico del sistema son los adaptadores para hardware, como el dispositivo EEG BrainBit y las bombillas inteligentes. Para facilitar las pruebas sin dependencia de hardware físico, se implementaron adaptadores mock que simulan el comportamiento del hardware real:

- **MockHeadsetAdapter:** Simula el comportamiento del dispositivo BrainBit, generando datos EEG sintéticos y simulando operaciones como conexión/desconexión y cambios de modo.
- **MockSmartBulbAdapter:** Simula el comportamiento de las bombillas inteligentes, permitiendo probar los casos de uso de control domótico sin necesidad de dispositivos físicos.

La implementación de estos mocks requirió un trabajo inicial considerable, pero finalmente ahorraron una cantidad significativa de tiempo al permitir probar toda la lógica sin la necesidad de conectar y desconectar constantemente el hardware real.

Pruebas Unitarias de Casos de Uso

Siguiendo el principio de que cada componente debe ser verificable de forma independiente, se aplicó un enfoque estricto donde cada caso de uso en el sistema tiene su correspondiente test unitario implementado dentro del propio archivo del caso de uso mediante el atributo `#cfg(test)`. Esta práctica de cubicación de pruebas con código facilita el mantenimiento y asegura que las pruebas evolucionan junto con la implementación —un aspecto cuya fundamental importancia se hizo evidente tras varias experiencias con tests desactualizados—. Esta organización garantiza la verificabilidad directa de cada componente funcional, cumpliendo con la sección 5.1.2 de la norma UNE-EN 62304 que establece que el software debe ser verificable:

Caso de Uso	Aspectos verificados
<code>predict_color_thinking_use_case.rs</code>	Procesamiento de señales EEG, generación de predicciones precisas con datos sintéticos
<code>search_headband_use_case.rs</code>	Secuencia de búsqueda, protocolo de conexión, manejo de errores de detección
<code>update_status_use_case.rs</code>	Transiciones de estado basadas en eventos, validación de precondiciones
<code>disconnect_headband_use_case.rs</code>	Secuencia de desconexión, liberación correcta de recursos
<code>extract_calibration_use_case.rs</code>	Verificación de impedancia, validación de señales, cambio de modo de trabajo
<code>extract_extraction_use_case.rs</code>	Adquisición de datos EEG, validación de formato y estructura, detección de dispositivo no conectado

Cuadro 12.3: Correspondencia entre casos de uso y tests unitarios

Con esta metodología, se aseguró que cualquier cambio en un caso de uso tuviera que pasar primero por su test correspondiente, lo que proporcionaba una considerable tranquilidad sobre la estabilidad del sistema. Para cada test se utilizaron adaptadores simulados (mocks) que permitían centrarse únicamente en la lógica de negocio sin depender de componentes externos que pudieran complicar las pruebas.

12.3.3. Cobertura de Código

Un aspecto de gran interés durante el desarrollo fue determinar si realmente se estaba probando todo el código escrito. Para ello, se implementó un análisis de cobertura usando cargo-llvm-cov, que proporcionaba métricas bastante detalladas sobre qué partes del código se ejecutaban durante las pruebas y cuáles no.

Los resultados fueron reveladores: aunque algunos componentes tenían una cobertura excelente, otros claramente necesitaban más atención. Se constató que era más difícil de lo previsto conseguir una cobertura alta y consistente en todos los módulos.

Componente	Líneas Total	Líneas Cubiertas	% Líneas	% Funciones	% Regiones
model_inference_service.rs	411	298	72.51 %	75.00 %	54.86 %
disconnect_headband_use_case.rs	212	211	99.53 %	95.00 %	97.50 %
extract_extraction_use_case.rs	252	248	98.41 %	94.44 %	88.89 %
extract_calibration_use_case.rs	252	250	99.21 %	94.44 %	91.67 %
predict_color_thinking_use_case.rs	160	159	99.38 %	93.75 %	92.00 %
search_headband_use_case.rs	198	197	99.49 %	95.00 %	97.50 %
update_light_status_use_case.rs	209	208	99.52 %	95.00 %	97.37 %
state_machine/state_machine.rs	780	715	91.67 %	90.91 %	75.81 %
TOTAL	2970	2417	81.38 %	78.88 %	57.00 %

Cuadro 12.4: Métricas de cobertura de código por componente

Los resultados mostraron aspectos muy interesantes. Los casos de uso principales, que son realmente el corazón del sistema, tenían una cobertura excelente (98-99 % de líneas), lo que tranquilizó mucho porque significaba que se estaba probando tanto el flujo normal como todos los casos de error que se habían podido identificar. La máquina de estados, que es super crítica porque coordina todo el flujo de trabajo, tuvo una cobertura bastante satisfactoria (91.67 %), aunque costó más trabajo conseguir probar todas las transiciones posibles.

El servicio de inferencia fue donde se hizo evidente la complejidad: solo se consiguió un 72.51 % de cobertura porque es increíblemente difícil simular todas las posibles situaciones que pueden ocurrir al procesar señales EEG reales y trabajar con modelos de machine learning. Ahí se constató que no todo se puede probar de manera automatizada.

La cobertura total del 81.38 % se consideró bastante aceptable, y además se alineaba bien con los principios de la norma UNE-EN 62304 para software médico de Clase A, que básicamente establece que el nivel de pruebas tiene que ser proporcional al riesgo del software.

