UNIVERSIDAD EUROPEA DE MADRID

ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO

MÁSTER UNIVERSITARIO EN ANÁLISIS DE DATOS MASIVOS (BIG DATA)

TRABAJO FIN DE MÁSTER

APLICACIÓN DE LARGE VISION MODELS EN CONDUCCIÓN AUTÓNOMA: DETECCIÓN DE OBJETOS Y ANÁLISIS DE VIABILIDAD

ENRIQUE ÁNGEL ARRABAL RUIZ

Dirigido por

MIGUEL TORRES PORTA

CURSO 2024-2025

TÍTULO: APLICACIÓN DE LARGA VISION MODELS EN CONDUCCIÓN AUTÓNOMA: DETECCIÓN DE OBJETOS Y ANÁLISIS DE VIABILIDAD

AUTOR: ENRIQUE ÁNGEL ARRABAL RUIZ

TITULACIÓN: MÁSTER UNIVERSITARIO EN ANÁLISIS DE DATOS MASIVOS. BIG DATA

DIRECTOR/ES DEL PROYECTO: MIGUEL TORRES PORTA

FECHA: OCTUBRE de 2025

RESUMEN

La conducción es una actividad realizada cotidianamente por muchas personas, y es una tarea repetitiva, compleja, y en muchos casos, peligrosa. Requiere un buen conocimiento de normas de tráfico, atención constante, y una capacidad de reacción y decisión rápida. Estas exigencias han impulsado el desarrollo de sistemas encargados de automatizar estas tareas y poder así liberar a los humanos de estas responsabilidades.

El avance de la inteligencia artificial, especialmente de los modelos visuales de gran escala, o *Large Vision Models* (LVMs), ha abierto nuevas oportunidades en el contexto de los coches autónomos. En este Trabajo Fin de Máster se estudia el potencial de estos modelos en tareas como la detección de objetos en la carretera, concretamente vehículos típicos, evaluando así su rendimiento, precisión y aplicabilidad en escenarios reales.

Este estudio se centra en algunos de los LVMs más relevantes actualmente, destacando su capacidad para interpretar imágenes en tiempo real. Se realiza una comparativa con otros enfoques de visión artificial, tanto tradicionales como actuales, resaltando ventajas y desafíos en escenarios reales. Además, se analizan las limitaciones relacionadas con la latencia y la precisión en la detección, así como la viabilidad de su implementación en plataformas reales.

Los resultados obtenidos muestran un buen rendimiento en cuanto a precisión, tanto en la detección de objetos como en la descripción semántica de imágenes, considerando métricas clave como *precision* y *recall*. Sin embargo, para ambas tareas, la velocidad de procesamiento junto al tiempo de inferencia alcanzados por los modelos está aún muy lejos de los requisitos que demanda la conducción autónoma en tiempo real. El modelo más rápido solo ha sido capaz de alcanzar 12,64 FPS, muy lejos de los 30 demandados por la percepción visual. Estos hallazgos permiten valorar el potencial de los LVMs, pero también señalan claramente sus fuertes limitaciones en cuanto a tiempo de inferencia, sensibilidad a objetos pequeños y su elevada complejidad computacional en ciertos modelos.

Palabras clave: Conducción Autónoma, Modelos Visuales de Gran Escala (LVMs), Visión Artificial, Inteligencia Artificial, Aprendizaje Profundo, Latencia.

ABSTRACT

Driving is an activity carried out daily by many people, and it is a repetitive, complex, and, in many cases, dangerous task. It requires a good knowledge of traffic regulations, constant attention, and the ability to react and make decisions quickly. These demands have driven the development of systems aimed at automating such tasks, thereby freeing humans from these responsibilities.

The advancement of artificial intelligence, particularly large-scale vision models, or LVMs, has opened new opportunities in the field of autonomous vehicles. This Master's Thesis explores the potential of these models in tasks such as road object detection, specifically focusing on common vehicle types, and evaluates their performance, accuracy, and applicability in real-world scenarios.

The study focuses on some of the most relevant LVMs currently available, highlighting their ability to interpret images in real time. A comparison is made with other computer vision approaches, both traditional and modern, emphasizing advantages and challenges in real-world conditions. Additionally, limitations related to latency and detection accuracy, as well as the feasibility of implementation on real platforms, are analyzed.

The results show good performance in terms of accuracy, both in object detection and semantic image description, considering key metrics such as precision and recall. However, for both tasks, the processing speed and inference time achieved by the models are still far from meeting the requirements of real-time autonomous driving. The fastest model was only able to reach 12.64 FPS, well below the 30 FPS demanded by visual perception. These findings highlight the potential of LVMs, while also clearly revealing their significant limitations regarding inference time, sensitivity to small objects, and the high computational complexity of certain models.

