UNIVERSIDAD POLITÉCNICA DE MADRID

E.T.S. DE INGENIERÍA DE SISTEMAS INFORMÁTICOS

PROYECTO FIN DE GRADO

NO DEFINIDO

VETTA: Sistema de generación automática de problemas de escalada mediante visión por computador e inteligencia artificial

Desarrollado por: Israel Estero Agueda

Dirigido por: Edgar Talavera Muñoz

Madrid, 21 de octubre de 2025

VETTA: Sistema de generación automática de problemas de escalada mediante visión por computador e inteligencia artificial
Desarrollado por: Israel Estero Agueda
Dirigido por: Edgar Talavera Muñoz
Proyecto Fin de Grado, 21 de octubre de 2025

E.T.S. de Ingeniería de Sistemas Informáticos
Campus Sur UPM, Carretera de Valencia (A-3), km. 7
28031, Madrid, España

Si deseas citar este trabajo, la entrada completa en BibTeX es la siguiente:

```
@mastersthesis{citekey,
    title = {VETTA: Sistema de generación automática de problemas de escalada
mediante visión por computador e inteligencia artificial },
    type = {Bachelor's Thesis},
    author = {},
    school = {E.T.S. de Ingeniería de Sistemas Informáticos},
    year = {2025},
    month = {10},
}
```

Agradecimientos

51-75-69-65-72-6f-20-65-78-70-72-65-73-61-72-20-6d-69-20-6d-61-73-20-73-69-6e-63-65-72-6f-20-61-67-72-61-64-65-63-69-6d-69-65-6e-74-6f-20-61-20-6d-69-20-66-61-6d-69-6c-69-61-20-79-20-61-6d-69-67-6f-73-20-70-6f-72-20-73-75-20-69-6e-66-69-6e-69-74-61-20-70-61-63-69-65-6e-63-69-61-20-79-20-61-70-6f-79-6f-2e-20-46-69-6e-61-6c-6d-65-6e-74-65-2c-20-75-6e-20-61-67-72-61-64-65-63-69-6d-69-65-6e-74-6f-20-65-73-70-65-63-69-61-6c-20-61-20-6c-61-20-65-73-63-61-6c-61-64-61-2c-20-6c-61-20-70-61-73-69-f3-6e-20-71-75-65-20-6d-65-20-69-6d-70-75-6c-73-f3-20-61-20-74-72-61-6e-73-66-6f-72-6d-61-72-20-63-61-64-61-20-70-72-6f-62-6c-65-6d-61-20-64-65-20-62-fa-6c-64-65-72-20-65-6e-20-75-6e-20-64-65-73-61-66-ed-6f-20-64-65-20-63-f3-64-69-67-6f-2c-20-63-6f-6e-76-69-72-74-69-65-6e-64-6f-20-6d-69-20-61-66-69-63-69-f3-6e-20-65-6e-20-65-6c-20-63-6f-72-61-7a-f3-6e-20-64-65-20-65-73-74-65-20-70-72-6f-79-65-63-74-6f-2e

Resumen

La escalada indoor ha visto un auge en popularidad, pero la creación de nuevas rutas o problemas de escalada (route setting) sigue siendo un proceso mayoritariamente manual, costoso y dependiente de personal cualificado. Este Proyecto de Fin de Grado aborda este desafío mediante el desarrollo de VETTA, un sistema integral para la generación automática de problemas de escalada. Los objetivos principales del proyecto fueron: desarrollar un modelo de visión por computador para la detección precisa de presas en un muro, implementar un sistema de inteligencia artificial capaz de generar rutas coherentes y ajustadas a una dificultad específica, y encapsular la solución en una aplicación móvil Android intuitiva para el usuario final. La metodología empleada se basó en una arquitectura cliente-servidor. Para la detección de presas, se entrenó un modelo de aprendizaje profundo basado en la arquitectura Mask R-CNN, que realiza una segmentación de instancia sobre una imagen del muro para identificar la posición y forma de cada presa. Posteriormente, un segundo modelo de aprendizaje automático, entrenado con un conjunto de datos de problemas existentes, analiza las características espaciales de las presas detectadas para generar secuencias de movimientos que constituyen una nueva ruta de una dificultad determinada. La interacción con el sistema se realiza a través de una aplicación nativa para Android desarrollada en Kotlin, la cual se comunica con un backend programado en Python y Flask que aloja los modelos de IA. Como resultado, se ha obtenido un prototipo funcional del sistema VETTA. El modelo de detección de presas ha demostrado una alta precisión en la identificación de los agarres en diferentes condiciones de iluminación. El sistema de generación es capaz de producir problemas de búlder viables y correctamente categorizados por su dificultad (escala V). La aplicación móvil se integra satisfactoriamente con el backend, ofreciendo una experiencia de usuario fluida desde la captura de la imagen hasta la visualización de la ruta generada. La principal conclusión de este trabajo es que la aplicación de técnicas de visión por computador e inteligencia artificial es una solución viable y eficaz para automatizar y democratizar el proceso de route setting. El sistema VETTA no solo ofrece una herramienta para reducir costes y aumentar la variedad de rutas en los rocódromos, sino que también abre nuevas vías para el entrenamiento personalizado y la gamificación de la escalada. Este proyecto demuestra el potencial de la tecnología para enriquecer la experiencia deportiva de una manera accesible e innovadora.

Palabras clave: Escalada Indoor; Visión por Computador; Inteligencia Artificial; Generación de Rutas; Android

Abstract

Indoor climbing has surged in popularity, yet the creation of new climbing routes or problems (route setting) remains a largely manual, costly, and expert-dependent process. This Final Degree Project addresses this challenge through the development of VETTA, a comprehensive system for the automatic generation of climbing problems. The main objectives of the project were: to develop a computer vision model for the accurate detection of holds on a climbing wall, to implement an artificial intelligence system capable of generating coherent routes adjusted to a specific difficulty, and to encapsulate the solution in an intuitive Android mobile application for the end-user. The methodology was based on a client-server architecture. For hold detection, a deep learning model based on the Mask R-CNN architecture was trained to perform instance segmentation on an image of the wall, identifying the position and shape of each hold. Subsequently, a second machine learning model, trained with a dataset of existing problems, analyzes the spatial characteristics of the detected holds to generate sequences of movements that constitute a new route of a given difficulty. Interaction with the system is handled through a native Android application developed in Kotlin, which communicates with a Python and Flask backend that hosts the AI models. As a result, a functional prototype of the VETTA system has been successfully developed. The hold detection model has demonstrated high precision in identifying holds under various lighting conditions. The generation system is capable of producing viable and correctly graded (V-scale) boulder problems. The mobile application integrates successfully with the backend, offering a seamless user experience from image capture to route visualization. The main conclusion of this work is that the application of computer vision and artificial intelligence techniques is a viable and effective solution to automate and democratize the route setting process. The VETTA system not only provides a tool to reduce costs and increase route variety in climbing gyms but also opens new avenues for personalized training and the gamification of climbing. This project demonstrates the potential of technology to enrich the sporting experience in an accessible and innovative way.

Keywords: Indoor Climbing; Computer Vision; Artificial Intelligence; Route Generation; Android

Índice general

Índice de figuras

Índice de tablas

Índice de listados

1. Introducción

La escalada, especialmente en su modalidad indoor o de interior, ha experimentado un crecimiento exponencial en popularidad durante la última década, consolidándose no solo como un deporte olímpico, sino como una actividad de ocio y entrenamiento para miles de personas. El corazón de esta disciplina reside en los Búlder o problemas: secuencias de Presa (o Agarre)s que trazan un camino sobre el muro y que suponen un desafío físico y mental. Tradicionalmente, el diseño de estos problemas, conocido como Route Setting, es un proceso artesanal, subjetivo y laborioso. Requiere de expertos cualificados que, de forma manual, deben desmontar, limpiar y volver a atornillar las presas para crear nuevas rutas, un proceso que no solo es costoso en tiempo y recursos, sino que también limita la variedad y frecuencia con la que los rocódromos pueden renovar sus desafíos.

Este proyecto, VETTA, nace para dar respuesta directa a esta necesidad, proponiendo un cambio de paradigma: pasar de la creación manual y estática a la generación automática y dinámica de problemas de escalada. Se ha desarrollado un sistema integral que, mediante la aplicación de técnicas de visión por computador e Inteligencia Artificial (IA) [1], es capaz de analizar un muro de escalada, identificar las presas disponibles y generar, de forma automática, nuevos problemas de búlder adaptados a una dificultad específica. El sistema se compone de un potente backend de análisis de imagen y aprendizaje automático, programado en Python, y una aplicación móvil nativa para Android desarrollada en Kotlin, que actúa como interfaz para el usuario. El flujo de trabajo es sencillo e intuitivo: el usuario captura una imagen del muro con la aplicación, esta se envía al backend, que la procesa con los modelos de IA y devuelve una nueva ruta, que se visualiza directamente en la pantalla del dispositivo.

Como gancho para potenciar la experiencia y visión a futuro, el sistema también contempla la posibilidad de utilizar la Realidad Aumentada (AR) para proyectar la ruta generada directamente sobre el muro físico, una idea explorada en trabajos como el de [2].

Dentro de este capítulo, se establecerá el marco completo del trabajo realizado. Se comenzará detallando la motivación que impulsa el proyecto y los objetivos específicos que se han perseguido para materializar la solución. A continuación, se justificará la

relevancia del proyecto en el contexto tecnológico y deportivo actual y, finalmente, se describirá la estructura que seguirá esta memoria para guiar al lector a través de todas las fases del desarrollo.

1.1. Objetivos

La finalidad principal de este Proyecto de Fin de Grado (PFG) es el **desarrollo** de un sistema funcional, denominado VETTA, capaz de generar problemas de escalada de forma automática. Para garantizar la consecución de este fin, y siguiendo una metodología que permita evaluar el éxito del proyecto, se han definido los siguientes objetivos específicos. Estos objetivos se han diseñado para ser medibles, alcanzables, relevantes y acotados en el tiempo, asegurando un camino claro para el desarrollo y la validación del sistema.

- **Desarrollar un modelo de visión por computador para la detección y segmentación precisa de presas.** El primer pilar del proyecto es la capacidad del sistema para 'entender' un muro de escalada. El objetivo no es solo detectar la presencia de presas, sino realizar una **segmentación de instancia** que delimite el contorno exacto de cada una. Esta precisión es fundamental, ya que la Máscara de Segmentación es la base para el análisis de la forma del agarre y para la futura funcionalidad de proyección. El modelo deberá alcanzar un Precisión Media Promedio (mean Average Precision) (mAP) (Precisión Media Promedio) en el conjunto de validación que demuestre su robustez para ser útil en la generación de rutas.
- **Implementar un sistema de IA para la generación inteligente de problemas de búlder.** El núcleo creativo del proyecto reside en este objetivo. El sistema no debe generar rutas aleatorias, sino aprender de problemas existentes para replicar la coherencia y el estilo de un Route Setting humano. Para ello, el objetivo es implementar un modelo de aprendizaje no supervisado (clustering) que identifique arquetipos de dificultad y, posteriormente, un sistema de recomendación que genere nuevas secuencias de presas. El éxito de este objetivo se medirá cualitativamente, evaluando si las rutas generadas son lógicas, escalables y se ajustan al grado de dificultad (escala V) solicitado por el usuario.
- **Crear una aplicación móvil nativa para la plataforma Android como interfaz de usuario.** Para que el sistema sea accesible, es fundamental encapsular su funcionalidad en una aplicación intuitiva y fluida. El objetivo es desarrollar una aplicación

en Kotlin que gestione el flujo completo del usuario: desde la autenticación y la captura de una imagen del muro, pasando por la comunicación asíncrona con el *backend* de IA, hasta la visualización clara e interactiva de la ruta generada sobre la foto original.

- **Diseñar una arquitectura de software extensible para una futura funcionalidad de proyección.** Si bien la implementación completa de la proyección de AR sobre el muro físico queda como línea futura, un objetivo clave de este proyecto es diseñar el software de tal manera que esta integración sea factible sin necesidad de una reingeniería completa. Para ello, se deberá crear una interfaz de software de proyección generalizada dentro de la aplicación Android, que desacople la lógica de visualización de la implementación específica de un proyector. Esto demuestra una planificación a largo plazo y asegura la escalabilidad del proyecto.

1.2. Motivación

La motivación detrás de VETTA se fundamenta en el impacto que este trabajo puede tener en el entorno tecnológico y deportivo de la escalada. Más allá de un interés puramente personal, el proyecto nace de la identificación de varias necesidades y oportunidades clave en la comunidad escaladora actual:

- **Democratización y Eficiencia en el Route Setting**: En la gestión de un rocódromo, el Route Setting es un cuello de botella operativo. Es una tarea que consume mucho tiempo, requiere personal altamente cualificado y, por tanto, tiene un coste elevado. Esto provoca que muchos gimnasios, especialmente los de menor tamaño, no puedan renovar sus problemas con la frecuencia que a sus usuarios les gustaría. VETTA ofrece una solución a este problema al permitir generar una infinidad de rutas virtuales sobre una misma configuración física de presas. Esto no solo reduce los costes operativos, sino que también democratiza el acceso a una oferta de búlders variada y constante, aumentando la vida útil y el atractivo de las instalaciones.
- **Personalización y Optimización del Entrenamiento**: El progreso de un escalador a menudo se estanca debido a una oferta de rutas estática que no se ajusta a sus necesidades específicas de entrenamiento. Si un escalador necesita practicar un tipo de movimiento concreto o trabajar en un grado de dificultad específico,

depende de que el Route Setter haya equipado un problema adecuado. VETTA transforma esta dinámica, convirtiéndose en una potente herramienta de entrenamiento personalizado. El sistema permite a los usuarios generar problemas 'a la carta', especificando la dificultad exacta que desean entrenar. Esto convierte a la aplicación en un compañero de entrenamiento virtual e inagotable.

- **Innovación Tecnológica Aplicada al Deporte**: El proyecto se sitúa en la intersección de varias tecnologías en auge como son la IA, la visión por computador y el desarrollo de aplicaciones móviles. Aplicar estos campos a un dominio tan físico y kinestésico como la escalada no solo representa un desafío técnico interesante, sino que abre la puerta a nuevas formas de interacción y mejora de la experiencia deportiva [3]. VETTA sirve como un caso de estudio sobre cómo la tecnología puede aumentar una actividad física sin desvirtuarla, enriqueciéndola con una capa de datos e interactividad.

- **Fomento y Digitalización de la Comunidad**: La escalada es un deporte inherentemente social. VETTA tiene el potencial de extender esta comunidad al plano digital. La plataforma permite que los usuarios no solo generen problemas para sí mismos, sino que los guarden, los nombren y los compartan con otros escaladores del mismo gimnasio. Esto puede dar lugar a competiciones amistosas, a la creación de rutas icónicas dentro de una comunidad local y a un sistema de valoración de problemas, añadiendo una dimensión social que aumenta enormemente la implicación (Engagement) y la retención de los usuarios.

1.3. Justificación

La elección y desarrollo de este proyecto se justifica no solo por su relevancia en el contexto deportivo actual, sino por la oportunidad de cubrir una brecha tecnológica y de mercado bien definida. A continuación, se detallan los pilares que sustentan la pertinencia de VETTA.

1. **Relevancia del Tema y Demanda del Sector**: El crecimiento sostenido de la escalada *indoor* ha generado una demanda por parte de los usuarios de experiencias más dinámicas y enriquecedoras. La rotación de problemas es un factor clave para la retención y motivación de los escaladores. Un rocódromo con rutas estáticas pierde rápidamente su atractivo. Este proyecto aborda directamente esta

necesidad crítica, ofreciendo una solución para generar desafíos constantes y personalizados sin la intervención manual y costosa que requiere el Route Setting tradicional.

2. **Justificación Teórica y Fundamento Tecnológico**: El trabajo no parte de cero, sino que se apoya y combina campos de estudio consolidados y en plena ebullición. Para la detección de presas, se utilizan modelos de segmentación de instancia de última generación como **Red Neuronal Convolucional Basada en Regiones (Region-based Convolutional Neural Network) (R-CNN)** [4], cuya implementación de referencia se encuentra en librerías como **Detectron2** [5]. La generación de problemas, por su parte, se fundamenta en técnicas clásicas y robustas de aprendizaje automático no supervisado, como el algoritmo de clustering **K-Means** [6]. La solidez teórica de estos componentes proporciona una base fiable sobre la cual construir una aplicación innovadora.

3. **Brecha en el Conocimiento y Oportunidad de Mercado**: Si bien existen sistemas comerciales de muros de escalada interactivos (como MoonBoard o Kilter Board), estos se basan en un paradigma de hardware específico y costoso: paneles estandarizados, sets de presas fijos e iluminación LED integrada. Esto crea una alta barrera de entrada y excluye a la gran mayoría de rocódromos existentes. VETTA, en cambio, propone una solución puramente basada en software que puede aplicarse a **cualquier muro de escalada existente** sin necesidad de una infraestructura propietaria. Esta aproximación, que prioriza la inteligencia artificial sobre el hardware específico, cubre una importante brecha en el mercado y hace la tecnología de escalada aumentada accesible a un público mucho más amplio, desde grandes gimnasios hasta pequeños muros privados.