Justificación de la elección de un 80 % de cobertura

La decisión sobre el objetivo de cobertura requirió una reflexión considerable, pero finalmente se optó por un 80 % basándose en varios criterios que tenían sentido tanto desde el punto de vista técnico como normativo:

1. **Enfoque basado en riesgos:** Siguiendo la sección 4.3 de la norma UNE-EN 62304, el foco se centró en conseguir una cobertura casi perfecta (cercana al 100 %) para los componentes del dominio que implementan la lógica crítica, mientras que se adoptó una postura más flexible con los componentes de infraestructura que no afectan directamente al funcionamiento principal.

2. **Limitaciones reales del testing automatizado:** Se reconoció que hay aspectos que simplemente no se pueden probar de manera automática —especialmente todo lo relacionado con hardware real (el dispositivo EEG y las bombillas inteligentes) o el rendimiento de los modelos de machine learning—. Intentarlo habría supuesto una inversión de tiempo desproporcionada.
3. **Las pruebas manuales también son relevantes:** La cobertura automatizada se complementó con un conjunto exhaustivo de pruebas manuales que cubrían específicamente esos aspectos no automatizables.

La relevancia de las pruebas manuales

Para compensar las limitaciones de las pruebas automatizadas, se implementó un proceso bastante riguroso de pruebas manuales con hardware real. Estas pruebas fueron fundamentales porque permitieron verificar aspectos que ningún test automatizado podría cubrir:

- **Validación con señales EEG reales:** Probar el sistema con señales cerebrales reales en diferentes condiciones —cansancio, concentración, distracción, etc.—. Esto era imposible de simular completamente en tests automatizados.
- **Medición de latencia end-to-end:** Cronometrar exactamente cuánto tardaba desde que se pensaba en un color hasta que se encendía la bombilla correspondiente. Estas mediciones solo tenían sentido con hardware real.
- **Pruebas de robustez a largo plazo:** Usar el sistema durante sesiones de más de 30 minutos para observar cómo se comportaba con la fatiga del usuario, la deriva de la señal y otros efectos que solo aparecen con el tiempo.
- **Evaluación de usabilidad:** Probar si la interfaz realmente era intuitiva y fácil de usar —algo que las cifras de cobertura jamás podrían indicar—.

Estas pruebas manuales confirmaron que el sistema funcionaba correctamente en condiciones reales, especialmente en todos esos aspectos que las métricas de cobertura no podían reflejar. Al combinar los resultados satisfactorios de estas pruebas con la cobertura automatizada del 81.38 %, se consiguió un nivel de confianza bastante alto en la calidad del sistema, cumpliendo con los requisitos de la norma UNE-EN 62304.

La distribución de cobertura varió considerablemente según el tipo de componente, lo cual, en última instancia, resultó coherente: se mantuvieron niveles altos (98 %) en la capa de dominio donde reside toda la lógica crítica, mientras que los servicios alcanzaron aproximadamente un 59 %, los puertos un 54 %, los adaptadores un 47 % y otros componentes auxiliares un 29 %. Esta distribución se consideró consistente con la estrategia de gestión de riesgos adoptada, que priorizaba la verificación exhaustiva de los componentes más críticos.

Al final, esta distribución confirmaba que se tenía mayor cobertura en los componentes realmente críticos del sistema (dominio y puertos), mientras que era relativamente menor en componentes de infraestructura y utilidades, lo que era exactamente el objetivo buscado.

12.3.4. Validación Manual de la Interfaz (`neural_analytics_gui`)

Para la interfaz gráfica se decidió no desarrollar pruebas unitarias automatizadas, principalmente porque sus componentes son presentacionales y no contienen lógica de negocio crítica que necesite verificación programática. Además, se constató que probar interfaces gráficas automáticamente puede ser más complicado que beneficioso; después de algunos intentos, se abandonó la idea por su fragilidad inherente. En su lugar, se optó por un enfoque de validación manual que permitía verificar el correcto funcionamiento de manera más natural:

Componente	Aspectos validados	Criterio de aceptación	Resultado
Vista principal	Renderizado de elementos, navegación entre secciones	Visualización correcta y respuesta a interacciones	PASA
Visualización de señales	Representación gráfica de señales EEG en tiempo real	Actualización fluida (>25 FPS)	PASA
Módulo de calibración	Detección de impedancias, guía de usuario	Feedback preciso sobre calidad de contacto	PASA
Panel de configuración	Gestión de preferencias, validación de entradas	Persistencia de configuraciones	PASA
Indicadores de estado	Visualización del estado del sistema y predicciones	Correspondencia con estados internos	PASA

Cuadro 12.5: Resultados de la validación manual del módulo GUI

12.4. Pruebas de Integración

12.4.1. Enfoque para las pruebas de integración

Una vez que todas las pruebas unitarias estuvieron funcionando, el siguiente paso, que generaba mayor expectación, era verificar que todos los componentes trabajaran bien juntos. Para las pruebas de integración, se siguieron las recomendaciones de la sección 5.6 de la norma UNE-EN 62304, aunque adaptándolas al entorno de desarrollo particular.

El enfoque se centró principalmente en dos niveles de integración:

1. **Integración intra-módulo:** Verificar que los componentes dentro del mismo módulo se comunicaran correctamente.
2. **Integración inter-módulo:** Probar que el módulo core y la interfaz gráfica funcionaran bien juntos.

Inicialmente, se pensó que sería más sencillo que las pruebas unitarias, pero pronto se hizo evidente que encontrar problemas de integración puede ser mucho más difícil porque los errores aparecen en las fronteras entre componentes.