Keywords: Autonomous Driving, Large Vision Models (LVMs), Computer Vision, Artificial Intelligence, Deep Learning, Latency.

AGRADECIMIENTOS

Quiero expresar mi más sincero agradecimiento a mi tutor Miguel,
por su guía y comprensión.

A mi madre, a mi hermana y a mi pareja,
porque han sido mi apoyo en mi peor momento.

A mi abuelo,
cuyo ejemplo y cariño llevaré siempre conmigo.

Pero especialmente a ti papá,
porque todo lo que soy, y todo lo que tengo es por ti.
Te lo dedico con todo mi corazón.

TABLA RESUMEN

	DATOS
Nombre y apellidos:	Enrique Ángel Arrabal Ruiz
Título del proyecto:	Aplicación de Large Vision Models en conducción autónoma: detección de objetos y análisis de viabilidad
Directores del proyecto:	Miguel Torres Porta
El proyecto se ha realizado en colaboración de una empresa o a petición de una empresa:	NO
El proyecto ha implementado un producto: (esta entrada se puede marcar junto a la siguiente)	NO
El proyecto ha consistido en el desarrollo de una investigación o innovación: (esta entrada se puede marcar junto a la anterior)	SI
Objetivo general del proyecto:	Estudio de la viabilidad de la inteligencia artificial en entorno de conducción autónoma

ÍNDICE

Índice de Figuras

Índice de Tablas

Capítulo 1. RESUMEN DEL PROYECTO

El objetivo de este capítulo es exponer la motivación que ha llevado al desarrollo de este Trabajo Fin de Máster, presentando el contexto, el planteamiento del problema, los objetivos del proyecto, los resultados obtenidos y la estructura del presente documento.

1.1 Contexto y justificación

La tarea de la conducción es exigente, repetitiva, en algunos casos compleja, y requiere una atención continua al volante y a elementos de la carretera. La conducción requiere de una rápida capacidad de reacción ante situaciones imprevistas, y de una toma de decisiones ágil, efectiva, y precisa. Esta carga y esta alta responsabilidad ha motivado el interés por el desarrollo de sistemas capaces de asumir, en distinto grado, las funciones típicas que realiza el humano mientras conduce.

Los avances en inteligencia artificial y, concretamente, en los modelos visuales de gran escala, abren nuevas posibilidades para mejorar la percepción del entorno por parte de los vehículos autónomos, y facilitan la automatización de las tareas que debe realizar el conductor.

1.2 Planteamiento del problema

Recientemente ha crecido el interés por el desarrollo de sistemas que sean capaces de conducir de manera automática y eso ha planteado la necesidad de mejorar la precisión, efectividad y eficiencia en la detección de objetos por parte de los modelos o agentes que implementan los vehículos autónomos, además de la criticidad del tiempo real. En este contexto, los modelos visuales de gran escala emergen como una posible solución, aunque presentan importantes desafíos técnicos que se deben estudiar antes de su pronta implementación.

En este Trabajo Fin de Máster se evalúan las principales limitaciones y dificultades técnicas que presentan estos modelos, como por ejemplo la latencia de respuesta o velocidad de inferencia de las imágenes que se procesan—un factor muy importante en entornos de tiempo real—y su complejidad computacional y precisión en la detección.

El trabajo tiene un carácter investigador, y se enmarca en el ámbito científico-técnico, teniendo por objetivo evaluar la aplicabilidad estos modelos en entornos reales. Con ello, se pretende contribuir a generar nuevos conocimientos para la comunidad investigadora que trabaja en la aplicación de estos modelos visuales de gran escala y de otros modelos de visión artificial en el ámbito de la conducción autónoma.

1.3 Objetivos del proyecto

El proyecto se estructura en torno a un objetivo general y varios objetivos específicos. El objetivo general del proyecto es evaluar la aplicabilidad y rendimiento de los LVM más relevantes en la actualidad, en tareas propias de la conducción autónoma, concretamente la detección de vehículos en imágenes, que consiste en detectar y clasificar los objetos de una imagen sin

información previa, es decir, sin haber sido entrenados sobre el conjunto de imágenes o texto sobre los que se van a evaluar. También se va a evaluar el rendimiento que tienen los modelos para describir semánticamente una imagen a través de una pregunta. De esta forma, no solo se evaluará la detección de objetos, sino también si el modelo es capaz de interpretar la imagen correctamente.