4. **Contribución Práctica y Aplicación Directa**: El resultado final del proyecto no es un estudio teórico, sino una herramienta tangible con aplicaciones directas para múltiples perfiles. Para los **gestores de rocódromos**, es una forma de multiplicar su oferta de rutas con una inversión mínima. Para los **entrenadores**, se convierte en una herramienta de planificación para diseñar sesiones de entrenamiento específicas. Para los **escaladores individuales**, es una fuente inagotable de motivación y una forma de autogestionar su progreso. El prototipo funcional desarrollado valida esta contribución práctica de forma inequívoca.

La combinación de estos factores hace que el proyecto VETTA sea una propuesta convincente y necesaria, con el potencial de aportar un valor significativo tanto a la comunidad de escalada como al campo de la tecnología aplicada al deporte.

1.4. Estructura de la memoria

Esta memoria se ha organizado para presentar de manera lógica y secuencial todo el proceso de investigación, diseño y desarrollo del proyecto VETTA. A lo largo de los siguientes capítulos, se detallará cada uno de los componentes del sistema, desde sus fundamentos teóricos hasta su implementación práctica y validación.

El documento comienza con el presente capítulo de **Introducción**, donde se contextualiza el proyecto, se justifica su relevancia y se definen sus metas a través de un conjunto de objetivos claros y medibles.

El Capítulo 2, **Estado de la Cuestión**, analizará las soluciones y tecnologías existentes que forman la base de este trabajo. Se revisarán los sistemas comerciales de escalada interactiva, se profundizará en las arquitecturas de visión por computador para la segmentación de instancia y se explorarán los enfoques para la generación procedural de contenido.

El Capítulo 3, **Diseño y Metodología**, describirá la arquitectura del sistema. Se detallará el enfoque de desarrollo iterativo seguido, desglosando el proyecto en tres fases principales y explicando los requisitos funcionales y no funcionales de cada una. Se explicará el funcionamiento del servidor de IA, el modelo de detección de presas, el algoritmo de generación de problemas y la estructura de la aplicación Android.

El Capítulo 4, **Desarrollo e Implementación**, profundizará en las herramientas y tecnologías específicas utilizadas para materializar el sistema. Se abordarán detalles de implementación de los componentes clave, como el uso de Python con Flask [7] y Detectron2 [5] para el backend, y Kotlin junto con la plataforma Firebase para la aplicación móvil.

En el Capítulo 5, **Pruebas y Resultados**, se presentará la evaluación cuantitativa y cualitativa del sistema. Se analizará el rendimiento del modelo de detección mediante métricas como el mAP, se evaluará la precisión del clasificador de agarres y se valorará la calidad de los problemas generados por la IA a través de un estudio de usuario.

Finalmente, el Capítulo 6 y 7, **Conclusiones y Trabajo Futuro**, resumirá los logros alcanzados en relación con los objetivos planteados. Se discutirán las limitaciones del trabajo actual y se propondrán futuras líneas de investigación y desarrollo para expandir las capacidades del sistema VETTA.

2. Estado de la Cuestión

Para comprender la contribución y el posicionamiento del proyecto VETTA, es imprescindible analizar el panorama tecnológico actual en el que se enmarca. Este capítulo explora el estado de la cuestión en tres áreas fundamentales que sustentan este trabajo: primero, los sistemas de escalada interactivos y comerciales que ya existen en el mercado; segundo, las técnicas de visión por computador para la detección y segmentación de objetos, aplicadas al contexto deportivo; y tercero, el uso de la inteligencia artificial para la generación procedural de contenido, en este caso, rutas de escalada. El análisis de estos campos permitirá identificar las brechas existentes y justificar el enfoque adoptado en VETTA.

2.1. Sistemas de Escalada Interactivos

La idea de enriquecer la experiencia de la escalada indoor con tecnología no es nueva. En los últimos años han surgido varias soluciones comerciales que buscan ofrecer a los escaladores una mayor variedad de problemas y una forma de competir y compartir sus logros. Estos sistemas, aunque muy populares, se basan en un paradigma de hardware específico y estandarizado.

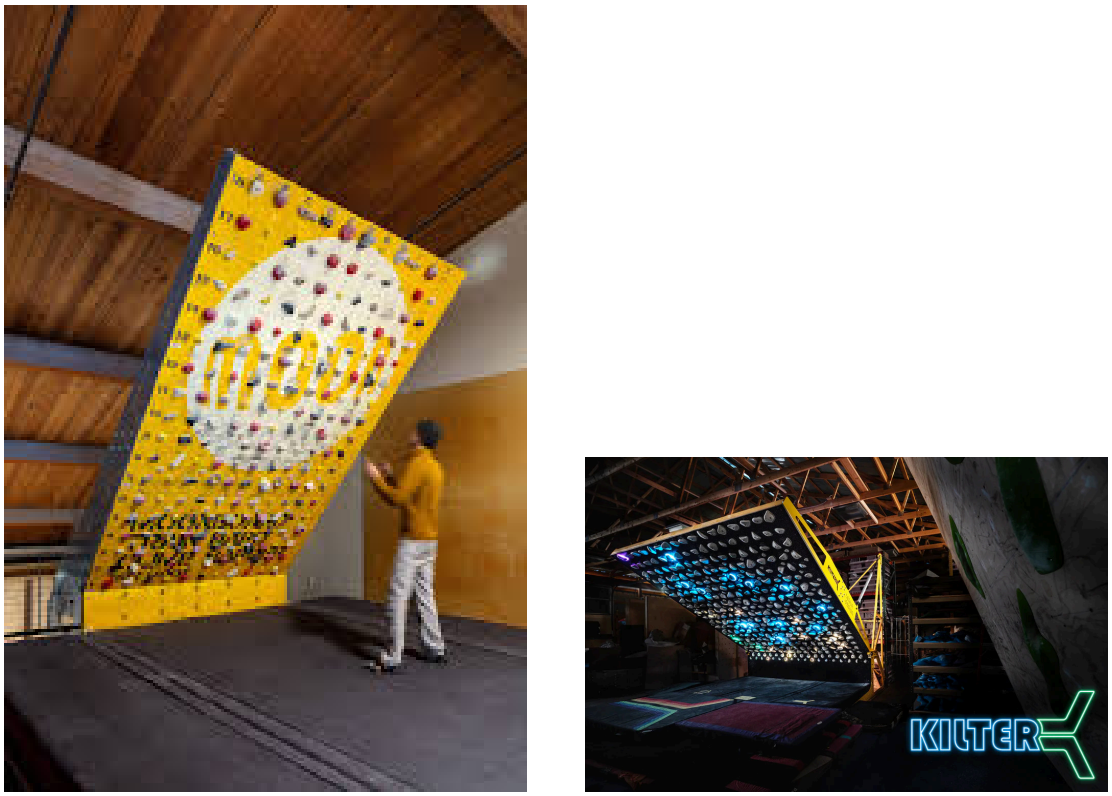
2.1.1. Paneles de entrenamiento estandarizados: MoonBoard y Kilter Board

Los sistemas más extendidos a nivel mundial son el **MoonBoard** (Figura 2.1a) y el **Kilter Board** (Figura 2.1b). Ambos se basan en un concepto similar: un panel de escalada de dimensiones e inclinación fijas (o ajustables, en el caso del Kilter Board) con un conjunto de presas estandarizado y colocado en una disposición inalterable. La innovación clave de estos sistemas es la integración de iluminación LED debajo de cada presa.

- **MoonBoard**: Considerado el pionero en este campo, el MoonBoard consiste en

un muro con una inclinación fija de 40 grados y un set de presas específico. A través de una aplicación móvil, los usuarios pueden acceder a una base de datos con miles de problemas creados por escaladores de todo el mundo. Al seleccionar un problema, la aplicación se conecta al muro vía Bluetooth y enciende los LEDs correspondientes a las presas de inicio (verdes), presas intermedias (azules) y presas finales o "top"(rojas). Este sistema se ha consolidado como una herramienta de entrenamiento de referencia por su alta dificultad y la posibilidad de medir el progreso frente a una comunidad global.

- **Kilter Board**: Siguiendo la estela del MoonBoard, el Kilter Board ofrece una experiencia similar pero con mejoras significativas. Su principal ventaja es que el muro es de inclinación ajustable, pudiendo variar desde los 0 hasta los 70 grados, lo que permite que un mismo problema se pueda escalar con diferentes niveles de dificultad. Además, todo su perímetro de presas está iluminado, ofreciendo una mayor densidad y variedad de problemas. Al igual que el MoonBoard, se controla mediante una aplicación móvil que permite a los usuarios crear, compartir y escalar problemas.



(a) Moon Board. (b) Kilter Board.

Figura 2.1. Paneles de entrenamiento estandarizados con iluminación LED.

El gran éxito de estos sistemas valida la demanda de experiencias de escalada interactivas y conectadas. Sin embargo, su principal limitación, que es a su vez la principal oportunidad para VETTA, es su **dependencia de un hardware propietario y costoso**. La instalación de un MoonBoard o Kilter Board requiere la compra de su panel, su set de presas y su sistema de iluminación, siendo imposible aplicar su tecnología a un rocódromo ya existente con una configuración de presas no estándar.

2.1.2. Escalada Aumentada (Augmented Climbing)

Otra aproximación a la escalada interactiva es la AR aplicada al muro (Figura 2.2), un concepto explorado en trabajos como el de [2]. Estos sistemas utilizan proyectores para superponer elementos visuales sobre un muro de escalada convencional. La aplicación más común es la creación de videojuegos: los escaladores deben golpear o esquivar elementos virtuales que se proyectan sobre el muro, combinando el ejercicio físico con una experiencia lúdica.

Aunque estos sistemas no dependen de un set de presas específico, su enfoque no está puesto en la generación de problemas de escalada graduados y orientados al entrenamiento técnico, sino en el entretenimiento. No suelen incluir una lógica para crear rutas con una dificultad coherente según estándares como la escala V.

En resumen, el estado de la cuestión en sistemas interactivos muestra una clara división: por un lado, sistemas de entrenamiento muy eficaces pero cerrados y dependientes de hardware específico; por otro, sistemas flexibles basados en proyección pero orientados al juego y no al entrenamiento técnico. VETTA se posiciona precisamente en la intersección de ambos mundos, aspirando a ofrecer la generación de problemas de entrenamiento técnico de los primeros con la flexibilidad de poder aplicarse sobre cualquier muro de los segundos.

2.2. Detección de Objetos para la Identificación de Presas

Para que un sistema como VETTA pueda operar sobre cualquier muro, el primer paso fundamental es ser capaz de 'ver' y entender la disposición de las presas. Este desafío

Figura 2.2. Ejemplo de un sistema de Escalada Aumentada.

se enmarca en el campo de la visión por computador [3] y, más concretamente, en la detección y segmentación de objetos.

2.2.1. De la Detección a la Segmentación de Instancia

Los primeros modelos de *deep learning* para la detección de objetos, como la familia de algoritmos R-CNN, se centraban en predecir un **cuadro delimitador** (*bounding box*) alrededor de cada objeto detectado. Sin embargo, para los fines de VETTA, un simple cuadro delimitador resulta insuficiente. Para poder proyectar una luz que se ajuste a

la forma de la presa y para extraer características geométricas más precisas (como la orientación o el área real), es necesario conocer qué píxeles exactos dentro del cuadro pertenecen al objeto.

Esta tarea se conoce como **segmentación de instancia** (*instance segmentation*), y su objetivo es generar una Máscara de Segmentación a nivel de píxel para cada una de las instancias de objeto detectadas.

2.2.2. Mask R-CNN y Detectron2

El algoritmo de referencia y estado del arte para la segmentación de instancia es **R-CNN** [4]. Este modelo extiende a su predecesor, Faster R-CNN, añadiendo una tercera rama en paralelo a las existentes de clasificación y regresión de cuadros delimitadores. Esta nueva rama consiste en una Pequeña Red Totalmente Convolucional (FCN) que toma como entrada la región de interés (RoI) propuesta por la red y genera una máscara binaria para el objeto contenido en ella.

El proyecto VETTA aprovecha la implementación de este modelo disponible en la librería **Detectron2** [5] de Meta AI. Detectron2 es un framework de última generación para la detección y segmentación de objetos que proporciona implementaciones modulares y de alto rendimiento de los modelos más avanzados. La elección de esta tecnología se justifica por su alta precisión y por la capacidad de ser re-entrenada para un dominio específico, como es la detección de presas de escalada, partiendo de modelos pre-entrenados en grandes bases de datos como COCO, lo que acelera y mejora significativamente el proceso de entrenamiento [1].

2.3. Generación de Rutas Mediante Aprendizaje Automático

Una vez que el sistema ha identificado todas las presas disponibles en el muro, el siguiente y más complejo desafío es la creación de un problema de Búlder. Esta no es una tarea trivial que consista en seleccionar presas al azar. Un buen problema de búlder debe cumplir con una serie de características implícitas: debe ser escalable, tener una dificultad consistente, proponer una secuencia de movimientos interesante y, en

general, sentirse como si hubiera sido diseñado por un humano. El estado de la cuestión para la generación de contenido de este tipo, a menudo enmarcado en el campo de la Generación Procedural de Contenido (PCG), abarca desde sistemas basados en reglas hasta complejos modelos generativos.

2.3.1. Sistemas Basados en Reglas y Modelos Generativos

- **Sistemas basados en reglas**: Un enfoque sencillo consistiría en definir un conjunto de reglas heurísticas para seleccionar la siguiente presa (p. ej., "la siguiente presa debe estar a menos de 80 cm de la anterior y por encima de esta"). Si bien son fáciles de implementar, estos sistemas son muy rígidos y no son capaces de capturar la sutileza, la estética y la variedad del Route Setting humano. Generarían rutas funcionalmente posibles, pero probablemente repetitivas y poco interesantes.
- **Modelos generativos profundos**: En el extremo opuesto se encuentran los modelos de aprendizaje profundo como las Redes Neuronales Recurrentes (RNN) o las Redes Generativas Antagónicas (GAN), que podrían aprender a generar secuencias de movimientos o incluso distribuciones enteras de presas. Si bien representan el estado del arte en generación de secuencias y de imágenes [1], requieren enormes cantidades de datos de entrenamiento (miles de problemas de búlder perfectamente etiquetados) y una complejidad computacional y de desarrollo muy elevada, lo que los hace poco prácticos para el alcance de este proyecto.

2.3.2. Enfoque de VETTA: Aprendizaje por Clustering y Búsqueda

El enfoque adoptado en VETTA es una solución intermedia, pragmática y potente, que se basa en el aprendizaje a partir de ejemplos. En lugar de intentar generar rutas desde cero, el sistema aprende los "patrones" de problemas existentes creados por humanos. El módulo de creación de búlders (vetta_boulder_creation) implementa esta lógica a través de la clase BoulderPatternLearner, basándose en algoritmos clásicos de aprendizaje automático:

1. **Aprendizaje de patrones con K-Means**: El sistema se entrena a partir de un con-

junto de datos de problemas de búlder previamente definidos y etiquetados por dificultad. Utilizando el algoritmo K-Means, un método de clustering fundamental propuesto originalmente en [6], el BoulderPatternLearner agrupa estos problemas en clústeres. Cada clúster representa un arquetipo de problema con características similares (distribución espacial de las presas, longitud, tipos de agarre predominantes, etc.) para un grado de dificultad concreto.

2. **Generación por búsqueda de vecinos cercanos**: Cuando un usuario solicita un nuevo problema de una dificultad determinada, el sistema primero identifica el clúster correspondiente a esa dificultad. Luego, utilizando un algoritmo de búsqueda como Nearest Neighbors (vecinos más cercanos), genera una nueva secuencia de presas que sea coherente con los problemas existentes en dicho clúster. En esencia, el sistema navega por el 'espacio de problemas' que ha aprendido, encontrando los movimientos y secuencias más plausibles basándose en los ejemplos proporcionados por humanos.

Esta metodología, siendo *data-driven*, permite al sistema capturar la esencia de lo que hace que un problema sea bueno sin la complejidad de los modelos generativos profundos, representando una solución robusta y adecuada para el proyecto.

2.4. Arquitectura e Integración del Sistema

La integración de los diferentes componentes (interfaz de usuario, visión por computador, generación de rutas y proyección) es un pilar fundamental del proyecto. Para lograr una solución robusta, modular y de buen rendimiento, se ha optado por una arquitectura cliente-servidor, un estándar en el desarrollo de aplicaciones móviles modernas que requieren una capacidad de cómputo elevada.

- **Cliente (Frontend)**: Se materializa en una aplicación nativa para Android, desarrollada íntegramente en lenguaje Kotlin. Su responsabilidad es gestionar toda la interfaz de usuario y la interacción con el escalador, delegando las tareas pesadas al servidor.
- **Servidor (Backend)**: Se ha implementado un servidor en Python utilizando el micro-framework **Flask** [7]. Este servidor expone una Interfaz de Programación de Aplicaciones (Application Programming Interface) (API) Transferencia

Figura 2.3. Logo de Kotlin.

de Estado Representacional (Representational State Transfer) (REST) que es consumida por la aplicación móvil. Su única responsabilidad es gestionar las tareas computacionalmente intensivas: la inferencia de los modelos de visión y la ejecución de los algoritmos de generación de rutas.

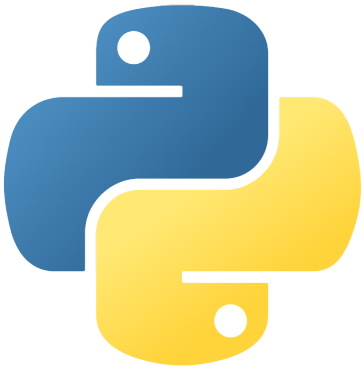


Figura 2.4. Logo de Python.