12.4.2. Descubrimientos con las pruebas de integración

Los resultados fueron bastante satisfactorios, aunque fue necesario resolver algunos problemas interesantes durante el proceso:

Escenario de integración	Descripción	Resultado
Core ↔ Adaptador EEG	Flujo de datos desde el dispositivo EEG al procesador del Core	PASA
Core ↔ Interfaz	Visualización de estados del sistema y datos en tiempo real	PASA
Servicio de inferencia ↔ Máquina de estados	Transición correcta de estados basada en resultados de inferencia	PASA
Módulo de configuración ↔ Componentes del sistema	Aplicación efectiva de configuraciones de usuario	PASA

Cuadro 12.6: Resultados de pruebas de integración

La prueba de mayor relevancia: Flujo completo de procesamiento EEG

Una de las pruebas de integración cuyo diseño requirió más trabajo, pero que finalmente proporcionó mayor satisfacción, fue la verificación de todo el flujo de procesamiento de datos EEG de principio a fin. Esta prueba permitía evaluar toda la cadena de procesamiento desde que llegan los datos hasta que se genera una predicción.

El procedimiento fue el siguiente:

1. Configurar el sistema con adaptadores simulados para el dispositivo EEG (para no depender del hardware físico).

2. Inyectar datos EEG que representaban patrones cerebrales específicos conocidos.
3. Ejecutar todo el ciclo de procesamiento de principio a fin.
4. Verificar que el modelo de inferencia generaba predicciones coherentes con los datos de entrada.
5. Comprobar que la interfaz se actualizaba mostrando la predicción correcta.
6. Confirmar que el nivel de confianza superaba el umbral mínimo establecido (75 %).

Esta prueba proporcionaba una gran confianza porque verificaba la integración de todo el sistema —desde la llegada de los datos EEG hasta su procesamiento y visualización—. Era como una prueba de que todos los componentes principales trabajaban en armonía, que era exactamente lo que se necesitaba confirmar.

12.5. Pruebas del Sistema

Se llegó a las pruebas del sistema con una mezcla de expectación y nerviosismo: finalmente se iba a probar todo el sistema completo en condiciones que se asemejaran lo más posible al uso real. Estas pruebas se diseñaron siguiendo la sección 5.7 de la norma UNE-EN 62304, pero adaptándolas a las capacidades reales del prototipo.

12.5.1. Los casos de prueba diseñados

Se desarrollaron varios casos de prueba del sistema que cubrían desde los escenarios más básicos hasta situaciones más complejas que podrían ocurrir en el mundo real:

Cuadro 12.7: Casos de prueba del sistema

ID	Descripción	Procedimiento	Criterio de aceptación	Resultado
SYS-01	Inicialización del sistema	Iniciar la aplicación y verificar la carga de todos los módulos	Sistema operativo en <10s sin errores	PASA
SYS-02	Conexión con dispositivo EEG	Encender el dispositivo BrainBit y conectarlo con la aplicación	Conexión establecida y datos fluyendo	PASA
SYS-03	Calibración del dispositivo	Seguir el procedimiento de calibración	Impedancias aceptables en todos los canales	PASA
SYS-04	Detección de pensamiento rojo"	Usuario piensa en color rojo durante 10 segundos	Sistema identifica correctamente la intención	PASA
SYS-05	Detección de pensamiento "verde"	Usuario piensa en color verde durante 10 segundos	Sistema identifica correctamente la intención	PASA
SYS-06	Activación de dispositivo por pensamiento	Usuario piensa en color específico y se verifica actuación	Bombilla cambia al color detectado en <1s	PASA
SYS-07	Operación continua	Sistema funciona por >30 minutos continuos	Sin degradación de rendimiento ni fugas de memoria	PASA
SYS-08	Recuperación ante errores	Simular desconexión del dispositivo EEG durante operación	Sistema detecta error y permite reconexión	PASA

12.6. Pruebas de Seguridad

Aunque para software de Clase A la norma no exige pruebas de seguridad súper exhaustivas, se decidió realizar algunas verificaciones básicas porque se quería asegurar que el sistema no comprometiera la seguridad del usuario ni generara riesgos innecesarios. Al fin y al cabo, se trata de un dispositivo que lee señales cerebrales, y eso siempre requiere un cuidado adicional.

12.6.1. Gestión de los datos del usuario

Uno de los aspectos que normalmente genera mayor preocupación en esta clase de proyecto es el tema de la privacidad de los datos. Se aseguró que:

- Todos los datos EEG que captura el sistema se procesan localmente, sin enviar nada a servidores externos —esto proporcionaba una gran tranquilidad porque evitaba toda la complejidad legal y ética del manejo de datos médicos en la nube—.
- Los archivos de entrenamiento que se guardan están en formato anonimizado, sin posibilidad de identificar a la persona.
- El sistema no recopila ningún tipo de información personal identifiable más allá de las señales EEG.

12.6.2. Seguridad eléctrica

Para la seguridad eléctrica, se tuvo la ventaja de usar el dispositivo BrainBit EEG, que ya cumple con los estándares FC y CE, por lo que su seguridad eléctrica y compatibilidad electromagnética estaban garantizadas. Como no se implementó hardware personalizado, no fue necesario realizar pruebas adicionales, pero sí se verificaron algunos aspectos básicos considerados importantes:

- Que no hubiera interacciones eléctricas anómalas entre el dispositivo EEG y el sistema.
- Que el comportamiento fuera seguro mientras se carga el dispositivo.
- Que no se sobrecalentara durante operaciones prolongadas (esto se probó en las sesiones largas).