Los objetivos específicos que se plantean están orientados a tareas más analíticas y experimentales que buscan comprender el funcionamiento y las capacidades de estos LVM. En primer lugar, se busca analizar las arquitecturas que las componen, así como los principales modelos, y compararlos con enfoques tradicionales de visión artificial, como las redes neuronales convolucionales. Además, se aborda la evaluación de su rendimiento en tareas concretas como la detección de objetos y la descripción semántica de imágenes. Por último, en este Trabajo Fin de Master, se analizan las limitaciones prácticas en el contexto de la conducción autónoma, identificando fortalezas y debilidades y se planean futuras líneas de investigación orientadas a mejorar y adaptar estos modelos al entorno.

1.4 Resultados obtenidos

Los modelos evaluados han presentado un buen rendimiento en términos de precisión, tanto en la tarea de detección, como en la de descripción de imágenes, considerando diversas métricas de *precision* y *recall*. Sin embargo, su baja velocidad de procesamiento está aún lejos de los requisitos de la conducción autónoma en tiempo real. El modelo más rápido en procesar imágenes ha sido OmDet-Turbo, que alcanzó 12,64 FPS, lejos del mínimo de 30 FPS necesario para asemejar el procesado a la visión humana. Por otro lado, GroundingDINO-base ha destacado por la calidad de su detección de objetos, mientras que LLaVAv1.6 ha sido el más preciso en la descripción de imágenes. Los resultados permiten valorar el potencial de estos modelos, pero señalan claramente las limitaciones actuales de tiempo de inferencia, sensibilidad a objetos pequeños y su alta complejidad computacional que impiden su actual implementación en entornos reales.

1.5 Estructura de la memoria

El presente documento se estructura en los siguientes apartados:

1. **Resumen del proyecto:** Se presenta una visión general del trabajo, incluyendo el contexto y motivación que han llevado al desarrollo del trabajo, así como el problema planteado. Después, se detallan los objetivos y los resultados obtenidos. Por último, se expone la estructura del documento.
2. **Antecedentes y Estado del arte:** En este apartado se detallan los antecedentes del proyecto, y se revisan conocimientos de interés y trabajos similares para entender la naturaleza del proyecto.
3. **Objetivos:** En este capítulo se exponen tanto los objetivos generales como los específicos que se deben cumplir con el desarrollo del trabajo, así como sus beneficios.
4. **Desarrollo del proyecto:** En este apartado se describe el desarrollo del proyecto, incluyendo la planificación de este, una descripción de la solución y metodologías

utilizadas, una descripción de los recursos requeridos y el presupuesto del proyecto. Además, se expone la viabilidad del proyecto y los resultados de este.

5. **Discusión:** En este apartado se discuten los resultados obtenidos, y también se exponen las limitaciones de los modelos de inteligencia artificial empleados. También se exponen los cambios que han surgido respecto al planteamiento inicial.
6. **Conclusiones:** Se presentan las conclusiones obtenidas en el proyecto, tanto técnicas como personales, analizando la experiencia personal y los resultados obtenidos.
7. **Futuras líneas de trabajo:** En este apartado se exponen y analizan aspectos que están fuera del alcance del proyecto y que han surgido en base a las limitaciones encontradas, pero que son interesantes para desarrollar en futuras líneas de investigación.
8. **Referencias:** Incluye todas las fuentes bibliográficas que han sido utilizadas para el desarrollo del proyecto.
9. **Anexos:** En este apartado se incluye documentación complementaria relevante para completar el proyecto, como imágenes de código desarrollado.

Capítulo 2. ANTECEDENTES / ESTADO DEL ARTE

En este capítulo se presenta el estado del arte relacionado con este Trabajo Fin de Máster, así como el contexto, la justificación y el planteamiento del problema que se aborda. Se analiza la evolución de la conducción desde sus inicios hasta la actualidad, con especial atención al papel de la visión artificial en el desarrollo de la conducción autónoma. Además, se profundiza en el uso de los *Large Vision Models* y se examinan los principales desafíos y limitaciones que estas tecnologías presentan en la actualidad, así como sus fortalezas.

2.1 Estado del arte

La conducción autónoma es uno de los grandes retos a los que se enfrenta la inteligencia artificial, dónde se combina la visión artificial, la percepción computacional, y la toma de decisiones en tiempo real. En los últimos años, el uso de modelos de visión artificial, concretamente los *Large Vision Models* o LVMs, han cobrado protagonismo gracias a su capacidad para interpretar escenarios complejos y detectar gran variedad de elementos dentro de las imágenes, lo que ha planteado grandes retos y oportunidades en el desarrollo de vehículos autónomos.