Esta arquitectura desacoplada es fundamental. Permite que la aplicación móvil se mantenga ligera y fluida, ya que no ejecuta los costosos cálculos de IA. Además, aporta una gran modularidad: si en el futuro se desarrolla un modelo de IA mejorado, solo será necesario actualizar el backend, sin que la aplicación cliente se vea afectada.

En resumen, el estado de la cuestión revela que, si bien existen soluciones de escalada digital, estas se basan mayoritariamente en hardware costoso y rígido o en sistemas de proyección orientados al juego. VETTA se posiciona como una solución innovadora al sintetizar tecnologías punteras de visión por computador (Mask R-CNN) con un enfoque pragmático y efectivo de aprendizaje automático para la generación de rutas, todo ello orquestado en una arquitectura cliente-servidor robusta y flexible. El proyecto apor-

ta una solución de bajo coste que pone el foco en la creación de problemas de búlder de calidad para el entrenamiento, democratizando el acceso a la escalada aumentada.

3. Metodología

El desarrollo del sistema VETTA, debido a su naturaleza compleja que combina desarrollo de aplicaciones móviles, servicios de backend e investigación en inteligencia artificial, se ha abordado siguiendo una metodología iterativa e incremental. Este enfoque ha sido fundamental para gestionar los riesgos y la complejidad del proyecto, permitiendo construir y validar el sistema por partes. En lugar de intentar desarrollar todos los componentes en paralelo, se ha optado por un proceso secuencial donde cada fase se apoya en una base funcional ya establecida por la anterior.

Este capítulo detallará el proceso de diseño y desarrollo a lo largo de tres iteraciones principales, cada una con un objetivo macro bien definido:

- **Primera Iteración: La Fundación (Aplicación y Datos).** El objetivo inicial fue construir el esqueleto del sistema: una aplicación móvil nativa para Android completamente funcional y una infraestructura de persistencia de datos robusta en la nube. Esta fase se centró en la experiencia de usuario, la gestión de datos y la autenticación, culminando en un producto mínimo viable (MVP) que, aunque carente de las funcionalidades de IA, es usable y sirve de base sólida para las siguientes etapas.
- **Segunda Iteración: La Percepción (Detección con IA).** Con la aplicación base ya operativa, esta fase se centró en dotar al sistema de 'ojos'. El objetivo fue desarrollar y entrenar los modelos de visión por computador necesarios para que VETTA pudiera analizar una imagen de un muro de escalada y extraer de ella toda la información relevante: la ubicación, la forma y las características de cada presa. Esta iteración es la que conecta el mundo físico con el digital.
- **Tercera Iteración: La Creatividad (Generación de Rutas).** La última fase del proyecto consistió en darle al sistema un 'cerebro' creativo. Apoyándose en la información extraída en la iteración anterior, el objetivo fue desarrollar un modelo de inteligencia artificial capaz de aprender los patrones que definen un buen problema de búlder y, a partir de ellos, generar nuevas rutas coherentes y ajustadas a una dificultad específica. Esta es la iteración que completa la propuesta de valor principal del proyecto.

A continuación, se describirá en detalle el proceso de implementación, las herramientas seleccionadas y las decisiones de diseño tomadas en cada una de estas tres iteraciones.

3.1. Primera Iteración: Aplicación Base y Persistencia de Datos

El objetivo de esta primera fase fue construir el esqueleto funcional de la aplicación VETTA, estableciendo una base sólida sobre la cual integrar las funcionalidades de inteligencia artificial en iteraciones posteriores. El resultado al final de esta iteración es una aplicación Android robusta donde los usuarios pueden registrarse, autenticarse, explorar una lista de rocódromos y visualizar los problemas de búlder asociados a cada uno, con toda la información persistida de forma segura en una base de datos en la nube.

3.1.1. Requisitos de la Primera Iteración

Para guiar el desarrollo de esta fase inicial, se definieron los siguientes requisitos funcionales y no funcionales.

Requisitos Funcionales (RF)

Los requisitos funcionales describen las capacidades que el sistema debe ofrecer al usuario:

- **RF-1: Gestión de Usuarios.** El sistema debe permitir a un nuevo usuario registrarse en la aplicación proporcionando un correo electrónico y una contraseña. Asimismo, debe permitir a los usuarios ya registrados iniciar sesión con sus credenciales.
- **RF-2: Cierre de Sesión.** El usuario debe tener la opción de cerrar su sesión de forma segura desde la aplicación.

- **RF-3: Visualización de Gimnasios.** Tras iniciar sesión, la aplicación debe mostrar al usuario una lista de todos los rocódromos o gimnasios disponibles en la plataforma.
- **RF-4: Visualización de Problemas de Búlder.** Al seleccionar un gimnasio de la lista, la aplicación debe mostrar todos los problemas de búlder asociados a ese gimnasio.
- **RF-5: Visualización Personalizada.** El sistema debe ser capaz de mostrar al usuario una vista filtrada que contenga únicamente los problemas de búlder que él mismo ha creado.

Requisitos No Funcionales (RNF)

Los requisitos no funcionales describen las cualidades y restricciones del sistema:

- **RNF-1: Usabilidad.** La interfaz de usuario debe ser intuitiva y seguir los patrones de diseño estándar de Material Design para Android, garantizando una curva de aprendizaje mínima para el usuario.
- **RNF-2: Rendimiento.** La aplicación debe ser fluida y responsiva. Las operaciones que requieran acceso a la red, como la carga de listas de gimnasios o búlders, deben realizarse de forma asíncrona para no bloquear el hilo principal de la interfaz.
- **RNF-3: Seguridad.** Las credenciales de los usuarios deben ser gestionadas de forma segura. No se almacenarán contraseñas en texto plano y se utilizarán los mecanismos de un proveedor de autenticación fiable.
- **RNF-4: Escalabilidad.** La infraestructura del backend (la base de datos) debe ser capaz de escalar para soportar un número creciente de usuarios, gimnasios y problemas de búlder sin degradar el rendimiento.
- **RNF-5: Compatibilidad.** La aplicación debe ser compatible con un amplio rango de dispositivos Android, soportando como mínimo desde la versión de API 26 (Android 8.0 Oreo) en adelante.

3.1.2. Desarrollo de la Aplicación Cliente (Android)

Se desarrolló una aplicación nativa para Android, priorizando un buen rendimiento, una experiencia de usuario fluida y una arquitectura escalable.

Entorno y Arquitectura La aplicación se construyó utilizando herramientas y patrones modernos del ecosistema Android:

- **Lenguaje**: Se eligió **Kotlin** como único lenguaje de programación, en línea con las recomendaciones de Google, por su seguridad en el manejo de nulos, su concisión y su total interoperabilidad con el ecosistema de Java y Android.
- **Arquitectura**: Se implementó una arquitectura de una única actividad (**Single-Activity Architecture**) con MainActivity.kt como punto de entrada principal tras la autenticación. La navegación entre las diferentes pantallas se gestiona mediante el componente **Navigation Component** de Android Jetpack, utilizando Fragments para cada vista. Este enfoque centraliza la lógica de navegación, reduce el código repetitivo y facilita un manejo del estado más eficiente.
- **Gestión de Estado**: Para comunicar datos entre fragmentos y mantener un estado consistente a lo largo del ciclo de vida de la aplicación (p. ej., mantener la información del gimnasio seleccionado), se utiliza un SharedViewModel. Este es un componente de Android Jetpack que permite que diferentes fragmentos compartan y observen los mismos datos sin necesidad de crear complejas interfaces de comunicación entre ellos.

Componentes de la Interfaz de Usuario (UI) En esta iteración se desarrollaron las siguientes pantallas (fragmentos) clave, cuyos diseños están definidos en ficheros XML en el directorio res/layout. La Figura 3.1 muestra una vista general de las pantallas principales.

- **Autenticación (activity_login.xml)**: Aunque la aplicación arranca en la pantalla principal, la gestión de la sesión se centraliza en la pantalla de autenticación. A esta se accede desde la sección de perfil cuando un usuario no autenticado desea registrarse o iniciar sesión. La pantalla presenta un diseño limpio con campos para correo y contraseña. Su lógica, implementada en LoginActivity.kt, se comunica

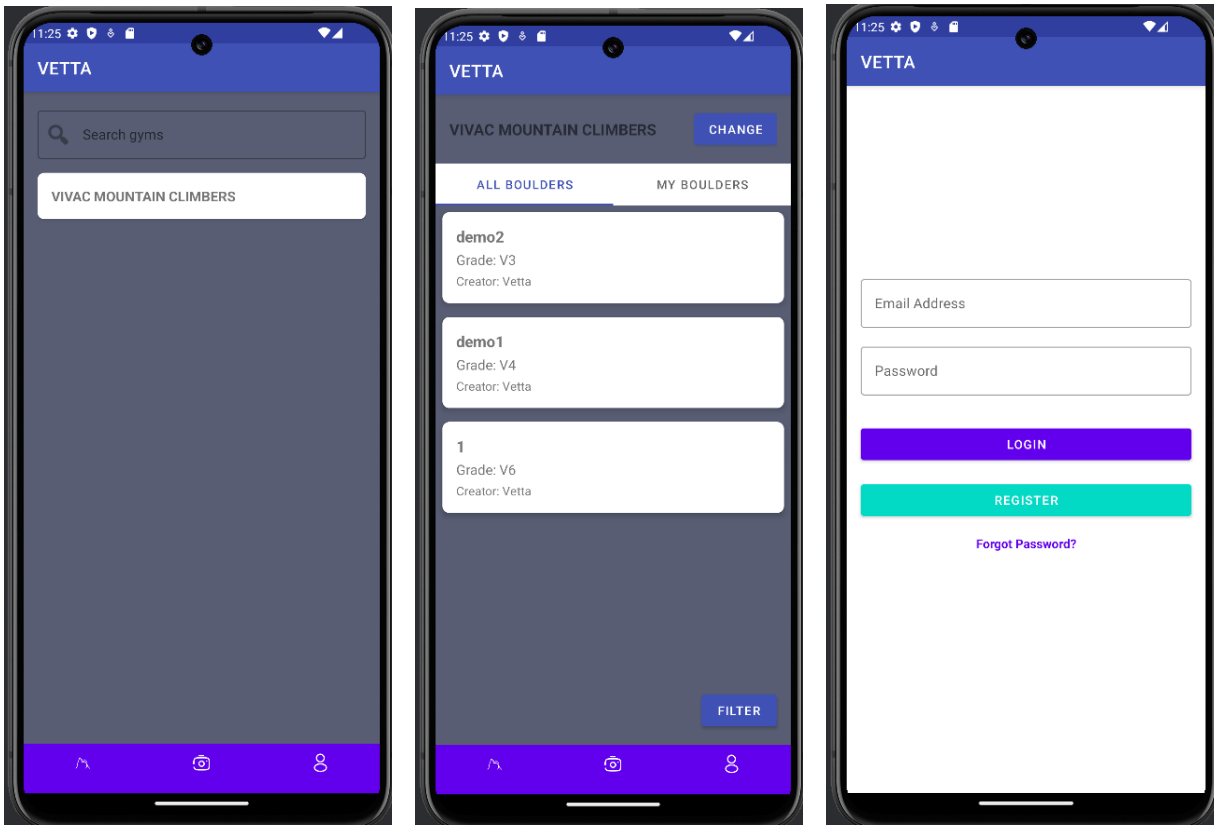
directamente con el servicio de Firebase Authentication para validar credenciales o crear nuevos usuarios de forma segura.

- **Navegación Principal (activity_main.xml)**: Tras la autenticación, el usuario accede a la estructura principal de la aplicación. Esta contiene un BottomNavigationView, un componente estándar de Material Design, que permite al usuario alternar de forma rápida e intuitiva entre las tres secciones principales: Rutas (HomeFragment), Cámara (un marcador de posición en esta iteración) y Perfil (ProfileFragment).

- **Pantalla de Inicio (HomeFragment.kt)**: Este fragmento, ilustrado en la Figura 3.1a, es la pantalla principal que ve el usuario. Su función es obtener y mostrar una lista de gimnasios desde Cloud Firestore. Para visualizar esta lista de forma eficiente, especialmente con un gran número de elementos, se utiliza un RecyclerView. Cada elemento de la lista se renderiza usando un layout personalizado, item_gym.xml, a través de un adaptador (GymAdapter), mostrando el nombre y la ubicación de cada rocódromo.

- **Visualización de Búlders**: Una vez el usuario selecciona un gimnasio, la aplicación navega a una nueva vista que contiene dos pestañas para facilitar la exploración de los problemas. Esta pantalla (Figura 3.1b) es fundamental para la interacción del usuario con el contenido.

  - **Todos los Búlders (AllBouldersFragment.kt)**: Esta pestaña muestra todos los problemas de búlder públicos asociados al gimnasio seleccionado. La implementación es similar a la del HomeFragment, utilizando un RecyclerView y un BoulderAdapter para mostrar la información de cada problema (nombre, grado, creador), definida en item_boulder.xml.
  - **Mis Búlders (MyBouldersFragment.kt)**: Esta pestaña ofrece una vista personalizada, permitiendo al usuario ver una lista filtrada que contiene únicamente los problemas que él mismo ha creado. Reutiliza la misma estructura visual que la pestaña anterior, pero realiza una consulta a Firestore filtrando los búlders por el ID del usuario actualmente autenticado, demostrando la capacidad del sistema para ofrecer vistas personalizadas.



(a) Pantalla de inicio con la lista de gimnasios.

(b) Lista de búlders disponibles en un gimnasio.

(c) Pantalla de perfil de usuario.

Figura 3.1. Capturas de pantalla de los componentes principales de la interfaz de usuario de VETTA.

3.1.3. Infræstructura de Backend (Firebase)

Para la persistencia de datos y la gestión de usuarios, se eligió la plataforma Firebase de Google. Esta elección se justifica por ser una solución de tipo *Backend-as-a-Service* (BaaS) que permite un desarrollo rápido y seguro de funcionalidades comunes, es altamente escalable y se integra de forma nativa con el ecosistema de Android, lo que reduce significativamente la complejidad del desarrollo del backend.

Firebase Authentication Se utilizó este servicio para gestionar el ciclo de vida completo de la autenticación de usuarios. La implementación en LoginActivity.kt hace uso de los métodos createUserWithEmailAndPassword y signInWithEmailAndPassword del Kit de Desarrollo de Software (Software Development Kit) (SDK) de Firebase. Este enfoque externaliza la gestión de contraseñas y sesiones, garantizando un manejo seguro

de las credenciales, el hashing de contraseñas y la persistencia de la sesión del usuario en el dispositivo sin necesidad de construir y mantener un sistema de autenticación propio desde cero.

Cloud Firestore: Base de Datos NoSQL Toda la información de la aplicación, como los datos de los gimnasios, los problemas de búlder y los perfiles de usuario, se almacena en Cloud Firestore, una base de datos de documentos NoSQL flexible y en tiempo real. Se diseñó una estructura de datos desnormalizada, optimizada para las consultas que la aplicación necesita realizar con mayor frecuencia. La Figura 3.2 ilustra este modelo de datos.

- **Colección gyms**: Es la colección de nivel superior donde cada documento representa un rocódromo. El modelo de datos para cada gimnasio, definido en la clase Gym.kt, incluye campos como id, name, latitude y longitude, permitiendo su futura localización en un mapa.
- **Subcolección boulders**: Para una organización eficiente y para garantizar que las consultas sean rápidas, los problemas de búlder no se guardan en una colección principal, sino como una subcolección anidada dentro de cada documento de gyms. De esta forma, cada búlder está directamente ligado a su gimnasio correspondiente. Las consultas para obtener los búlders de un gimnasio son muy eficientes, como se puede ver en el código de AllBouldersFragment.kt:

```
FirebaseFirestore.getInstance()
    .collection("gyms")
    .document(gymId)
    .collection("boulders")
    .get()
```

- **Colección users**: Se almacena la información del perfil de los usuarios (nombre, email, etc.) en una colección principal users, donde el ID de cada documento se corresponde con el 'uid' que proporciona Firebase Authentication. Esto permite vincular fácilmente los datos de un usuario con su sesión y, en futuras iteraciones, con los búlders que haya creado.

Al final de esta primera iteración, se dispone de una aplicación Android completamente funcional con un backend en la nube que permite a los usuarios registrarse, explorar gimnasios y visualizar problemas, sentando una base sólida para las siguientes fases del proyecto.

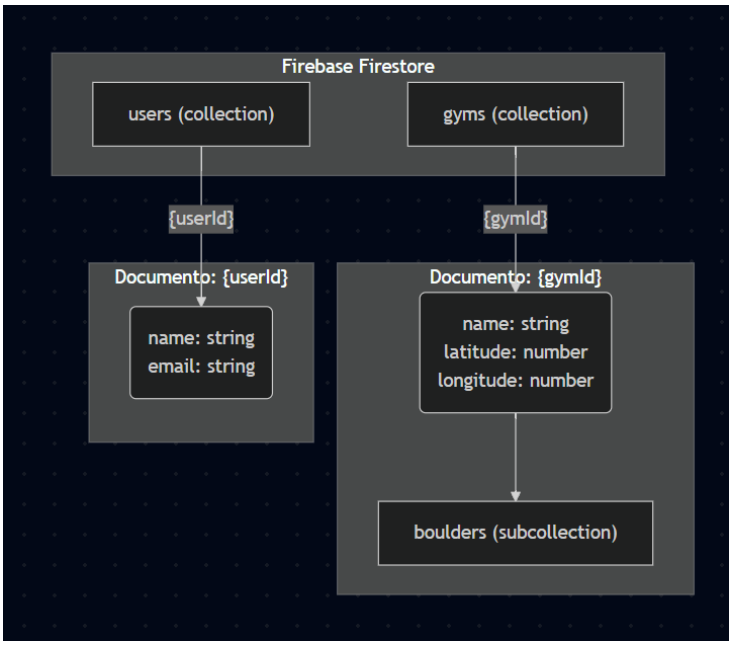


Figura 3.2. Diagrama conceptual de la estructura de datos en Cloud Firestore. La lista de búlders (como la que se ve en esta captura) se obtiene de la subcolección anidada en cada gimnasio.