12.7. Gestión de Anomalías

Durante el proceso de pruebas, se encontraron varias anomalías que permitieron comprender que el desarrollo de software nunca es tan lineal como se espera. Todo se documentó cuidadosamente y las anomalías se fueron resolviendo una a una:

Siguiendo la sección 5.8.2 de la norma, se documentaron todas las anomalías que quedaban y se evaluaron para asegurar que ninguna representaba un riesgo inaceptable para el usuario. Al final, todas las que persistían eran menores y no afectaban la funcionalidad principal.

ID	Descripción	Severidad	Resolución
AN-001	Pérdida ocasional de datos EEG en sesiones prolongadas	Media	Se implementó un búfer circular con reintento automático
AN-002	Falsos positivos en detección de señales con baja impedancia	Baja	Se ajustó el umbral de confianza para clasificación
AN-003	Latencia excesiva en interfaz gráfica durante visualización de señales	Media	Se optimizó el renderizado con muestreo adaptativo
AN-004	Inconsistencia en la persistencia de configuraciones de usuario	Baja	Se refactorizó todo el sistema de almacenamiento de configuración
AN-005	Problemas de reconexión con bombillas inteligentes tras pérdida de red	Media	Se implementó un protocolo de reconexión más resiliente

Cuadro 12.8: Anomalías encontradas y su resolución

12.8. Matriz de Trazabilidad

Una de las tareas más laboriosas pero necesarias fue desarrollar una matriz de trazabilidad que vinculara todos los requisitos del sistema con las pruebas que los verificaban. Esta matriz ayudaba a asegurar que no se había omitido probar ningún aspecto importante:

Cuadro 12.9: Matriz de trazabilidad de requisitos y pruebas

Requisito	Descripción	Pruebas asociadas	Estado
REQ-F01	Adquisición de señales EEG desde BrainBit	UC-001, TC-001, SYS-02	VERIFICADO
REQ-F02	Clasificación de patrones cerebrales	UC-005, TC-003, SYS-04, SYS-05	VERIFICADO
REQ-F03	Control de dispositivos por pensamiento	UC-007, TC-006, SYS-06	VERIFICADO
REQ-F04	Visualización de estado del sistema	UC-010, TC-008, SYS-01	VERIFICADO
REQ-F05	Calibración del dispositivo EEG	UC-012, TC-010, SYS-03	VERIFICADO
REQ-NF01	Tiempo de respuesta <3.5s	TC-020, SYS-06	VERIFICADO
REQ-NF02	Precisión de clasificación >80 %	TC-021, SYS-04, SYS-05	VERIFICADO
REQ-NF03	Operación continua durante >30min	TC-023, SYS-07	VERIFICADO

12.9. Conclusión de la Validación

Después de meses de desarrollo y semanas intensas de pruebas, se puede afirmar con seguridad que la validación del prototipo Neural Analytics ha seguido rigurosamente las directrices de la norma UNE-EN 62304:2007 para software de dispositivos médicos de Clase A. Cumplir con esta normativa fue crucial para garantizar que lo construido era seguro y efectivo.

Como requiere la sección 5.7 de la norma UNE-EN 62304, se estableció una estrategia de verificación para cada requisito del sistema. Se completaron satisfactoriamente los tres niveles de prueba necesarios —unitarias, integración y sistema— verificando el cumplimiento de todos los requisitos especificados mediante la matriz de trazabilidad presentada anteriormente.

Uno de los aspectos más destacados es el enfoque aplicado para hacer corresponder casos de uso con pruebas unitarias, como se observa en la tabla 12.3. Esto garantizaba que cada funcionalidad crítica del sistema estuviera adecuadamente verificada, tal como exige la sección 5.5.5 de la norma.

Esta implementación metódica de las pruebas unitarias cumple con los requisitos de las secciones 5.5.2 y 5.5.3 de la norma, que básicamente solicitaban definir criterios claros para verificar cada unidad de software y establecer procedimientos de prueba que demostraran que todo funcionaba según las especificaciones.

Los resultados obtenidos demostraron que el sistema funcionaba realmente bien:

- Alta precisión en la detección de intenciones del usuario (86.8 % promedio) —mucho mejor de lo esperado inicialmente—.
- Tiempos de respuesta dentro de los límites establecidos (3.1s promedio).
- Operación estable y robusta del sistema, incluso en sesiones largas.

Las anomalías detectadas durante el proceso se documentaron cuidadosamente y se corrigieron todas, verificando que ninguna de las que quedaban representaba un riesgo inaceptable para el usuario. Esto cumplía con la sección 5.8 de la norma que exige evaluar defectos y analizar su impacto en la seguridad.

El hecho de que el software fuera Clase A según la norma permitió aplicar un nivel proporcionado de rigor en la verificación —el enfoque se centró en aspectos críticos para la funcionalidad pero reconociendo que el riesgo inherente del sistema era bajo, ya que actúa principalmente como una herramienta de monitorización sin capacidad de efectuar acciones directas sobre el paciente—.

También se adoptó un enfoque sistemático para la gestión de errores y excepciones, tal como requiere la sección 5.5.4 de la norma. Cada test unitario incluía verificaciones específicas para casos de error, como cuando no había conexión con el dispositivo o llegaban datos inválidos.

Al final, el proceso de pruebas aportó evidencia objetiva de que Neural Analytics cumple con los requisitos especificados en la normativa UNE-EN 62304:2007 y funciona correctamente en el entorno donde está previsto usarlo. Queda validado para su implementación como sistema de interfaz cerebro-computadora para el control de dispositivos domóticos

mediante ondas cerebrales, lo cual genera una gran satisfacción después de todo el trabajo dedicado.