2.1.1 Evolución de la conducción autónoma

2.1.1.1 Breve historia de la conducción autónoma

El primer indicio de la conducción autónoma surge en Nueva York en el año 1925, cuando los conductores de dos de las vías más concurridas de Manhattan, Broadway y la Quinta Avenida, observaron un vehículo que circulaba sin conductor. El responsable de este acontecimiento fue la empresa Houdina Radio Control, que utilizó un coche de la marca Chandler para desplazarse mediante un sistema de radiocontrol. Este consistía en una antena que recibía señales enviadas desde otro coche que circulaba detrás.

En 1939, Norman Bel Geddes, diseñador industrial de la época, se inspiró en dicho diseño para estructurar Futurama, una maqueta de una ciudad futurista basada en Nueva York, donde los vehículos se movían libremente propulsados por campos electromagnéticos generados gracias a circuitos que estaban insertados en el suelo.

En los años 80, la idea de desplazar los vehículos mediante tecnología integrada en el pavimento dio lugar a nuevas aproximaciones. Ernst Dickmanns, en la Universidad de Múnich, diseñó un coche, robotizado, capaz de procesar imágenes para dirigir el vehículo de manera autónoma. Al mismo tiempo, en Estados Unidos, la agencia DARPA del Departamento de Defensa, desarrolló el primer prototipo que funcionaba mediante radar láser y visión computarizada, sentando las bases del coche autónomo moderno [1].

2.1.1.2 Niveles de autonomía en la conducción autónoma.

La *Society of Automotive Engineers* (SAE International) ha establecido una clasificación de seis niveles de autonomía, que van del nivel 0 al nivel 5, y permiten categorizar el grado de automatización en los vehículos autónomos:

- **Nivel 0 – Sin automatización**: El conductor, que es un humano, tiene todo el control y es el encargado de realizar todas las tareas tanto de percepción como conducción.
- **Nivel 1 – Asistencia al conductor**: El vehículo puede realizar algunas acciones mínimas, como manejar la dirección, pero todas las acciones restantes e importantes son controladas por el humano y este debe estar completamente atento.
- **Nivel 2 – Automatización parcial:** El vehículo ya puede llegar a realizar acciones importantes como acelerar, a través de la información que obtiene de sensores. En este nivel todavía es necesaria la presencia de un conductor responsable de llevar a cabo las tareas primarias.
- **Nivel 3 – Automatización condicionada:** El vehículo comienza a conducir en la mayoría de las ocasiones, como puede ser un cambio de carril. A pesar de este nivel de autonomía, el conductor puede decidir realizar estas acciones si así lo decide.
- **Nivel 4 – Alta automatización:** El vehículo tiene capacidad para asumir la conducción frente a cualquier situación sin requerir la intervención de un humano. Sin embargo, ese piloto automático se puede desactivar en situación climáticas adversas. Actualmente, Google se encuentra desarrollando vehículos autónomos en este nivel.
- **Nivel 5 – Automatización completa:** Este nivel de autonomía se caracteriza por la completa capacidad del vehículo para conducir en cualquier situación, independientemente de la complejidad de dicha situación, sin la necesidad de la intervención de un humano [2].

A continuación, se muestra una tabla con los detalles de estos niveles de autonomía:

Nivel	Descripción	Conductor	Detección de objetos	Límites del sistema	Control de movimientos	Ejemplos
0	No hay automatización	Humano	No	No hay sistema	No hay automatización	-
1	Asistencia al conductor	Humano	No completa	Condicionado (limitado a ciertas situaciones)	Movimiento longitudinal o lateral	control de crucero adaptativo
2	Automatización parcial	Humano	No completa	Condicionado (limitado a ciertas situaciones)	Movimiento longitudinal y lateral	Asistente Piloto de Volvo
3	Automatización condicionada a la conducción	Humano-sistema	Completa	Condicionado (humano preparado para intervenir)	Movimiento longitudinal y lateral	Asistente de atascos de Audi
4	Automatización elevada de la conducción	Sistema	Completa	Condicionado (sistema puede no seguir conduciendo en ciertas situaciones)	Movimiento longitudinal y lateral	Waymo Firefly de Google
5	Automatización completa de la conducción	Sistema	Completa	Inexistentes	Movimiento longitudinal y lateral	-

Tabla 1. Características de los diferentes niveles de automatización de la conducción [3]

Actualmente existe competencia en el sector de automoción por alcanzar los niveles 4 y 5 de autonomía, ya que se considera la cima del desarrollo tecnológico en este ámbito. En este punto se encuentran empresas como Google, Tesla, Uber, Apple, BMW, Mercedes-Benz, Volkswagen, Toyota, Audi, entre muchas otras, y están invirtiendo grandes recursos en investigación y desarrollo con el objetivo de liderar este nuevo mercado. El logro de estos niveles podría representar una ventaja comercial estratégica en la movilidad del futuro [2].