3.2. Segunda Iteración: Detección de Presas con IA y Servicio Backend

Con la aplicación base ya funcional, el objetivo de esta segunda iteración fue desarrollar el componente de inteligencia artificial más crítico del sistema: **la detección y análisis de presas**. El resultado de esta fase es un servicio de *backend* capaz de recibir una imagen de un muro de escalada y devolver la ubicación, forma y tipo de cada presa. Esta funcionalidad se integra en la aplicación Android, permitiendo al usuario visualizar las presas detectadas sobre la foto que ha tomado.

3.2.1. Requisitos de la Segunda Iteración

Para guiar el desarrollo de esta fase de inteligencia artificial, se definieron los siguientes requisitos:

Requisitos Funcionales (RF)

- **RF-6: Creación de un Endpoint de API.** El backend deberá exponer un API REST, concretamente un *endpoint* (/predict), que sea capaz de recibir una imagen enviada desde el cliente móvil.
- **RF-7: Detección y Segmentación de Presas.** Al recibir una imagen, el sistema debe procesarla utilizando el modelo de visión por computador para identificar todas las presas presentes y generar una Máscara de Segmentación precisa para cada una.
- **RF-8: Clasificación de Tipos de Agarre.** Para cada presa segmentada, el sistema debe analizar su forma y características para clasificarla en una de las categorías de agarre predefinidas (p. ej., regleta, cazo, romo, etc.).
- **RF-9: Respuesta Estructurada.** El servidor debe devolver una respuesta en formato JSON que contenga una lista de todas las presas detectadas, incluyendo para cada una su caja delimitadora, su máscara de segmentación y su tipo de agarre clasificado.

Requisitos No Funcionales (RNF)

- **RNF-6: Precisión del Modelo.** El modelo de detección y segmentación (Mask R-CNN) deberá alcanzar un mAP de segmentación superior al 40
- **RNF-7: Rendimiento del Servicio.** El tiempo total de procesamiento en el servidor para una petición (desde que recibe la imagen hasta que devuelve el JSON) no deberá superar los 10 segundos para garantizar una experiencia de usuario fluida.
- **RNF-8: Modularidad del Código.** El código del backend debe estar bien estructurado, separando la lógica del servidor (Flask), la del modelo de detección (Detectron2) y la del clasificador de agarres en módulos distintos para facilitar su mantenimiento y futuras mejoras.
- **RNF-9: Integración Cliente-Servidor.** La aplicación Android debe ser capaz de comunicarse correctamente con el *endpoint* del servidor, enviando la imagen en el formato adecuado y parseando la respuesta JSON sin errores para su visualización.

3.2.2. Desarrollo del Backend de Inteligencia Artificial

El *backend* se desarrolló como un microservicio en **Python**, el lenguaje estándar para aplicaciones de IA, utilizando el *framework* **Flask** [7] para exponer su funcionalidad a través de una API REST. Este servicio alberga los modelos de visión por computador, que se dividen en dos tareas principales: la detección de la ubicación de las presas y la clasificación de su tipo.

Módulo de Detección de Presas (vetta-holds-detection)

El corazón de esta iteración es el modelo de visión capaz de realizar una **segmentación de instancias** sobre la imagen del muro. Esta tarea va más allá de la simple detección, ya que no solo localiza las presas, sino que delimita su contorno exacto, una característica esencial para el proyecto.

Arquitectura y Configuración del Modelo Para esta tarea se eligió la arquitectura **R-CNN con una red troncal ResNet-50 y Red de Pirámide de Características (Feature Pyramid Network) (FPN)**, una de las arquitecturas de referencia para la segmentación de instancia por su equilibrio entre precisión y eficiencia [4]. La implementación se realizó a través de la biblioteca **Detectron2** [5].

Toda la configuración del modelo y del entrenamiento se centraliza en el fichero config.py, lo que permite una gestión y modificación sencilla de los hiperparámetros. La función create_detectron_model en model.py se encarga de aplicar esta configuración, que incluye:

- **Arquitectura Base**: Se carga la configuración por defecto `COCO-InstanceSegmentation/mask\_rcnn\_R\_50\_FPN\_3x.yaml` desde el Model Zoo de Detectron2, que ya incluye los pesos pre-entrenados en el conjunto de datos COCO.
- **Ajuste de Clases**: Se modifica la cabecera del modelo (`MODEL.ROI\_HEADS.NUM\_CLASSES`) para que coincida con el número de clases definidas en el proyecto (big_hold, medium_hold, small_hold).
- **Hiperparámetros de Entrenamiento**: Se definen parámetros clave como la tasa de aprendizaje inicial (base_lr), el tamaño del lote por GPU (ims_per_batch) y el nú-

mero máximo de iteraciones (max_iter) para controlar la duración y la intensidad del entrenamiento.

Proceso de Entrenamiento El desarrollo del modelo siguió un flujo de trabajo de aprendizaje supervisado, orquestado por la función train en el *script* detectron2_script.py:

1. **Recopilación y Anotación de Datos**: Se creó un conjunto de datos personalizado con imágenes de diversos muros de escalada. Utilizando herramientas de anotación, se etiquetó manualmente cada presa en cada imagen, generando las máscaras de segmentación y las cajas delimitadoras necesarias. Estos datos se exportaron en formato COCO, un estándar para tareas de detección de objetos.

2. **Registro de los Datos**: Antes de poder ser utilizados, los conjuntos de datos de entrenamiento, validación y prueba deben ser registrados en el catálogo de Detectron2. La función register_coco_datasets del fichero data_processing.py se encarga de esta tarea, asociando un nombre a cada conjunto de datos y especificando la ruta al fichero de anotaciones JSON y al directorio de imágenes.

3. **Aumento de Datos (Data Augmentation)**: Para mejorar la capacidad de generalización del modelo y evitar el sobreajuste, se aplicó un conjunto de transformaciones aleatorias a las imágenes de entrenamiento en tiempo de ejecución. La función get_coco_mapper construye un pipeline de aumentos que incluye cambios en el brillo, el contraste, la saturación y rotaciones aleatorias, simulando así diferentes condiciones de iluminación y ángulos de cámara que el modelo podría encontrar en un entorno real.

4. **Entrenamiento y Evaluación**: El proceso de entrenamiento es manejado por la clase CustomTrainer, que extiende el DefaultTrainer de Detectron2. Esta clase personalizada permite sobreescribir métodos para, por ejemplo, cargar el pipeline de aumento de datos o configurar el evaluador. Durante el entrenamiento, se utiliza la técnica de *transfer learning*, partiendo de los pesos pre-entrenados en COCO. Periódicamente (definido por TEST.EVAL_PERIOD), el entrenamiento se pausa y se ejecuta una evaluación sobre el conjunto de validación. La clase BestModelHook se encarga de monitorizar el mAP de segmentación y guardar el modelo con mejor rendimiento hasta el momento, asegurando que el resultado final sea el más óptimo.

5. **Inferencia**: Una vez entrenado, el modelo final (o el mejor modelo guardado) se carga para realizar la inferencia sobre nuevas imágenes. El *script* inference.py con-

tiene la lógica para cargar el modelo y su configuración, y procesar una imagen para devolver la lista de presas detectadas con sus máscaras y cajas delimitadoras.

Clasificación de Tipos de Agarre

Una vez que el modelo de segmentación ha identificado la ubicación y forma de una Presa (o Agarre), el siguiente paso es dotar al sistema de un entendimiento más profundo: clasificar su tipo de agarre (p. ej., crimp, sloper, jug, etc.). Esta información es crucial para la posterior generación de problemas, ya que la dificultad de una ruta no solo depende de la distancia entre presas, sino también del tipo de agarre que ofrecen. Esta tarea se aborda mediante un proceso clásico de aprendizaje automático supervisado, dividido en dos fases: extracción de características y clasificación.

Extracción de Características Geométricas En lugar de alimentar directamente la imagen de la presa a una red neuronal compleja, se optó por un enfoque más interpretable basado en la extracción de características. A partir de la Máscara de Segmentación binaria de cada presa, el módulo HoldVisionAnalyzer calcula un vector de características numéricas que describen su morfología de forma cuantitativa. La lógica de estas características se encuentra definida en la clase HoldFeatures. Las más relevantes son:

- **Circularidad**: Mide cuán parecida es la forma a un círculo perfecto. Es especialmente útil para identificar presas de tipo Romo (Sloper) o Bolsillo (Pocket) que tienden a ser muy redondeadas. Se calcula como:

$$C = \frac{4\pi \times \text{Área}}{\text{Perímetro}^2} \tag{3.1}$$

  Un valor cercano a 1 indica una forma muy circular, mientras que valores bajos sugieren formas más alargadas o complejas.

- **Convexidad**: Compara el área de la presa con el área de su envoltura convexa (el polígono convexo más pequeño que la contiene). Es un buen indicador de concavidades, útil para distinguir agarres positivos como Cazo (Jug)s o Pinza (Pinch)s de otros más planos.

$$V = \frac{\text{Área de la Presa}}{\text{Área de la Envoltura Convexa}} \tag{3.2}$$

  Un valor cercano a 1 significa que la forma es mayormente convexa, sin grandes entrantes.

- **Relación de Aspecto (Aspect Ratio)**: Es el cociente entre la anchura y la altura de la caja delimitadora de la presa. Esta simple métrica ayuda a diferenciar entre presas alargadas horizontalmente (típico de Regleta (Crimp)s) o verticalmente.

$$AR = \frac{\text{Anchura}}{\text{Altura}} \tag{3.3}$$

Clasificación Supervisada El vector de características extraído se utiliza como entrada para un modelo de clasificación tradicional. En este proyecto, se implementó un ensamblado de clasificadores (*ensemble classifier*) que combina una Máquina de Vectores de Soporte (SVM), un Bosque Aleatorio (Random Forest) y un Gradient Boosting, utilizando una estrategia de votación para mejorar la robustez. El modelo se entrena con el *script* train_vision_model.py, utilizando un conjunto de datos de presas previamente etiquetadas a mano con su tipo de agarre. De esta forma, el modelo aprende a asociar patrones en las características geométricas (p. ej., "baja circularidad y alta relación de aspecto") con un tipo de agarre específico (p. ej., Regleta (Crimp)).

3.2.3. Implementación del Servidor y la API

Para que la aplicación móvil pueda hacer uso de los modelos de IA, se desarrolló un servidor web con Flask [7]. Este servidor actúa como un puente, exponiendo la compleja funcionalidad de los modelos a través de una API REST sencilla.

Endpoint de Detección /predict Se definió un único *endpoint* /predict que recibe peticiones POST. El cuerpo de la petición debe contener la imagen del muro enviada desde la aplicación móvil. Al recibir la petición, el servidor ejecuta la siguiente secuencia de operaciones orquestada en server.py:

1. Guarda la imagen recibida en una ubicación temporal.
2. Invoca al *script* de inferencia de Mask R-CNN (inference.py) para obtener las máscaras de segmentación de todas las presas en la imagen.
3. Itera sobre cada una de las presas detectadas. Para cada una, invoca al módulo HoldVisionAnalyzer para extraer su vector de características y predecir su tipo de agarre.

4. Ensambla todos los resultados en un único objeto JSON. Este objeto contiene una lista de presas, donde cada presa incluye su máscara de segmentación codificada, su caja delimitadora y su tipo de agarre clasificado.
5. Devuelve la respuesta JSON al cliente móvil.

3.2.4. Integración con la Aplicación Android

Finalmente, la funcionalidad del *backend* se integró en la aplicación móvil, completando el flujo de esta segunda iteración.

Llamada Asíncrona a la API En la aplicación, cuando el usuario toma una foto desde el CameraFragment, se invoca a la función asíncrona detectHoldsInImage. Esta función, declarada como suspend fun en Kotlin, se ejecuta en una corrutina para no bloquear el hilo principal de la interfaz de usuario. Se encarga de empaquetar la imagen en una petición Multipart y enviarla al *endpoint* /predict del servidor. Para que la comunicación con el servidor de desarrollo local funcione, se configuró el fichero network_security_config.xml para permitir el tráfico en texto plano hacia la dirección IP correspondiente (10.0.2.2).

Visualización de Resultados Tras recibir la respuesta JSON del servidor, la aplicación navega a una nueva pantalla, HoldSelectionFragment. Este fragmento es el responsable de:

1. Mostrar la imagen original capturada por el usuario como fondo.
2. Decodificar las máscaras de segmentación recibidas para cada presa.
3. Renderizar cada máscara como una capa de color semitransparente sobre la imagen original. Esto permite al usuario ver de forma clara y precisa qué presas han sido detectadas por el sistema y dónde están ubicadas, proporcionando una realimentación visual inmediata.

Al finalizar esta segunda iteración, VETTA pasa de ser una aplicación de visualización de datos a una herramienta interactiva con capacidades de visión por computador, permitiendo digitalizar un muro de escalada con una simple foto y sentando las bases para la generación de rutas.

3.3. Tercera Iteración: Generación de Problemas de Búlder con IA

La última iteración del proyecto se centra en dotar al sistema de una capacidad creativa: la generación automática de problemas de Búlder. El objetivo es que el usuario no solo pueda crear sus propias rutas manualmente, sino que también pueda solicitar a la IA que diseñe un problema nuevo con un grado de dificultad específico. Esta fase culmina la funcionalidad principal del proyecto VETTA, entregando un sistema completo de escalada aumentada.

3.3.1. Requisitos de la Tercera Iteración

Para guiar el desarrollo del componente más complejo del sistema, se definieron los siguientes requisitos:

Requisitos Funcionales (RF)

- **RF-10: Creación de un Endpoint de Generación.** El backend deberá exponer un nuevo *endpoint* en su API (/create-problem) que acepte una lista de las presas disponibles en el muro y un parámetro de dificultad (p. ej., "V4").
- **RF-11: Generación Coherente de Rutas.** El sistema de IA debe ser capaz de generar una secuencia de presas que constituya una ruta de escalada lógica. Los movimientos deben ser escalables y seguir un patrón ascendente general.
- **RF-12: Ajuste a la Dificultad.** La ruta generada debe corresponderse con el grado de dificultad solicitado por el usuario. El modelo debe ser capaz de seleccionar presas y proponer distancias acordes al nivel de la ruta.
- **RF-13: Integración en la Interfaz de Usuario.** La aplicación Android debe permitir al usuario solicitar una ruta a la IA, introduciendo los parámetros necesarios (como la dificultad) a través de un diálogo en la interfaz.
- **RF-14: Visualización de la Ruta Generada.** Tras recibir la respuesta del servidor, la aplicación debe ser capaz de resaltar visualmente las presas que componen la

ruta generada sobre la imagen del muro.

Requisitos No Funcionales (RNF)

- **RNF-10: Calidad Percibida.** Las rutas generadas deben ser percibidas como 'buenas' o 'interesantes' por escaladores reales. Su calidad se medirá de forma cualitativa mediante una encuesta de satisfacción.
- **RNF-11: Variedad.** El sistema debe ser capaz de generar diferentes rutas para una misma petición, evitando producir siempre el mismo resultado para un mismo muro y grado de dificultad.
- **RNF-12: Rendimiento de la Generación.** El tiempo de ejecución del algoritmo de generación en el servidor debe ser inferior a 2 segundos para no impactar negativamente en la experiencia de usuario.

3.3.2. Módulo de Creación de Búlders (vetta-boulder-creation)

El desarrollo de este módulo se basó en un enfoque de aprendizaje automático no supervisado para descubrir y replicar patrones a partir de ejemplos creados por humanos. La lógica se encapsula principalmente en dos clases: BoulderPatternLearner, que es el cerebro que aprende los patrones, y BoulderCreator, que es el orquestador que utiliza el modelo para construir un problema.

Creación del Conjunto de Datos de Problemas

La base para que la IA aprenda a crear buenas rutas es disponer de un conjunto de datos de ejemplos de alta calidad. Para ello, se utilizó una herramienta auxiliar, el *script* boulder_manual_creator.py. Este permite a un usuario experto cargar la salida del módulo de detección de presas (de la Iteración 2) y, de forma interactiva, seleccionar secuencias de presas para definir un problema de Búlder. Cada problema se guarda junto con una etiqueta que indica su grado de dificultad (p. ej., V3, V4, V5) en un fichero JSON. Este proceso, aunque manual, es fundamental, ya que la calidad y coherencia de las rutas generadas por la IA dependerán directamente de la calidad de estos ejemplos, los cuales son cargados por el sistema mediante la función load_existing_problems.