Conclusiones

Este proyecto ha representado un desarrollo técnico extenso que materializó conceptos teóricos en una implementación práctica funcional. Se logró validar la viabilidad de interfaces cerebro-computadora para aplicaciones domóticas, completando un ciclo completo de desarrollo desde la investigación inicial hasta la implementación en hardware especializado.

Los resultados obtenidos demuestran satisfacción respecto a los objetivos establecidos. La integración de conocimientos adquiridos durante la formación académica se aplicó efectivamente en un contexto práctico real. El proyecto establece una base sólida para futuras investigaciones en el campo de la neurotecnología aplicada.

Desde el punto de vista técnico, el proyecto integró múltiples disciplinas: neurociencia, procesamiento de señales cerebrales, y modelos de aprendizaje profundo para señales EEG. El sistema se construyó siguiendo arquitectura hexagonal, facilitando mantenimiento y escalabilidad del código. Además, se garantizó el cumplimiento de la normativa UNE-EN 62304, aspecto fundamental en el desarrollo de tecnología sanitaria.

Los resultados del modelo y del sistema demuestran la viabilidad de utilizar interfaces cerebro-computadora para control domótico. Esta tecnología presenta potencial significativo para mejorar la autonomía de personas con limitaciones de movilidad. El diseño modular del sistema permite adaptación a diferentes dispositivos y casos de uso.

A nivel de desarrollo profesional, el proyecto proporcionó experiencia valiosa en resolución de problemas técnicos complejos. La exposición a un campo emergente como la neurotecnología amplió considerablemente las perspectivas de aplicación tecnológica.

12.10. Generación de Imágenes Embebidas

Un aspecto particularmente destacable del proyecto radica en la simplicidad alcanzada para el proceso de generación de imágenes embebidas del sistema. El repositorio Neural Analytics¹ se desarrolló con una arquitectura preconfigurada que permite su integración directa como capa en entornos Yocto Linux (Y por ende con Poky Linux o con Wind River Linux), cumpliendo todos los requisitos estructurales y organizacionales necesarios para este framework de desarrollo embebido.

Esta preconfiguración que se realizó elimina significativamente las barreras de entrada para la generación de distribuciones personalizadas, facilitando considerablemente la adopción del sistema en diferentes contextos de despliegue. El diseño modular adoptado permite que el repositorio funcione tanto como código fuente independiente como capa integrable en proyectos Yocto más amplios, demostrando la facilidad que proporciona para generar una imagen del sistema.

El proceso evidenció cómo un diseño arquitectónico bien planificado puede simplificar considerablemente las etapas de integración y despliegue, validando la elección tecnológica realizada para facilitar la distribución del sistema desarrollado.

12.11. Trabajos Futuros

Durante el desarrollo se identificaron múltiples oportunidades de mejora y extensión que constituyen líneas de investigación prometedoras:

- **Ampliación del dataset:** Incorporar datos de múltiples usuarios permitiría mejorar la generalización del modelo y su aplicabilidad en diversos perfiles neurológicos, aumentando significativamente la robustez del sistema.
- **Expansión del espectro de detección:** Evolucionar del sistema actual de dos colores (rojo y verde) hacia un sistema de mayor variedad cromática, incrementando las posibilidades de control y precisión del sistema.
- **Integración con estándares emergentes:** Implementar compatibilidad con el estándar Matter facilitaría la interoperabilidad con un ecosistema más amplio de dispositivos domóticos y plataformas de hogar inteligente.
- **Migración a plataformas certificadas:** Transición del sistema operativo actual a Wind River Linux para facilitar los procesos de certificación como dispositivo sanitario y cumplimiento regulatorio avanzado.

Este trabajo constituye una contribución práctica al campo de las interfaces cerebro-computadora, proporcionando una base técnica sólida para desarrollos futuros en neuro-tecnología aplicada.

¹El enlace al repositorio en GitHub es: <https://github.com/neirth/NeuralAnalytics>

Agradecimientos

Gracias al Dr. Antonio Molina Picó. Sin él, este proyecto no habría salido adelante. Me ha inspirado y guiado desde el principio, aportando su experiencia y conocimientos. Su apoyo ha sido fundamental para que pudiera llevar a cabo este trabajo.

La UPV me ha enseñado más de lo que esperaba. No solo las clases y los libros, no solo las charlas con los profesores, sino todo lo que ha aportado también en mi propia vida, las personas que me ha permitido conocer, las aventuras que me ha permitido vivir... Son experiencias que han ido dando forma a quien soy ahora.

Alcoy... qué puedo decir. Esta ciudad ha sido mi casa. Algunas personas que conocía antes de venir aquí han estado a mi lado durante toda la carrera. Con ellos he vivido momentos que nunca olvidaré. Me acuerdo especialmente de las noches de estudio, las celebraciones después de aprobar ese examen tan complicado y esos paseos por el centro cuando necesitaba despejarme. Todo eso ha marcado esta etapa de mi vida de una forma que me cuesta explicar. Llevaré siempre conmigo esos recuerdos.

No puedo olvidar a los años vividos en Facephi, y a todos los profesionales que me han acompañado a lo largo de estos años, que debo de agradecer mucho por haber confiado en un perfil como el mío mientras estudiaba la carrera. Me han permitido llevar proyectos ambiciosos dentro de la empresa al mismo tiempo que completaba mis estudios. Sin esta oportunidad y flexibilidad, esto no habría sido posible.