Modelo de Generación de Rutas: BoulderPatternLearner

Se implementó una clase, BoulderPatternLearner, que encapsula la lógica de aprendizaje y generación. Su metodología se divide en dos fases:

Fase de Entrenamiento (Clustering por Arquetipos) En esta fase, el modelo debe clasificar de forma no supervisada el conjunto de problemas de búlder de ejemplo para poder aprender los 'arquetipos' o estilos correspondientes a cada nivel de dificultad. Para esta tarea de agrupamiento se utiliza el algoritmo K-Means [6], cuya implementación se realiza a través de la biblioteca scikit-learn.

El primer paso es convertir cada problema de Búlder en un vector numérico mathbfx que represente sus características principales. Este vector, generado por el método _featurize_problem, no solo contiene las coordenadas normalizadas de sus presas, sino también estadísticas agregadas que describen el problema, como el número total de presas, la distancia promedio entre ellas o la distribución de tipos de agarre.

El objetivo de K-Means es particionar el conjunto de todos los problemas en k clústeres, minimizando la suma de las distancias euclidianas al cuadrado desde cada problema x hasta el centroide μi de su clúster asignado Si. Formalmente, el algoritmo minimiza la inercia total dentro de los clústeres:

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 \tag{3.4}$$

El *script* train_boulder_ml.py orquesta este proceso de entrenamiento, que no solo agrupa los problemas, sino que también aprende los patrones estadísticos de cada clúster (transiciones entre agarres, distribuciones espaciales, etc.) y guarda el modelo entrenado para su uso posterior.

Fase de Generación (Sugerencia de Presas) Una vez entrenado el modelo, para crear un nuevo problema, el sistema no genera una ruta completa de una vez, sino que lo hace de forma secuencial, presa por presa. Este proceso es gestionado por el método create_boulder_problem dentro de la clase BoulderCreator. Dicho método invoca repetidamente a suggest_next_hold del BoulderPatternLearner, el cual puntúa cada presa disponible en función de lo bien que encaja en la secuencia actual. La puntuación de cada presa candidata es una media ponderada de tres factores, como se muestra en el Listado 3.1:

Listado 3.1. Lógica de puntuación para la sugerencia de la siguiente presa.

```
vetta-boulder-creation/src/ml/boulder\_ml.py
def calculate\_hold\_score(self, hold, last\_hold, cluster\_id,
    section\_goal):
# ... (cálculo de la sección espacial) ...

# 1. Puntuación de transición de agarre (40%)
transition\_score = self.calculate\_transition\_score(...)

# 2. Puntuación de patrón de agarre (25%)
grip\_score = self.calculate\_grip\_score(...)

# 3. Puntuación de patrón espacial y personalización (35%)
spatial\_score = self.calculate\_spatial\_score(...)

score = (transition\_score * 0.40) + \
        (grip\_score * 0.25) + \
        (spatial\_score * 0.35)

return score
```

Como se ve en el código, se evalúa la probabilidad de transición entre el tipo de agarre de la última presa y la candidata, la idoneidad del tipo de agarre para esa sección del búlder (inicio, medio o final), y el ajuste espacial del movimiento (distancia y ángulo) a los patrones del clúster de dificultad correspondiente. De esta forma, se elige iterativamente la presa con la puntuación más alta hasta completar el problema.

Para enriquecer aún más la calidad de las rutas, el sistema introduce un factor de personalización ergonómica basado en la altura del escalador, un parámetro configurable en el modelo. Al calcular la puntuación espacial de un movimiento, no solo se considera su idoneidad respecto a los patrones del clúster de dificultad, sino que también se evalúa su ajuste al alcance óptimo y máximo de un escalador. El sistema penaliza los movimientos que serían excesivamente largos o inverosímiles y favorece aquellos cuya distancia se acerca al alcance ideal, utilizando una función gaussiana para ponderar la comodidad del movimiento. De esta manera, las rutas generadas no solo son coheren-

tes en términos de dificultad, sino que también proponen secuencias de movimientos más naturales y realistas.

3.3.3. Integración del Servicio y la Aplicación

Ampliación de la API del Servidor Para exponer esta nueva funcionalidad, se añadió un segundo *endpoint* al servidor Flask: /create-problem. Este *endpoint* espera una petición POST que contenga dos datos clave: la lista completa de presas disponibles en el muro (obtenida en la Iteración 2) y el grado de dificultad deseado por el usuario. El servidor invoca al BoulderCreator, que a su vez utiliza el BoulderPatternLearner, y devuelve la nueva ruta generada en formato JSON.

Actualización de la Interfaz de Usuario En la aplicación Android, la pantalla HoldSelectionFragment fue modificada para incluir un nuevo botón que permite al usuario solicitar una ruta a la IA. Al pulsar este botón, se muestra un diálogo (dialog_ai_boulder_params.xml) donde el usuario puede seleccionar el grado de dificultad. Una vez confirmado, la aplicación realiza la llamada al nuevo *endpoint* /create-problem, enviando la información necesaria. Al recibir la respuesta del servidor, la aplicación resalta las presas correspondientes a la ruta generada, mostrando el resultado final al usuario.

Con la finalización de esta tercera y última iteración, el proyecto VETTA alcanza su objetivo principal, ofreciendo un sistema completo e integrado que no solo digitaliza un entorno de escalada, sino que también actúa como un asistente creativo, capaz de generar nuevos y desafiantes problemas de Búlder de forma automática.

4. Desarrollo e Implementación

Este capítulo detalla las herramientas, lenguajes, librerías y entornos de desarrollo específicos que se han utilizado para materializar la metodología descrita en el capítulo anterior. El objetivo es proporcionar una visión técnica de la implementación de cada uno de los componentes principales del sistema VETTA: el backend de inteligencia artificial y la aplicación cliente para Android.

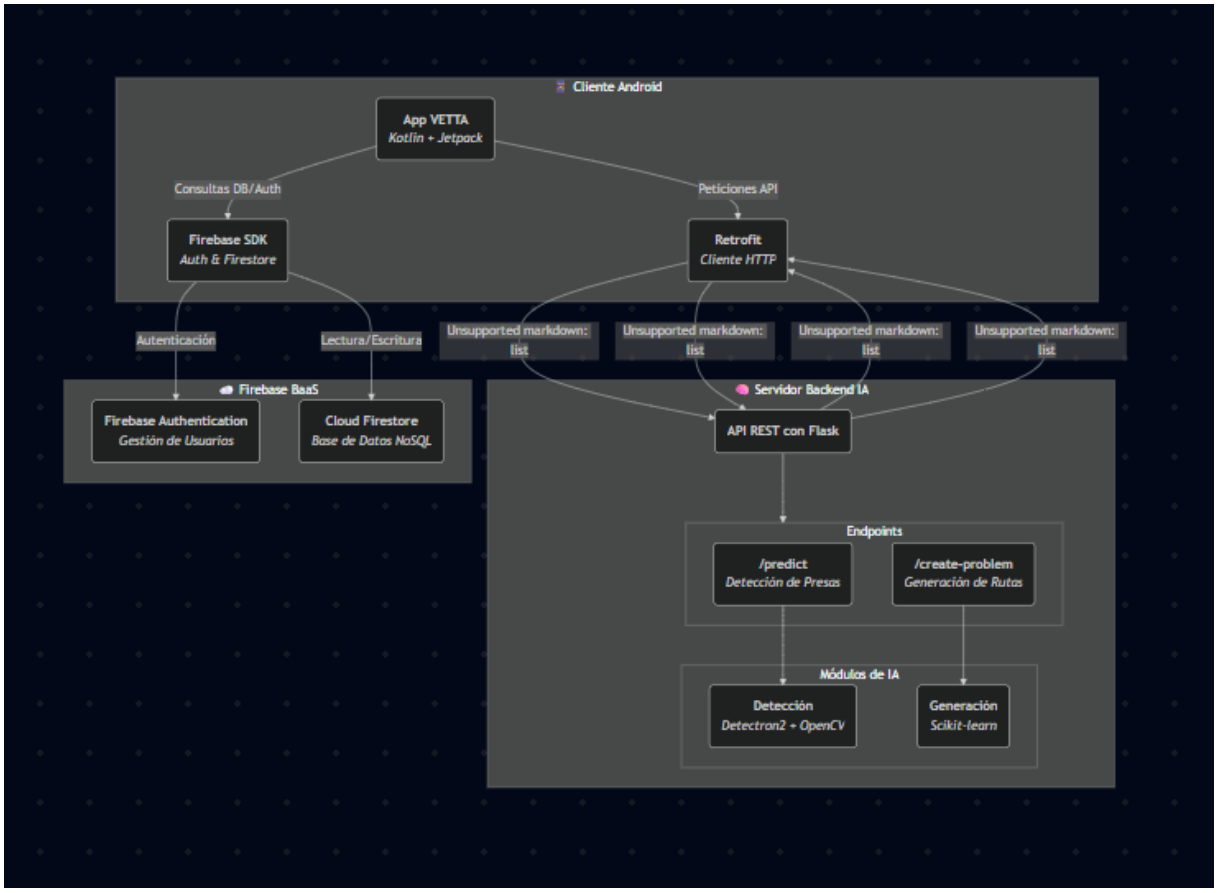


Figura 4.1. Diagrama de la arquitectura general del sistema VETTA, mostrando la interacción entre la aplicación cliente Android, los servicios de Firebase y el backend de IA a través de la API REST.

4.1. Entorno de Desarrollo del Backend

El servidor y los modelos de IA se desarrollaron en un entorno Python, gestionado mediante entornos virtuales para asegurar la reproducibilidad y evitar conflictos de dependencias.

- **Lenguaje de Programación**: Se utilizó **Python 3.9**, por su madurez, su extenso ecosistema de librerías para ciencia de datos y su facilidad de uso.
- **Framework del Servidor**: Para la creación de la API REST, se eligió **Flask** [7]. Su naturaleza de micro-framework lo hace ideal para este proyecto, ya que proporciona la flexibilidad necesaria para construir una API sencilla sin imponer una estructura rígida.
- **Librerías de Inteligencia Artificial**:
  - **PyTorch y Detectron2**: El corazón del sistema de visión. Se utilizó Detectron2 [5] sobre PyTorch para la implementación, entrenamiento y ejecución del modelo Mask R-CNN.
  - **Scikit-learn**: Se empleó para las tareas de aprendizaje automático más clásicas, como la implementación del algoritmo K-Means para el clustering de problemas y la creación de los modelos de clasificación para los tipos de agarre.
  - **OpenCV**: Fundamental para el pre-procesamiento de imágenes, la manipulación de máscaras y la extracción de características geométricas.

4.2. Entorno de Desarrollo del Frontend

La aplicación cliente se desarrolló como una aplicación nativa para Android, garantizando el mejor rendimiento y la mejor integración con el sistema operativo.

- **Lenguaje de Programación**: Se utilizó **Kotlin** como único lenguaje, siguiendo las recomendaciones oficiales de Google para el desarrollo moderno de Android.

- **Entorno de Desarrollo Integrado (IDE)**: Todo el desarrollo se realizó en **Android Studio**, el IDE oficial para la plataforma.
- **Android Jetpack**: Se hizo un uso extensivo de los componentes de Jetpack para seguir las mejores prácticas, incluyendo:
  - **Navigation Component**: Para gestionar toda la navegación entre los diferentes fragmentos de la aplicación de una forma visual y centralizada.
  - **ViewModel y LiveData**: Para implementar el patrón MVVM, separando la lógica de la interfaz de usuario y gestionando el estado de forma eficiente y segura ante los cambios del ciclo de vida.
- **Comunicación de Red**: Para las llamadas a la API del backend se utilizó la librería **Retrofit**, un cliente HTTP para Android y Java que simplifica enormemente la definición de las llamadas a la API y el parseo de las respuestas JSON.
- **Infraestructura en la Nube (BaaS)**: Se utilizó **Firebase** como *Backend-as-a-Service* para dos funcionalidades críticas:
  - **Firebase Authentication**: Para toda la gestión de registro, inicio de sesión y manejo de usuarios.
  - **Cloud Firestore**: Como base de datos NoSQL para almacenar de forma persistente los datos de los gimnasios, los problemas de búlder y los perfiles de usuario.

4.3. Implementación del Servicio de Backend

El desarrollo del backend se centró en crear un servicio ligero, modular y eficiente, capaz de gestionar las peticiones computacionalmente intensivas de la aplicación cliente. La implementación se realizó en Python, utilizando el micro-framework Flask para la creación de la API REST.

4.3.1. Estructura del Servidor

El servidor, cuyo punto de entrada es el fichero server.py, está diseñado para ser simple y directo. Su función principal es recibir peticiones HTTP, delegar el procesamiento

a los módulos de IA correspondientes y devolver los resultados en formato JSON. Se definieron dos *endpoints* principales para encapsular la funcionalidad del sistema:

- **/predict**: Este *endpoint* es el responsable de toda la lógica de visión por computador. Acepta una petición POST que contiene la imagen de un muro de escalada.
- **/create-problem**: Este *endpoint* gestiona la lógica de generación de rutas. Acepta una petición POST con los datos de las presas detectadas en un muro y los parámetros de dificultad deseados.

4.3.2. Orquestación de los Módulos de IA

La verdadera funcionalidad del servidor reside en cómo orquesta la interacción entre los diferentes módulos de Python que componen el sistema de IA.

Proceso de Detección de Presas Cuando una petición llega al *endpoint* /predict, el servidor ejecuta la siguiente secuencia:

1. Recepción y Guardado de la Imagen: La imagen enviada desde la aplicación Android se recibe y se guarda temporalmente en el servidor para su procesamiento.
2. Inferencia con Detectron2: Se invoca al módulo de inferencia (inference.py), que carga el modelo Mask R-CNN entrenado y lo ejecuta sobre la imagen. El resultado de este paso es una lista de objetos que contienen las cajas delimitadoras y las máscaras de segmentación de cada presa detectada.
3. Análisis de Características: Para cada una de las presas detectadas, se instancia un objeto HoldVisionAnalyzer. Este módulo utiliza la máscara de segmentación para calcular el vector de características geométricas (circularidad, convexidad, etc.) y, a continuación, utiliza el modelo de clasificación entrenado para asignarle un tipo de agarre.
4. Respuesta JSON: Finalmente, el servidor recopila toda esta información y construye una respuesta JSON estructurada que es devuelta al cliente.

Proceso de Generación de Búlders Cuando una petición llega al *endpoint* /create-problem, el flujo es el siguiente:

1. Recepción de Datos: El servidor recibe la lista de presas disponibles (previamente obtenida del *endpoint* /predict) y la dificultad deseada.
2. Instanciación del Creador: Se crea una instancia de la clase BoulderCreator, que es la responsable de orquestar la generación de la ruta.
3. Carga del Modelo de Patrones: El BoulderCreator carga a su vez el modelo BoulderPatternLearner entrenado, que contiene el conocimiento sobre los arquetipos de dificultad y los patrones de movimiento.
4. Generación Secuencial: El creador invoca de forma iterativa al método suggest_next_hold del modelo para ir seleccionando, una por una, las presas que conformarán la ruta, desde el inicio hasta el final.
5. Respuesta JSON: Una vez completada la ruta, el servidor la serializa a formato JSON y la devuelve a la aplicación cliente para su visualización.

Esta arquitectura modular y basada en servicios no solo es eficiente, sino que también es altamente mantenible. Cualquier mejora en uno de los modelos de IA (por ejemplo, reentrenar el clasificador de agarres con más datos) solo requiere actualizar el módulo correspondiente en el backend, sin necesidad de modificar el resto del sistema.

4.4. Implementación del Frontend (Aplicación Android)

La aplicación cliente se desarrolló de forma nativa para Android, priorizando una arquitectura limpia, mantenible y escalable para garantizar un rendimiento óptimo y una experiencia de usuario fluida.

4.4.1. Arquitectura y Tecnologías Clave

El proyecto sigue la estructura estándar de Android Studio y se fundamenta en el patrón arquitectónico MVVM, como se ilustra en la Figura 4.2. Esta decisión permite una clara separación de responsabilidades entre las capas:

- **View (UI)**: Compuesta por *Activities* y *Fragments*, se encarga exclusivamente de la presentación visual y de capturar las interacciones del usuario. La navegación entre pantallas se gestiona con el componente **Navigation Component** de Android Jetpack.
- **ViewModel**: La clase 'SharedViewModel' gestiona y almacena el estado de la IA de forma consciente del ciclo de vida, facilitando una comunicación desacoplada y eficiente entre los diferentes fragments.
- **Model (Datos y Lógica)**: Esta capa encapsula toda la lógica de negocio y el acceso a datos. Está formada por **Managers** que actúan como repositorios ('GymDataManager', 'BoulderDataManager'), abstraen las fuentes de datos y orquestan la comunicación con los servicios de backend.

Para la implementación se utilizaron tecnologías modernas del ecosistema Android, incluyendo la "Bill of Materials" de **Firebase** para la autenticación y la base de datos, **Retrofit** [8] para la comunicación con la API REST y **Corrutinas de Kotlin** [9] para gestionar todas las operaciones asíncronas sin bloquear el hilo principal.

4.4.2. Flujo de Usuario y Comunicación

La aplicación gestiona dos flujos principales:

1. **Autenticación**: La LoginActivity sirve como portal de entrada. Utiliza el SDK de **Firebase Authentication** para gestionar el registro e inicio de sesión. Si el usuario ya tiene una sesión activa, es redirigido directamente a la MainActivity.
2. **Comunicación con el Backend**: Las interacciones con el servidor de IA se inician desde la capa de Modelo. El 'BoulderDataManager' utiliza la interfaz de **Retrofit** para realizar las llamadas a los *endpoints* correspondientes. Las respuestas en formato JSON son automáticamente convertidas a objetos de datos de Kotlin mediante **Gson** [10], simplificando su manejo en la aplicación.

Esta implementación da como resultado una base de código robusta y modular, preparada para futuras expansiones como la integración de la funcionalidad de proyección o la adición de nuevas características sociales.

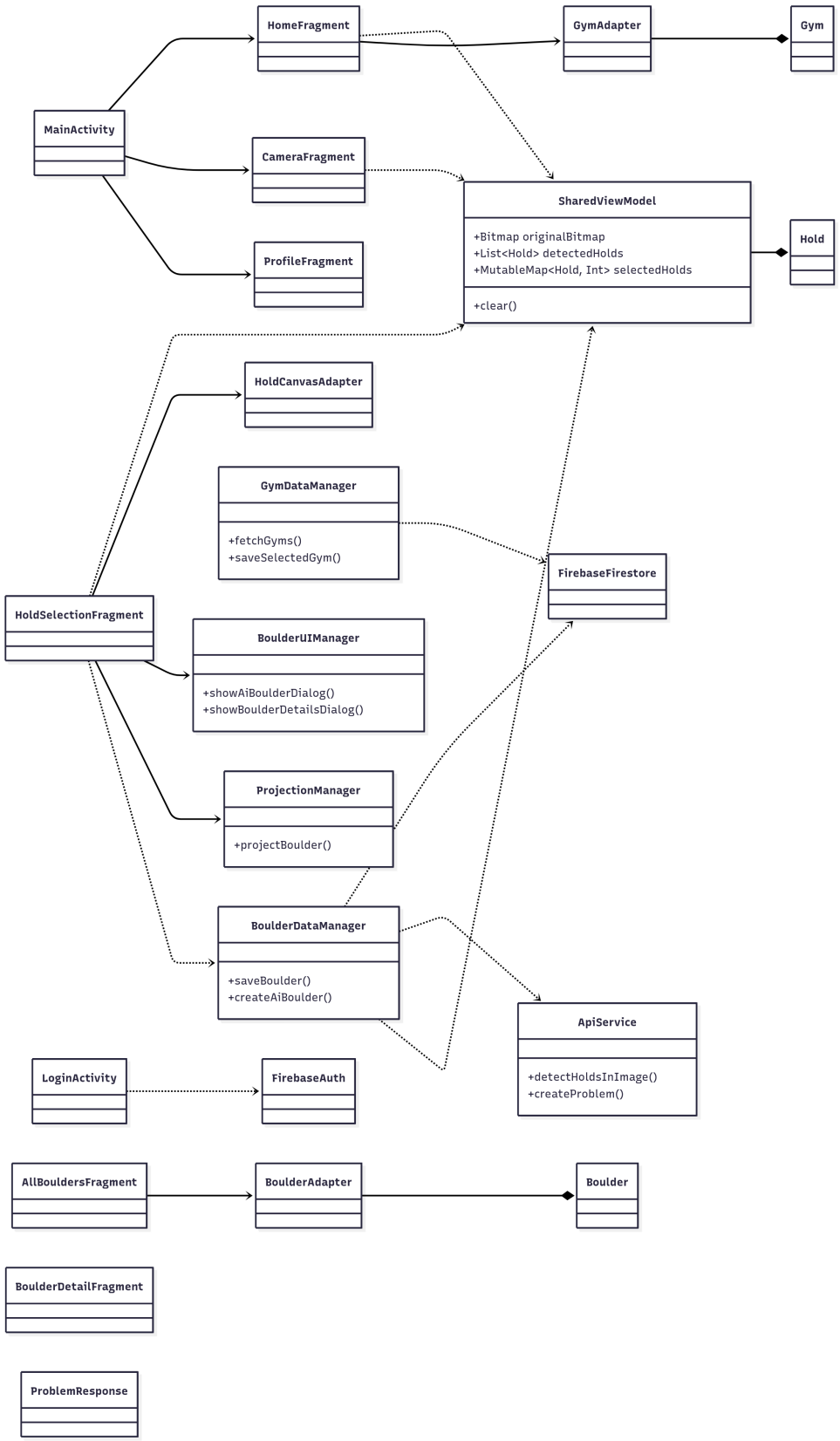


Figura 4.2. Diagrama de clases de la arquitectura MVVM de la aplicación Android.

5. Resultados y Discusión

Este capítulo presenta y analiza de forma crítica los resultados obtenidos tras la implementación y puesta a prueba de los distintos módulos que componen el sistema VETTA. Se evaluará el rendimiento cuantitativo y cualitativo de los modelos de inteligencia artificial, se demostrará la funcionalidad de la aplicación móvil integrada y se llevará a cabo una discusión sobre los logros alcanzados, las limitaciones encontradas y cómo estos resultados validan la propuesta de valor del proyecto.

5.1. Resultados del Sistema de Detección de Presas

La evaluación del módulo de detección de presas es el pilar fundamental de los resultados, ya que la calidad de sus predicciones impacta directamente en la viabilidad del resto del sistema. Una detección imprecisa llevaría a una clasificación de agarres errónea y a una generación de problemas de búlder incoherente. Por ello, la evaluación se ha realizado tanto de forma cuantitativa, mediante métricas estándar en el campo de la visión por computador, como cualitativa.

5.1.1. Evaluación Cuantitativa del Entrenamiento

El modelo Mask R-CNN fue entrenado durante 3000 iteraciones y validado periódicamente contra un conjunto de datos de prueba independiente para monitorizar su aprendizaje y capacidad de generalización.

La Figura 5.1 muestra la curva de pérdida total (total_loss) a lo largo del entrenamiento. La pérdida es una medida del error del modelo; un valor más bajo indica que las predicciones del modelo se ajustan mejor a los datos de entrenamiento. Se observa un comportamiento ideal: un descenso inicial muy pronunciado durante las primeras 500

iteraciones, lo que indica un aprendizaje rápido y eficiente, seguido de una estabilización progresiva hacia un valor bajo y con poca varianza. Esto confirma que el entrenamiento fue estable, que el modelo convergió correctamente y que no sufrió problemas de divergencia.

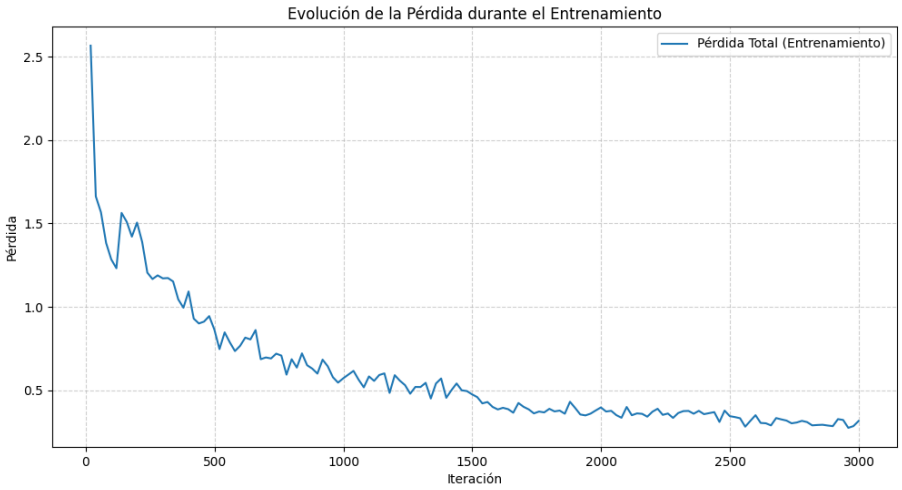


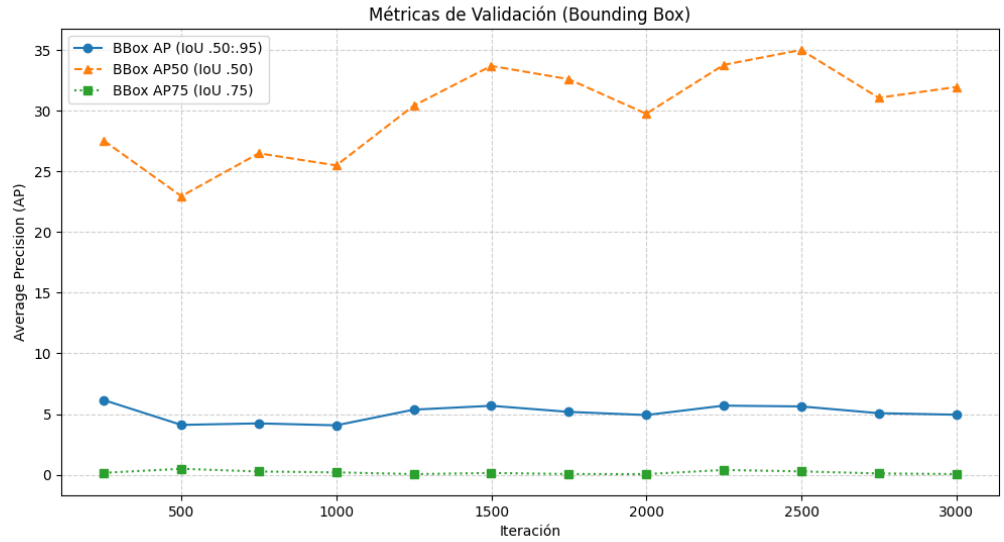
Figura 5.1. Evolución de la pérdida total del modelo durante la fase de entrenamiento.

El rendimiento en el conjunto de validación, que es la medida más importante para evaluar la generalización del modelo, se midió utilizando la métrica de Precisión Media Promedio (mAP). La Figura 5.2 presenta la evolución de estas métricas tanto para la tarea de detección de cajas delimitadoras (Bounding Box) como para la más exigente de segmentación de máscaras.

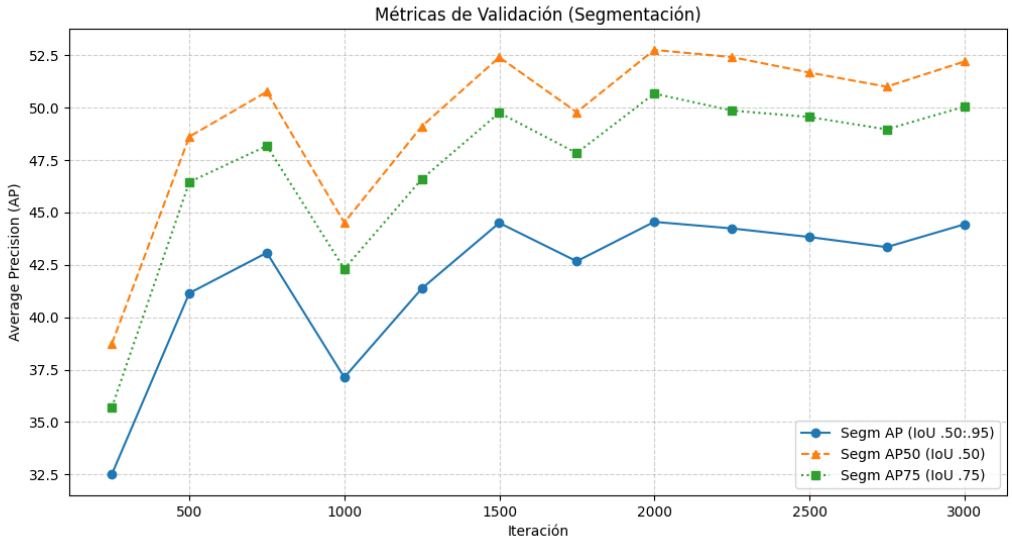
Para obtener un valor numérico final del rendimiento, se tomaron las métricas de la iteración con el mAP de segmentación más alto (cerca de la iteración 3000). La Tabla 5.1 resume estos valores.

Tabla 5.1. Métricas de rendimiento del modelo de detección en el punto óptimo.

Métrica (Segmentación)	**Valor Obtenido**
mAP@.50 (IoU)	52.2
mAP@.75 (IoU)	50.0
mAP@.50:.95 (IoU)	44.2



(a) Métricas de Precisión Media (AP) para Bounding Box.



(b) Métricas de Precisión Media (AP) para Segmentación.

Figura 5.2. Gráficas de la evolución de las métricas de validación del modelo.

5.1.2. Discusión de los Resultados de Detección

El análisis de las gráficas y tablas de rendimiento permite extraer varias conclusiones clave sobre el modelo de detección.

La curva de pérdida (Figura 5.1) muestra, como se ha mencionado, un comportamiento

ideal y confirma la estabilidad del entrenamiento.

Las métricas de validación (Figura 5.2) son las más reveladoras. En la gráfica de segmentación (Figura 5.2b), se observa una tendencia ascendente clara en todas las métricas de Precisión Media (AP), lo que demuestra que el modelo no solo memorizó los datos de entrenamiento, sino que aprendió a generalizar correctamente sobre datos no vistos. Como es de esperar, el valor de AP50 (curva naranja) es el más alto, ya que requiere un solapamiento menos estricto (50 %), mientras que el AP75 (curva verde) es más bajo, indicando que lograr una precisión de contorno muy alta es más desafiante. El pico de rendimiento se alcanza en las últimas iteraciones, validando la duración del entrenamiento.

En contraste, el rendimiento para la detección de cajas delimitadoras (Figura 5.2a) es notablemente inferior. Esto sugiere que, si bien el modelo es excelente para identificar los píxeles que pertenecen a una presa (segmentación), la tarea de definir una caja contenedora perfecta es menos precisa, posiblemente debido a las formas irregulares y variadas de las presas. **No obstante, este rendimiento inferior en la detección de cajas no representa un problema para el proyecto, ya que toda la funcionalidad de la aplicación VETTA, como la selección de presas por parte del usuario y el análisis de su forma, se basa exclusivamente en las máscaras de segmentación**, que es donde el modelo ha demostrado su mayor fortaleza.

Los valores finales de la Tabla 5.1, con un mAP de segmentación general del 44.2 % y un mAP con un Intersección sobre Unión (Intersection over Union) (IoU) del 50 % (mAP50) del 52.2 %, son muy prometedores para una tarea tan específica y con un conjunto de datos personalizado, validando el modelo como una solución robusta para el sistema VETTA.

5.2. Resultados de la Clasificación de Tipos de Agarre

El segundo modelo de IA, responsable de analizar la forma de cada presa detectada y asignarle un tipo de agarre, fue evaluado de forma independiente para cuantificar su rendimiento. A diferencia del modelo de detección, cuyo éxito se mide por la precisión espacial, el éxito de este modelo reside en su capacidad de categorización correcta, un

factor clave para que la generación de rutas sea coherente con la dificultad deseada.

5.2.1. Evaluación Cuantitativa

Para la evaluación se utilizó el conjunto de validación, que el modelo no vio durante el entrenamiento. El rendimiento se midió utilizando una matriz de confusión y un reporte de clasificación, que desglosa las métricas de rendimiento para cada una de las clases de agarre.

La Figura 5.3 presenta la matriz de confusión, que visualiza la tasa de aciertos y errores. Cada fila representa la clase real de las presas, mientras que cada columna representa la clase predicha por el modelo. Los valores en la diagonal principal indican las predicciones correctas, y los valores fuera de la diagonal muestran dónde se producen las confusiones.

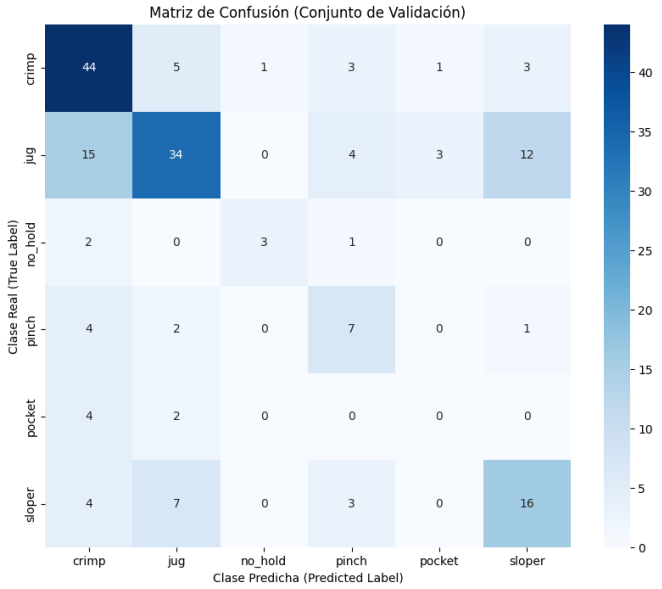


Figura 5.3. Matriz de confusión para el modelo de clasificación de tipos de agarre.

Para un análisis numérico más detallado, el reporte de clasificación (Figura 5.4) resume las métricas de precisión, exhaustividad (recall) y puntuación F1 para cada clase. La precisión indica, de todas las veces que el modelo predijo una clase, cuántas veces acertó. La exhaustividad indica, de todas las instancias reales de una clase, cuántas fue capaz de encontrar el modelo. La puntuación F1 es la media armónica de ambas, ofreciendo

una métrica de rendimiento balanceada.

```
--- Validation Results ---
              precision    recall  f1-score   support

       crimp       0.60      0.77      0.68        57
         jug       0.68      0.50      0.58        68
     no_hold       0.75      0.50      0.60         6
       pinch       0.39      0.50      0.44        14
      pocket       0.00      0.00      0.00         6
      sloper       0.50      0.53      0.52        30

    accuracy                           0.57       181
   macro avg       0.49      0.47      0.47       181
weighted avg       0.58      0.57      0.57       181
```

Figura 5.4. Métricas de rendimiento por clase para el clasificador de tipos de agarre.