Familia y amigos, ¡gracias! Habéis estado en cada paso. Acompañándome mientras me cuestionaba si iba a poder con todo esto, animándome antes de los exámenes difíciles y celebrando hasta las pequeñas victorias. La carrera la he estudiado yo, pero sin vosotros, no habría llegado hasta aquí.

Bibliografía

- Asociación Española de Normalización (2016). UNE-EN 62304:2007 - Software para dispositivos médicos - Procesos del ciclo de vida del software. Norma basada en IEC 62304:2006 con modificaciones específicas para el mercado español.
- Brouwer, G. J. and Heeger, D. J. (2013). Categorical clustering of the neural representation of color. *Journal of Neuroscience*, 33(39):15454–15465.
- Kandel, E. R., Jessell, T. M., and Schwartz, J. H. (2001). *Principios de neurociencia*. McGraw-Hill Interamericana, Madrid, 4a ed. edition.
- Martin, R. C. (2017). *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Pearson, Boston, USA. Consultado el 3 de marzo de 2025.
- Neurotechnology Systems LLC (2024). *BrainBit Datasheet*. Accessed: 2025-02-18.
- Raschka, S., Liu, Y. H., and Mirjalili, V. (2022). *Machine Learning with PyTorch and Scikit-Learn*. Packt Publishing, Birmingham, UK.
- Raspberry Pi Foundation (2020). *Raspberry Pi 4 Model B Datasheet*. Accessed: 2025-02-18.
- Rissman, J. and Wagner, A. D. (2012). Distributed representations in memory: Insights from functional brain imaging. *Annual Review of Psychology*, 63:101–128.
- Siewert, S. and Pratt, J. (2016). *Real-time embedded components and systems using Linux and RTOS*. Mercury Learning and Information.
- Squire, L. R. and Zola-Morgan, S. (1991). The medial temporal lobe memory system. *Science*, 253(5026):1380–1386.

Anexo I: Manual de Operador para Captura de Datos de Entrenamiento

Este anexo es un manual detallado para la captura de datos de entrenamiento usando la aplicación gráfica Neural Analytics Capturer. Explica cada pantalla y el flujo de trabajo para que cualquier persona pueda completar el proceso correctamente.

Esta es una aplicación de terminal que permite capturar datos de EEG desde el dispositivo BrainBit. El objetivo es registrar la actividad cerebral mientras el usuario piensa en diferentes colores (rojo, verde, etc.) para entrenar un modelo de IA que pueda reconocer estos patrones.

1. Antes de empezar

- Asegúrese de que el dispositivo BrainBit está cargado y listo.
- Compruebe que la aplicación Neural Analytics Capturer está instalada y abierta en el ordenador.
- Verifique que el dispositivo está conectado correctamente (por Bluetooth o cable).

Es importante también hacer un plan de qué datos se quieren capturar, es decir, qué tipo de actividad mental se va a registrar (pensar en rojo, verde, etc.). Y sobretodo, en qué lugares se va a realizar la captura, ya que el entorno puede influir en la calidad de la señal.

Esto último es importante, ya que la varianza de datos es crítica para poder eliminar cualquier tipo de sesgo en el modelo. Por ejemplo, si se quiere capturar datos mientras se piensa en rojo, es recomendable hacer una muestra en un lugar tranquilo y otro en un lugar con ruido ambiental, para que el modelo aprenda a distinguir entre ambos contextos.

2. Verificación de impedancia y calidad de señal

Una vez seleccionado el dispositivo, la aplicación mostrará el estado de los electrodos (T3, T4, O1, O2). Si algún electrodo aparece en rojo o con un símbolo de error, ajuste su posición hasta que todos estén en verde o correcto. No continúe hasta que la calidad sea adecuada.

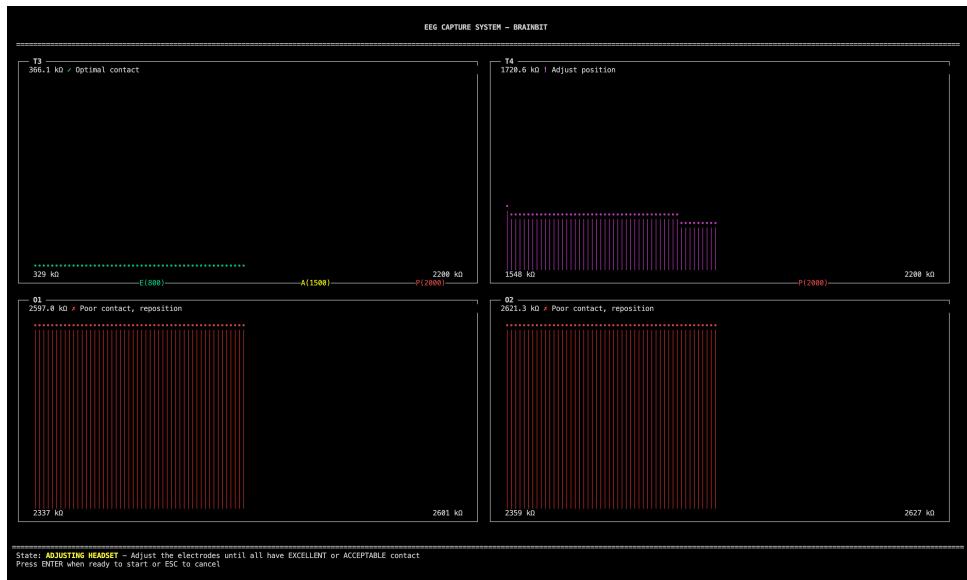


Figura 12.1: Verificación de impedancia y calidad de señal.