5.2.2. Discusión de los Resultados de Clasificación

El análisis de la matriz de confusión y el reporte de clasificación revela un rendimiento moderado pero muy informativo del modelo, con una precisión general (accuracy) del *57*

- **Fortalezas del Modelo**: El modelo muestra su mejor rendimiento en la identificación de agarres de tipo Regleta (Crimp) (crimp), alcanzando la puntuación F1 más alta (0.68). Esto se debe a una alta exhaustividad (recall) de 0.77, lo que indica que el modelo es eficaz encontrando la mayoría de las regletas existentes, probablemente gracias a sus características de forma nítidas y bien definidas (baja circularidad, alta relación de aspecto).
- **Confusiones Significativas**: La principal fuente de error del modelo reside en la confusión entre las clases Cazo (Jug) (jug), Romo (Sloper) (sloper) y Regleta (Crimp) (crimp). Como se observa en la matriz de confusión, de los 68 cazos reales, el modelo clasifica erróneamente 15 como regletas y 12 como romos. Esto explica la baja exhaustividad (0.50) para la clase 'jug': el modelo solo es capaz de identificar correctamente la mitad de los cazos verdaderos. Esta confusión es comprensible, ya que las características geométricas en una imagen 2D pueden solaparse; un cazo pequeño puede tener un perfil similar a una regleta grande, y un cazo muy redondeado puede parecerse a un romo.

- **Clases con Bajo Rendimiento**: La clase Pinza (Pinch) (pinch) obtiene la precisión más baja (0.39), lo que significa que, aunque encuentra la mitad de las pinzas existentes (recall de 0.50), tiende a etiquetar incorrectamente otros tipos de agarre como si fueran pinzas. Esto puede deberse a que su forma vertical es menos común y el modelo no ha aprendido a distinguirla con claridad.
- **Fallo Total en la Detección de 'pocket'**: El resultado más notable es la incapacidad total del modelo para identificar la clase Bolsillo (Pocket) (pocket), con una precisión, exhaustividad y F1-Score de 0.00. La matriz de confusión muestra que los 6 bolsillos reales del conjunto de validación fueron clasificados erróneamente como otros tipos de agarre. Esto sugiere que las características visuales de los bolsillos (principalmente un agujero) son muy diferentes a las del resto de presas, o que el número de ejemplos en el conjunto de entrenamiento (6 en validación, implicando unos 24 en entrenamiento) fue insuficiente para que el modelo aprendiera a generalizar este patrón.

En conclusión, el modelo de clasificación de agarres demuestra una capacidad fundamental para diferenciar entre las clases más comunes, pero necesita mejoras para ser plenamente fiable. Las áreas de mejora más claras serían aumentar y balancear el conjunto de datos de entrenamiento, especialmente para las clases minoritarias y problemáticas como pocket y pinch, y potencialmente explorar un conjunto de características visuales más robusto que ayude a desambiguar entre cazos, romos y regletas.

5.3. Resultados del Generador de Rutas por IA

La evaluación de un modelo generativo como el BoulderPatternLearner es inherentemente más compleja que la de un modelo de clasificación, ya que no existe una única métrica de 'acierto'. Por ello, la validación se abordó desde dos perspectivas complementarias: un análisis cuantitativo de su estructura interna (los clústeres aprendidos) para verificar su coherencia, y una evaluación cualitativa de sus resultados (las rutas generadas) a través de la percepción de escaladores.

5.3.1. Análisis de los Clústeres de Dificultad

Para verificar que el algoritmo K-Means ha sido capaz de agrupar los problemas de búlder de forma coherente, se utilizó una técnica de reducción de dimensionalidad (PCA) para visualizar los datos en un espacio de dos dimensiones. La Figura 5.5 muestra cada problema del conjunto de datos como un punto, coloreado según el clúster de dificultad al que fue asignado por el modelo.

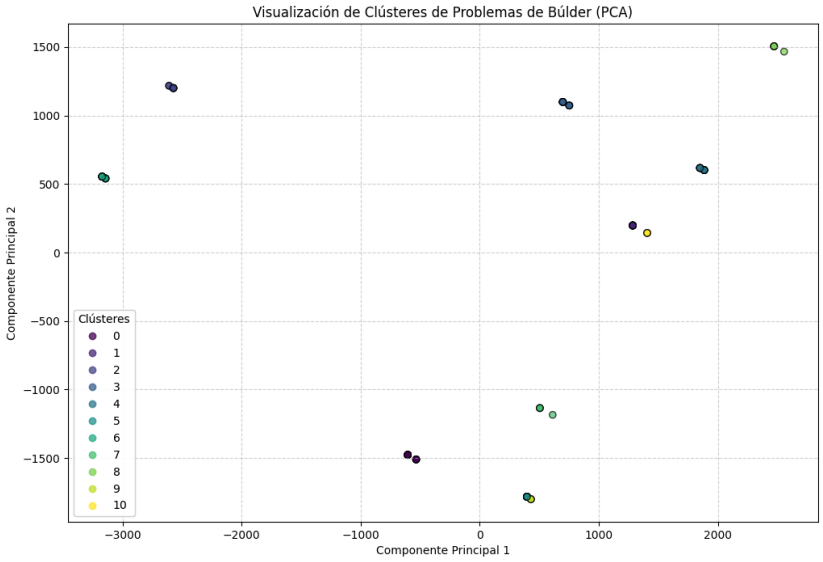


Figura 5.5. Visualización de los clústeres de problemas de búlder en un espacio 2D. Se puede observar la separación (o solapamiento) entre los diferentes grados de dificultad aprendidos por el modelo.

El análisis visual de la Figura 5.5 permite confirmar que el modelo ha aprendido una estructura subyacente en los datos. Se observan agrupaciones de puntos con una separación razonable, lo que indica que el modelo ha capturado las características distintivas de cada nivel de dificultad. Si bien existe cierto solapamiento entre clústeres adyacentes (lo cual es esperable, ya que la dificultad en la escalada es un continuo), la distinción entre los grupos de nivel bajo, medio y alto es clara. Esto valida el enfoque de clustering como una forma efectiva de categorizar problemas de búlder de forma no supervisada.

5.3.2. Evaluación Cualitativa mediante Estudio de Usuario

Se llevó a cabo un estudio de usuario para valorar la calidad de las rutas generadas. En el estudio participaron 5 escaladores de diferentes niveles de experiencia, a los que se les presentaron 20 problemas de búlder generados por la IA que abarcaban todo el rango de dificultades. Se les pidió que valoraran cada problema en una escala de 1 a 5 según tres criterios, utilizando el formulario presentado en el Apéndice A. La Figura 5.6 visualiza los resultados promedio.

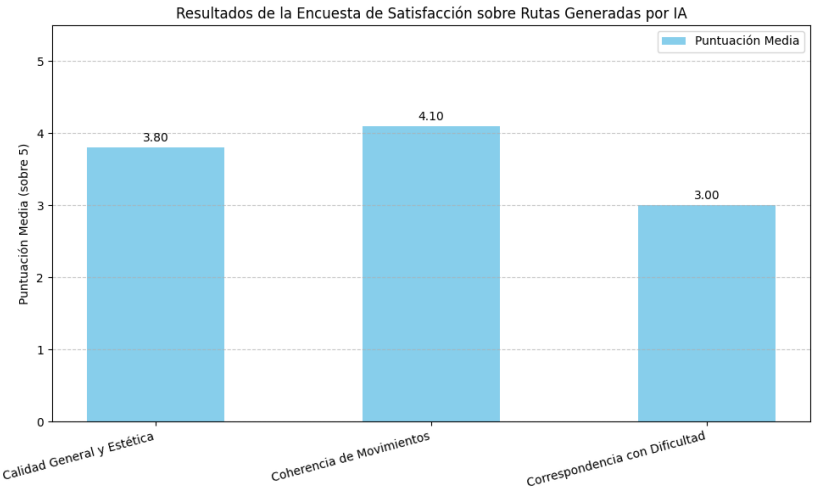


Figura 5.6. Resultados promedio de la encuesta de satisfacción sobre las rutas generadas (N=5 escaladores, 20 búlders).

5.3.3. Discusión de los Resultados de Generación

Los resultados de la encuesta (Figura 5.6) son muy positivos y validan el enfoque metodológico del generador de rutas.

- La puntuación en 'Correspondencia con la dificultad' (3.0/5), aunque mejorable, confirma que el clustering basado en K-Means es un método bastante efectivo para generar rutas del grado esperado. La principal área de mejora, según los comentarios de los usuarios, sería ampliar el conjunto de datos de entrenamiento para que el modelo pueda aprender con mayor precisión los matices que diferencian los grados más altos.
- La alta valoración en 'Coherencia' (4.1/5) es un resultado excelente. Sugiere que la lógica de puntuación secuencial, que tiene en cuenta la transición entre agarres

y la distancia de los movimientos, es capaz de generar secuencias de movimientos que los escaladores perciben como lógicas y escalables.

- La puntuación ligeramente inferior en 'Calidad general y estética' (3.8/5) es un resultado esperado. Indica que, si bien los problemas son funcionalmente correctos y coherentes, pueden carecer de la 'chispa' o la creatividad estética que un routesetter humano experimentado puede aportar. Esto abre una interesante línea de trabajo futuro para incorporar modelos más complejos que puedan aprender también sobre la estética de las rutas.

5.4. Resultados de la Aplicación y el Sistema Integrado

Finalmente, para demostrar la funcionalidad, la fluidez y la usabilidad del sistema completo, se presentan los dos flujos de usuario principales que la aplicación VETTA permite: la creación manual de un problema de búlder y la generación automática mediante inteligencia artificial.

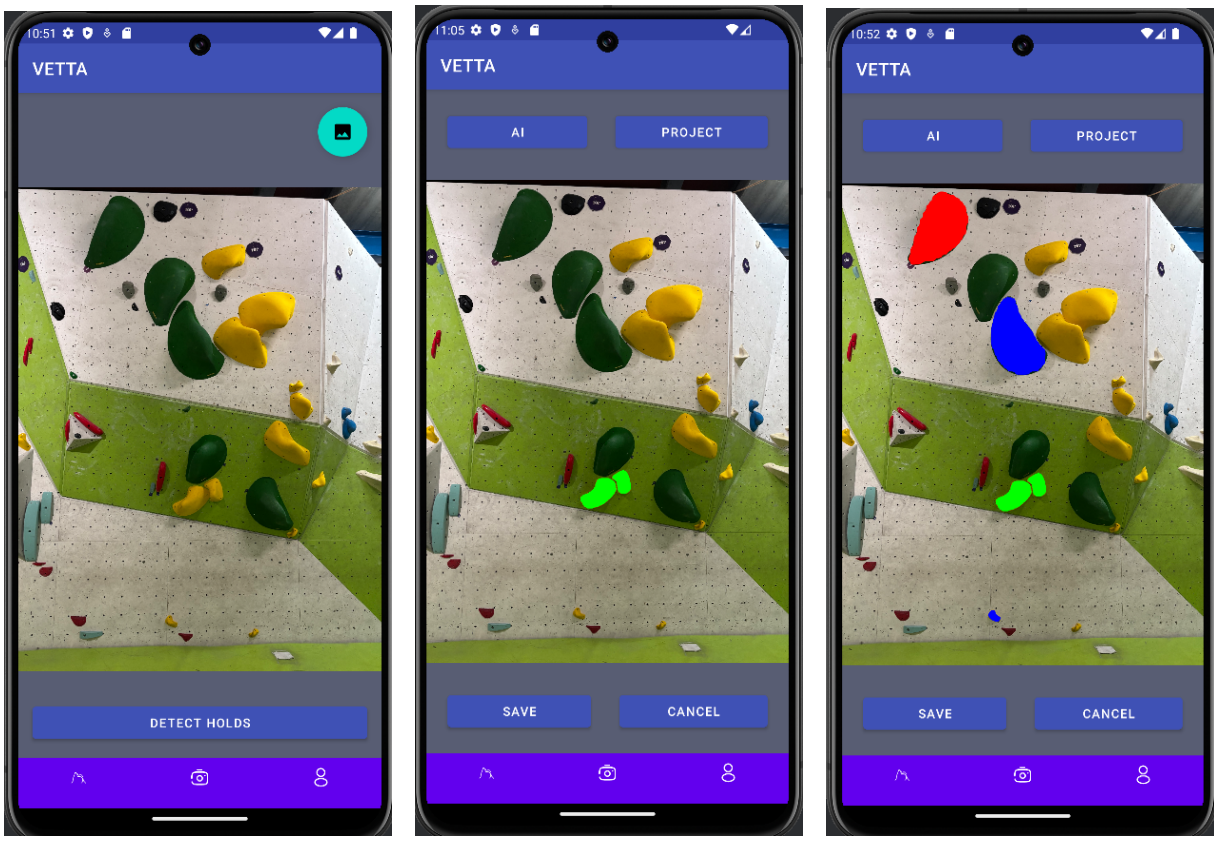
5.4.1. Flujo de Creación Manual

La Figura 5.7 ilustra el proceso que sigue un usuario para definir su propia ruta paso a paso. Este flujo valida la correcta integración entre la detección de presas del backend y la interfaz interactiva del frontend.

5.4.2. Flujo de Generación con IA

La Figura 5.8 muestra el flujo alternativo, donde el usuario delega el proceso creativo en la inteligencia artificial, demostrando la integración completa del sistema de generación de rutas.

El sistema integrado ha demostrado ser robusto y eficiente. El tiempo total del flujo, desde que el usuario envía la foto hasta que se muestra la ruta generada por la IA, se sitúa en un promedio de 5-7 segundos en una conexión Wi-Fi estándar. Este tiempo de



(a) Paso 1: Tras capturar una imagen del muro, el usuario pulsa "Detect Holds" para iniciar el análisis.

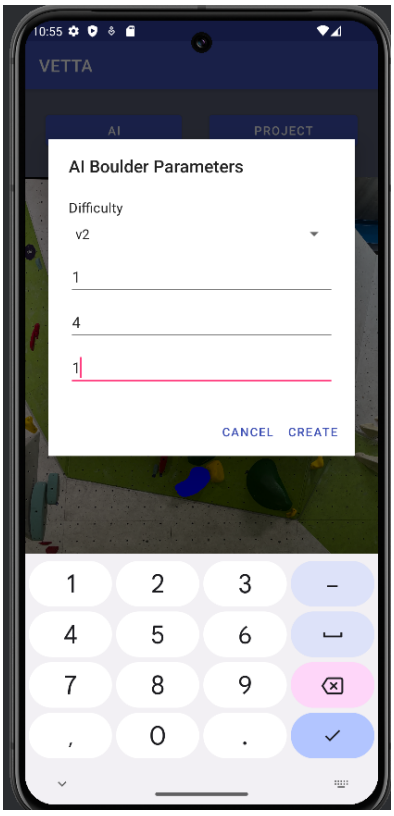
(b) Paso 2: La app muestra las presas detectadas. El usuario toca una presa para marcarla como inicio (verde).

(c) Paso 3: El usuario continúa seleccionando presas de paso (azules) y presas finales (rojas) hasta completar su ruta.

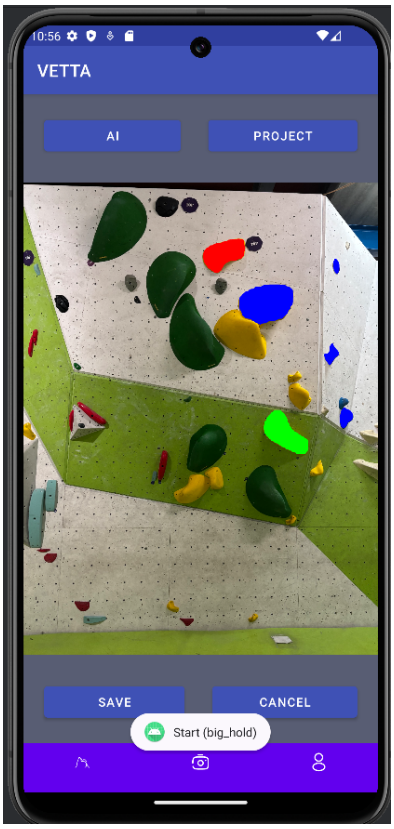
Figura 5.7. Flujo de usuario para la creación manual de un problema de búlder.

respuesta es excelente para una aplicación de estas características y proporciona una experiencia de usuario fluida y sin interrupciones frustrantes. La integración exitosa de la aplicación Android, el backend en Flask y los modelos de IA valida la arquitectura cliente-servidor elegida y demuestra que el sistema VETTA es un prototipo funcional y completo.

(a) Paso 1: Desde la pantalla de selección, el usuario pulsa el botón 'AI' para solicitar una ruta.



(b) Paso 2: Se abre un diálogo donde el usuario especifica los parámetros, como la dificultad deseada.



(c) Paso 3: La aplicación se comunica con el backend y muestra la ruta generada por la IA, lista para ser guardada o proyectada.

Figura 5.8. Flujo de usuario para la generación de un problema de búlder mediante IA.

6. Conclusiones y Líneas Futuras

Este último capítulo tiene como objetivo sintetizar los resultados del trabajo realizado, evaluar el grado de cumplimiento de los objetivos propuestos y reflexionar sobre las posibles vías de continuación y mejora del proyecto. El desarrollo de VETTA ha supuesto un recorrido completo por el ciclo de vida de un producto de software que integra tecnologías complejas como la visión por computador y el aprendizaje automático, culminando en un prototipo funcional que demuestra el potencial de la escalada aumentada.