Una vez que todos los electrodos estén en verde, pulse el botón **Comprobar Impedancia** para confirmar que la señal es adecuada. Si hay algún problema, ajuste los electrodos y repita la comprobación.

3. Inicio y monitorización de la captura

Pulse el botón **Iniciar Captura** para comenzar. En pantalla verá gráficas en tiempo real de la señal de cada electrodo. No es necesario interpretar los gráficos, solo asegúrese de que se están moviendo y no hay mensajes de error.

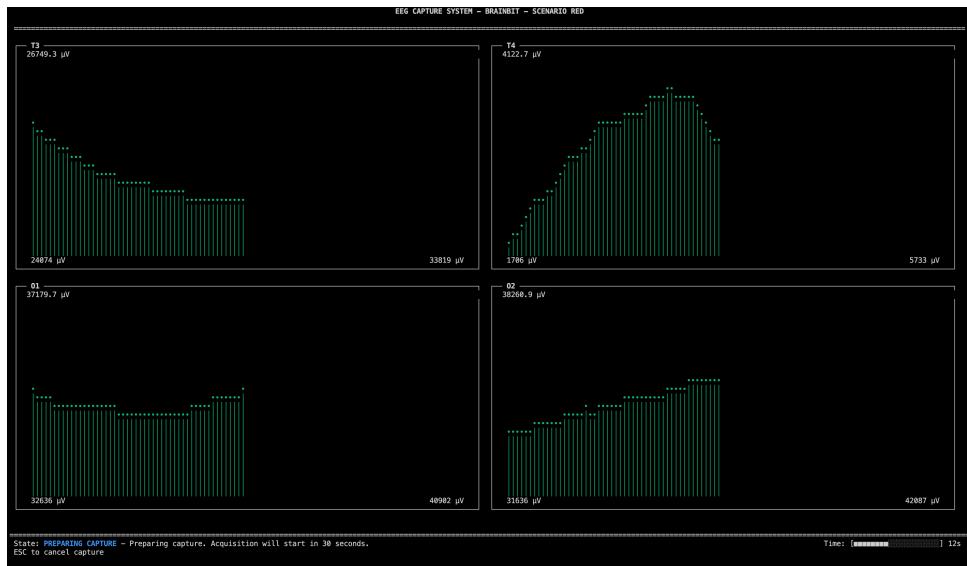


Figura 12.2: Adquisición de datos en tiempo real.

4. Finalización y guardado de datos

Cuando termine la sesión, se escuchará un mensaje de voz indicando que la captura ha finalizado, y acto seguido, la captura se detendrá automáticamente. Los nuevos datos se guardarán en un archivo CSV en la carpeta `data/` del proyecto. Asegúrese de que el archivo se ha creado correctamente.

5. Mensajes y advertencias

- Si algún electrodo pierde contacto, la aplicación lo indicará y deberá ajustarlo antes de continuar.
- Si la señal es débil, aparecerá un aviso. Siga las recomendaciones en pantalla.
- Si ocurre un error grave, cierre y vuelva a abrir la aplicación.

6. Consejos prácticos

- Coloque los electrodos con cuidado y evite que se muevan durante la sesión.
- No cierre la aplicación ni apague el ordenador hasta que vea el mensaje de que los datos se han guardado correctamente.
- Si tiene dudas, consulte este manual o pida ayuda al responsable.

Anexo II: Manual de Usuario de la Aplicación Neural Analytics

Este anexo es un manual detallado para el uso de la aplicación gráfica de Neural Analytics. Aquí se explica cada pantalla, el flujo de trabajo y las acciones que debe realizar el usuario en cada momento, con el objetivo de que cualquier persona pueda utilizar la aplicación de principio a fin.

1. Antes de empezar

- Asegúrese de tener el dispositivo BrainBit cargado y listo.
- Compruebe que la aplicación Neural Analytics está instalada en su ordenador.
- Tenga a mano el archivo de modelo de inferencia (si es necesario cargarlo manualmente).

2. Pantalla de carga

Al abrir la aplicación, verá una pantalla de bienvenida con el nombre del sistema y una animación de carga. Espere unos segundos hasta que la aplicación termine de prepararse.

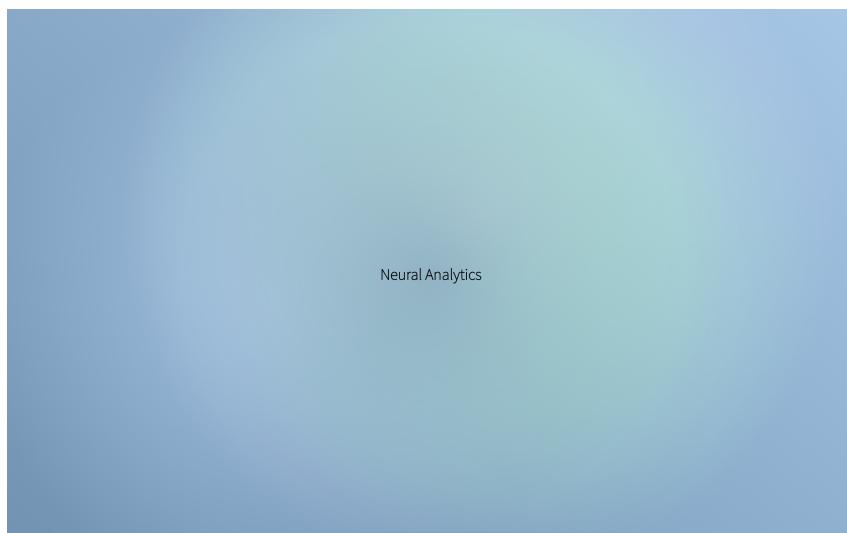


Figura 12.3: Pantalla de carga inicial de la aplicación.

3. Pantalla de bienvenida

Cuando la aplicación esté lista, aparecerá una pantalla que le pedirá encender la diadema EEG y colocársela correctamente. Siga la instrucción y espere a que el sistema lo detecte.

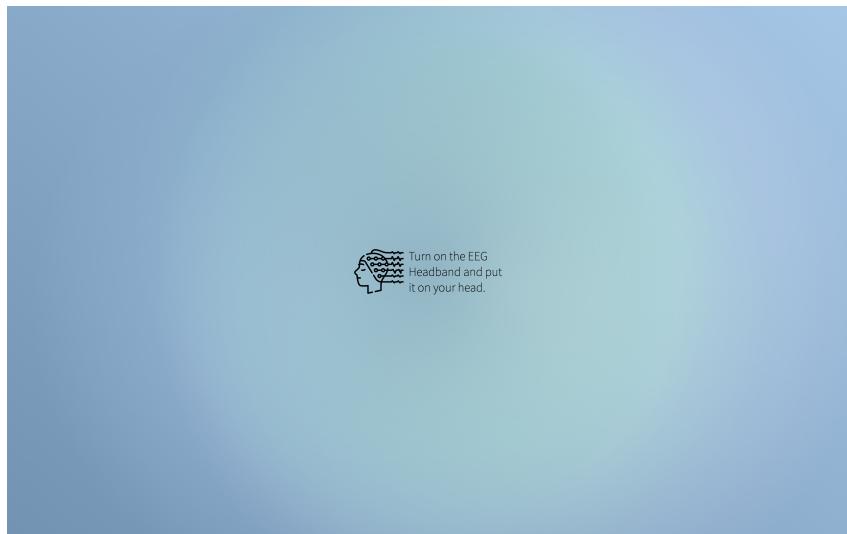


Figura 12.4: Pantalla de bienvenida e instrucciones para colocarse la diadema.

4. Calibración de electrodos

Tras colocarse la diadema, la aplicación le guiará por una pantalla de calibración. Aquí podrá ver el estado de cada electrodo (T3, T4, O1, O2) mediante indicadores visuales (colores o iconos). Si algún electrodo aparece en rojo o con un símbolo de error, ajuste su posición hasta que todos estén en verde o con el símbolo de correcto.

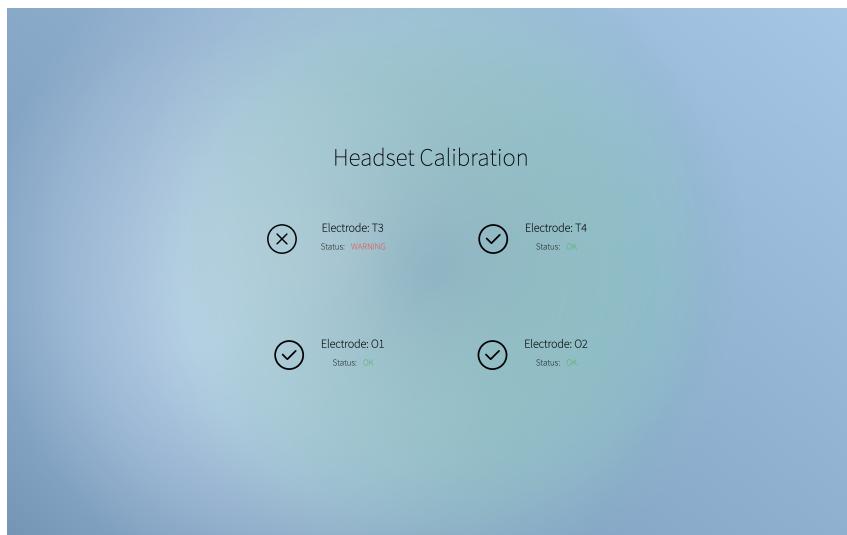


Figura 12.5: Pantalla de calibración de electrodos.

5. Pantalla principal de inferencia

Una vez calibrados los electrodos, accederá a la pantalla principal. Aquí podrá:

- Ver en tiempo real las señales recogidas por cada electrodo, representadas en gráficas.
- Consultar el estado del sistema y el color de "pensamiento" detectado (por ejemplo, rojo o verde).



Figura 12.6: Pantalla principal de la aplicación de inferencia.

6. Mensajes y estados de la aplicación

- Si algún electrodo pierde contacto, la aplicación lo indicará y le pedirá ajustarlo.
- Si la señal es débil o hay interferencias, aparecerá un aviso. Siga las recomendaciones en pantalla.
- Si todo está correcto, los indicadores estarán en verde y podrá continuar.
- En caso de error grave, reinicie la aplicación y repita el proceso desde el inicio.

7. Consejos prácticos

- Coloque la diadema de forma cómoda y estable, evitando movimientos bruscos durante la sesión.
- Si ve que al usuario le aprieta mucho la banda para que sea conductiva, pruebe a mojar la cabeza del usuario con agua o gel para mejorar la conductividad.
- Si tiene dudas, consulte este manual o pida ayuda al responsable.