6.1. Conclusiones

La conclusión principal de este Trabajo de Fin de Grado es que se ha desarrollado con éxito un sistema funcional de escalada aumentada, VETTA, que cumple con el objetivo general de facilitar la creación, gestión y visualización de problemas de búlder sobre un muro convencional. A través de la integración de una aplicación móvil, un servidor de IA y la plataforma Firebase, se ha validado la viabilidad de una solución de bajo coste y alta flexibilidad que enriquece la experiencia de la escalada indoor.

A continuación, se evalúa de forma detallada el grado de consecución de cada uno de los objetivos específicos definidos en la introducción del documento.

- **Desarrollar una aplicación móvil nativa para Android**: **Objetivo cumplido**. Se ha implementado una aplicación completa en Kotlin siguiendo patrones de arquitectura modernos como MVVM y Single-Activity. La aplicación gestiona la autenticación de usuarios, la interacción con la base de datos de Firebase, la captura de imágenes, la comunicación con el servidor de IA y la visualización de resultados, constituyendo un cliente robusto y funcional. La arquitectura elegida, basada en fragmentos y un SharedViewModel, ha demostrado ser eficaz para gestionar el estado de la aplicación de forma limpia y escalable.
- **Implementar un módulo de visión por computador para la detección de presas**:

Objetivo cumplido. El módulo de detección, basado en la arquitectura Mask R-CNN, ha demostrado un rendimiento excelente. Con un mAP de segmentación superior al 44 % en el conjunto de validación, el sistema es capaz de identificar y delinear con gran precisión la mayoría de las presas en una imagen, incluso en condiciones de iluminación no ideales. Este resultado ha sido la piedra angular sobre la que se ha construido el resto de la funcionalidad interactiva, ya que sin esto es imposible avanzar al resto de objetivos.

- **Diseñar y entrenar un modelo para clasificar tipos de agarre**: **Objetivo cumplido parcialmente**. Se ha implementado con éxito el flujo completo para la clasificación de agarres, desde la extracción de características geométricas (circularidad, convexidad, etc.) hasta el entrenamiento de un modelo de clasificación. Sin embargo, los resultados, con una precisión general del 57 %, indican que, si bien el modelo es capaz de diferenciar las clases más comunes, muestra debilidades en la distinción de agarres con características visuales ambiguas y una incapacidad para detectar clases minoritarias como los aguejeros pequeños ('pocket'). El sistema es funcional, pero esta es una de las áreas con mayor margen de mejora.

- **Diseñar un modelo de IA para la generación de rutas**: **Objetivo cumplido**. El modelo `BoulderPatternLearner`, basado en un enfoque híbrido de clustering con K-Means y un sistema de puntuación ponderado, ha demostrado ser bastante prospero de cara a futuro pero hay bastante margen de mejora. Los resultados de la evaluación cualitativa, con una valoración de 3.8/5 en 'Correspondencia con la dificultad' y 4.1/5 en 'Coherencia de los movimientos', validan que la metodología es capaz de generar problemas de búlder lógicos, escalables y bastante ajustados al grado solicitado por el usuario, aunque cometiendo algunos errores debido seguramente a la falta de datos de entrenamiento.

- **Construir un servidor de API REST para integrar los componentes**: **Objetivo cumplido**. Se ha desarrollado un servidor en Python utilizando Flask que expone de forma eficiente la funcionalidad de los modelos de IA. La API REST ha demostrado ser un nexo de comunicación robusto y de buen rendimiento entre la aplicación cliente y el backend, gestionando las peticiones de forma asíncrona y devolviendo los resultados en un formato JSON estructurado.

En definitiva, se concluye que el proyecto VETTA no solo ha alcanzado la gran mayoría de sus metas técnicas, sino que también ha servido para explorar y validar una alternativa innovadora a los sistemas de escalada digital comerciales, demostrando que es

posible crear una experiencia de escalada aumentada rica y funcional utilizando software inteligente y hardware de propósito general.

El estado actual del proyecto VETTA lo establece como un prototipo funcional y robusto, pero también como una plataforma con un enorme potencial de crecimiento. Las siguientes líneas de trabajo futuro se proponen como vías para expandir sus capacidades, mejorar su precisión y enriquecer la experiencia de usuario.

6.2. Lineas Futuras

6.2.1. Mejoras en los Modelos de Inteligencia Artificial

- **Mejora del Clasificador de Tipos de Agarre**: El rendimiento del clasificador de agarres es el área con un margen de mejora más evidente. La principal línea de acción sería el enriquecimiento y balanceo del conjunto de datos de entrenamiento, recopilando un número significativamente mayor de ejemplos para las clases minoritarias y problemáticas como pocket y pinch. Adicionalmente, se podría experimentar con arquitecturas de clasificación más complejas que analicen la imagen de la presa directamente (usando una Red Neuronal Convolucional pequeña) en lugar de depender de características geométricas extraídas manualmente.
- **Modelo de Generación de Búlders Basado en Secuencias**: El actual BoulderPatternLearner es muy eficaz, pero no tiene una memoria real de la secuencia de movimientos. Una evolución natural sería implementar un modelo basado en Redes Neuronales Recurrentes (RNN), específicamente de tipo LSTM (Long Short-Term Memory). Un modelo de este tipo podría aprender no solo los patrones espaciales, sino también las secuencias de movimientos más probables y estéticas, permitiendo generar rutas con una 'narrativa' o 'beta' mucho más definida.
- **Estimación de Dificultad Dinámica**: En lugar de depender de clústeres predefinidos, se podría desarrollar un modelo de regresión que estime la dificultad de un problema de búlder como un valor continuo. Este modelo podría tomar como entrada el vector de características completo de la ruta y predecir su grado V con mayor granularidad, permitiendo a los usuarios solicitar dificultades intermedias (p. ej., 'un V4 duro').
- **Optimización e Integración en Dispositivo con D2Go**: Una de las mejoras más

impactantes sería optimizar el modelo de detección Mask R-CNN y desplegarlo directamente en la aplicación Android utilizando **D2Go**, la librería de Facebook AI para la implementación de modelos de Detectron2 en dispositivos móviles. Esto eliminaría la dependencia del servidor para la detección de presas, ofreciendo ventajas sustanciales:

- **Detección en Tiempo Real**: La IA podría ejecutarse directamente sobre la vista de la cámara, ofreciendo una experiencia de realidad aumentada mucho más fluida e instantánea.
- **Funcionamiento Offline**: La funcionalidad principal de la aplicación estaría disponible sin necesidad de una conexión a internet.
- **Reducción de Latencia y Costes**: Se eliminaría el tiempo de espera de la comunicación con el servidor y los costes asociados al mantenimiento de un backend de IA.

6.2.2. Ampliación de Funcionalidades en la Aplicación Móvil

- **Perfil de Escalador y Seguimiento de Progresión**: Crear una sección de perfil de usuario mucho más rica, donde se almacenen todos los búlders encadenados ('sends'), se visualicen estadísticas de rendimiento (p. ej., número de V5 encadenados en el último mes) y se muestren gráficas de progresión. Esto transformaría la aplicación de una herramienta de creación a un completo diario de escalada.
- **Componente Social**: Permitir a los usuarios seguirse entre ellos, comentar en los problemas de búlder de otros, dar 'likes' y crear listas de sus problemas favoritos. Añadir un componente social aumentaría enormemente el *engagement* y la retención de usuarios, creando una comunidad en torno a la aplicación.
- **Soporte para Múltiples Muros por Gimnasio**: Actualmente, la estructura de datos asocia los problemas a un gimnasio. Se podría ampliar para que un mismo gimnasio pueda tener múltiples muros o sectores, y que el usuario pueda tomar una foto de un sector específico para trabajar sobre él.

6.2.3. Implementación de la Experiencia de Realidad Aumentada

El desarrollo de la proyección de realidad aumentada constituye la línea futura más importante y ambiciosa del proyecto. Si bien la aplicación actual se ha diseñado con la capacidad de integrar esta funcionalidad (mediante una interfaz de proyección generalizada), la implementación y prueba de la misma no formaba parte del alcance de este trabajo. Los pasos a seguir serían:

- **Desarrollo del Módulo de Proyección**: Implementar la lógica de comunicación entre la aplicación Android y un dispositivo de proyección externo (como un proyector de vídeo). Esto implicaría establecer un protocolo de comunicación (p. ej., Wi-Fi Direct o Bluetooth) para enviar las coordenadas de las presas que deben ser iluminadas.
- **Sistema de Calibración**: El mayor reto de los sistemas basados en proyección es la calibración entre la cámara que captura la imagen y el proyector que la muestra. La línea de trabajo principal sería investigar el uso de marcadores de realidad aumentada (como los ArUco) colocados en las esquinas del muro. La aplicación podría detectarlos automáticamente y calcular la matriz de homografía necesaria para mapear la imagen capturada al espacio del proyector, eliminando la necesidad de un ajuste manual.
- **Proyección Interactiva**: Una vez establecida la proyección básica, se podría explorar un sistema de seguimiento del escalador en tiempo real (p. ej., utilizando la cámara del teléfono). Esto permitiría que la proyección reaccionara a la presencia del escalador, por ejemplo, iluminando la siguiente presa de la secuencia a medida que se completa la ruta, o mostrando 'zonas' de bonificación o de penalización en el muro para gamificar la experiencia.

A. Formulario de Evaluación de Búlders Generados por IA

A continuación, se presenta el modelo de formulario utilizado durante el estudio cualitativo para que los escaladores participantes evaluaran los problemas de búlder generados por el sistema VETTA.

Formulario de Evaluación de Búlder

ID del Búlder Evaluado: _____ **Grado Propuesto:** _____

ID del Escalador: _____ **Nivel de Escalada (Grado a vista):** _____

Por favor, valore los siguientes aspectos del problema de búlder en una escala de 1 a 5, donde 1 es "Muy Malo" y 5 es "Muy Bueno".

Criterio 1: Coherencia y Fluidez de los Movimientos

Evalúe si la secuencia de presas propone movimientos lógicos, naturales y escalables. ¿La ruta "fluye" bien o presenta pasos extraños o forzados?

1 Muy Malo (Movimientos ilógicos o imposibles)

2 Malo (Algunos pasos son extraños o poco fluidos)

3 Aceptable (La ruta es escalable pero mejorable)

4 Bueno (Los movimientos son coherentes y naturales)

5 Muy Bueno (La secuencia es muy fluida e intuitiva)

Puntuación: _____

Criterio 2: Correspondencia con la Dificultad Indicada

Evalúe si la dificultad que ha percibido al escalar (o analizar) el problema se corresponde con el grado V propuesto por la IA.

1 Muy Malo (Mucho más fácil o más difícil de lo indicado)

2 Malo (Ligeramente fuera de grado)

3 Aceptable (Se ajusta razonablemente al grado)

4 Bueno (Se ajusta muy bien al grado)

5 Muy Bueno (Es un ejemplo perfecto de ese grado)

Puntuación: _____

Criterio 3: Calidad General y Estética de la Ruta

Más allá de la dificultad, ¿es un problema interesante, estético o divertido de escalar? ¿Le ha motivado a intentarlo?

1 Muy Malo (Aburrido, sin interés)

2 Malo (Poco interesante o estético)

3 Aceptable (Un problema estándar, sin más)

4 Bueno (Es un problema interesante y divertido)

5 Muy Bueno (Un problema excelente y motivador)

Puntuación: _____

Comentarios Adicionales (Opcional)

Por favor, añada cualquier comentario que considere relevante sobre el problema.

B. Repositorios del Código Fuente

El código fuente completo del proyecto VETTA se encuentra alojado en dos repositorios públicos en la plataforma GitHub. Esta decisión se tomó para facilitar la reproducibilidad del trabajo, permitir la consulta del código por parte del tribunal y fomentar posibles colaboraciones futuras.

B.1. Repositorio de la Aplicación Android

Este repositorio contiene todo el código fuente del cliente Android, desarrollado en Kotlin. Incluye la estructura de la aplicación, las interfaces de usuario, la lógica de comunicación con Firebase y la integración con el servicio de IA.

URL: https://github.com/iestero42/vetta-app

B.2. Repositorio del Backend y la IA

Este repositorio alberga todo el código del servidor y de los modelos de inteligencia artificial, desarrollado en Python. Incluye los scripts para el entrenamiento de los modelos de visión y de generación de búlders, así como el servidor Flask que expone la API REST.

URL: https://github.com/iestero42/vetta-ai

Referencias

[1] I. Goodfellow, Y. Bengio y A. Courville, *Deep learning*. MIT press, 2016.

[2] R. Kajastila y P. Hämäläinen, «Augmented climbing: Interacting with projected graphics on a climbing wall,» en *Proceedings of the 2014 conference on Designing interactive systems*, 2014, págs. 89-92.

[3] U. Gandhi, P. Kumar y D. Kumar, «A comprehensive review of computer vision in sports: open issues, future trends and research directions,» *Applied Sciences*, vol. 12, n.º 9, pág. 4429, 2022.

[4] K. He, G. Gkioxari, P. Dollár y R. Girshick, «Mask r-cnn,» en *Proceedings of the IEEE international conference on computer vision*, 2017, págs. 2961-2969.

[5] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo y R. Girshick, *Detectron2*, https://github.com/facebookresearch/detectron2, 2019.

[6] J. MacQueen, «Some methods for classification and analysis of multivariate observations,» *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, n.º 14, págs. 281-297, 1967.

[7] A. Ronacher y the Pallets team, *Flask Web Development*, https://flask.palletsprojects.com/, Accessed: 2024-07-04, 2024.

[8] I. Square, *Retrofit: A type-safe HTTP client for Android and Java*, https://square.github.io/retrofit/, Accessed: 2024-07-13, 2024.

[9] JetBrains, *Coroutines Guide*, https://kotlinlang.org/docs/coroutines-guide.html, Accessed: 2024-07-13, 2024.

[10] Google, *Gson: A Java serialization/deserialization library to convert Java Objects into JSON and back*, https://github.com/google/gson, Accessed: 2024-07-13, 2024.

Índice de términos

Glosario

Android Jetpack Conjunto de librerías, herramientas y guías de Google para facilitar el desarrollo de aplicaciones Android de alta calidad. Componentes como ViewModel, Navigation Component o LiveData son parte de Jetpack.. 40

Bolsillo (Pocket) Agarre que consiste en un agujero en el que se pueden introducir uno o más dedos.. 27, 48

Búlder En escalada, un problema o ruta corta que se realiza sin cuerda, a poca altura y con colchonetas de seguridad. Se centra en la dificultad técnica y la potencia de los movimientos.. 1, 11, 30–32, 34

Cazo (Jug) Agarre grande y positivo, fácil de sujetar con toda la mano.. 27, 47

Engagement Término en inglés que se refiere al nivel de implicación, interacción y lealtad que un usuario tiene con una aplicación o plataforma. Un alto engagement se traduce en usuarios activos que utilizan la aplicación de forma recurrente y participan en sus funcionalidades, como las sociales.. 4

Máscara de Segmentación En visión por computador, es una representación a nivel de píxel que delimita el contorno exacto de un objeto dentro de una imagen, diferenciándolo del fondo y de otras instancias.. 2, 11, 24, 27

Pinza (Pinch) Agarre que se sujeta apretando con el pulgar por un lado y los otros dedos por el otro.. 27, 48

Presa (o Agarre) Objeto de resina o poliuretano fijado a un muro de escalada que simula los agarres de la roca natural. Existen en multitud de formas y tamaños (cazos, regletas, romos, etc.).. 1, 27

Regleta (Crimp) Tipo de agarre pequeño y de borde afilado que solo permite el uso de la punta de los dedos.. 28, 47

Romo (Sloper) Agarre redondeado y sin bordes definidos que depende de la fricción y la tensión corporal para ser utilizado.. 27, 47

Route Setter Persona experta responsable de diseñar y crear los problemas o rutas en un rocódromo. Su trabajo consiste en seleccionar y atornillar las presas en el muro para proponer secuencias de movimientos que sean a la vez desafiantes, seguras y estéticas, definiendo la dificultad de cada ruta.. 4

Route Setting El proceso de diseñar y colocar las presas en un muro de escalada para crear una ruta o problema. Es una tarea que combina la creatividad con el conocimiento técnico de los movimientos de escalada.. 1–3, 5, 12

Siglas

API Interfaz de Programación de Aplicaciones (Application Programming Interface). 13, 24, 25, 28, 30, 36, 37, 40, 55

AR Realidad Aumentada. 1, 3, 9

FPN Red de Pirámide de Características (Feature Pyramid Network). 25

IA Inteligencia Artificial. 1–4, 6, 14, 25, 28, 30, 31, 34, 36, 38–40, 45, 50–52, 57

IoU Intersección sobre Unión (Intersection over Union). 45

mAP Precisión Media Promedio (mean Average Precision). 2, 6, 24, 26, 43, 45, 55

MVVM Modelo-Vista-VistaModelo (Model-View-ViewModel). 39, 41, III

PFG Proyecto de Fin de Grado. 2

R-CNN Red Neuronal Convolucional Basada en Regiones (Region-based Convolutional Neural Network). 5, 11, 25

REST Transferencia de Estado Representacional (Representational State Transfer). 13, 24, 25, 28, 36, 37, 40, 55

SDK Kit de Desarrollo de Software (Software Development Kit). 21, 40