2.1.1.3 Principales tecnologías en la conducción autónoma

El desarrollo de vehículos autónomos requiere la integración de distintas tecnologías encargadas de la percepción del entorno, la toma de decisiones, y la ejecución de acciones en tiempo real. En los vehículos, estas tecnologías trabajan conjuntamente para alcanzar distintos niveles de autonomía, concretamente los definidos por la SAE. A continuación, se detallan los principales sensores encargados de obtener la información del entorno.

LiDAR

El nombre LiDAR proviene de las siglas en inglés *Laser Imaging Detection and Ranging* [4]. Esta tecnología utiliza rayos láser para analizar cómo se dispersa la luz en la atmósfera. Es cierto que la interpretación de los datos que devuelve LiDAR pueden haber sido afectados por la pérdida de señal al atravesar aire. No obstante, sigue siendo una herramienta muy útil para detectar objetos y obstáculos en el entorno. En situaciones específicas también permite obtener información sobre las propiedades ópticas del aire, lo que puede ser útil para detectar condiciones meteorológicas adversas [5]. En el ámbito de los vehículos autónomos, LiDAR juega un papel fundamental, ya que genera un mapa tridimensional del entorno que rodea al vehículo. Esto lo hace midiendo precisamente el tiempo que tarda el láser en salir desde el objeto al vehículo que lo ha emitido [4]. Esta capacidad para mapear el entorno en 3D con alta precisión

es una de las razones por las que se ha consolidado como una tecnología esencial para los vehículos autónomos. En la siguiente imagen se puede ver una demostración visual de como funciona está tecnología y la imagen 3D que genera en torno al vehículo.

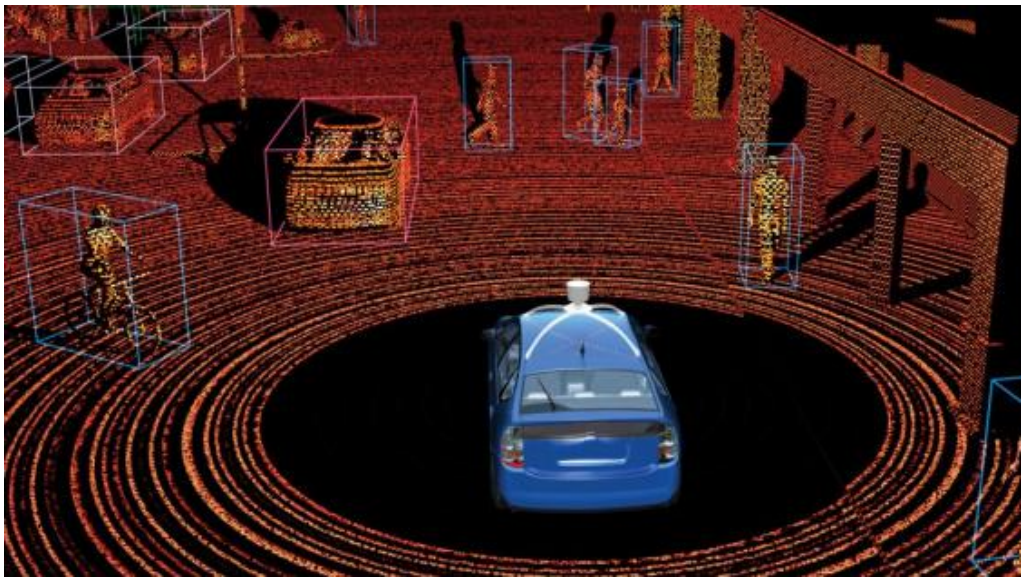


Ilustración 1. Mapeado 3D generado por LiDAR [4]

Radar

El radar (*Radio Detection and Ranging*) es una tecnología que se utiliza en el ámbito de la automoción ya que permite detectar la posición, velocidad y tamaño de objetos en la carretera a través de ondas electromagnéticas. Cuando estas ondas rebotan en un objeto, el sistema analiza el eco recibido para determinar su ubicación, su movimiento y su velocidad.

En los vehículos autónomos se pueden emplear radares de corto alcance y de largo alcance, y normalmente se utilizan ambos. Los de corto alcance son útiles para detectar bajas velocidades y para detectar obstáculos cercanos con un rango de 30 metros. Por el contrario, los de largo alcance pueden detectar objetos a 200 metros, y esto los hace adecuados para la conducción a alta velocidad.

Además de ofrecer esta información, algunos radares pueden activar sistemas de asistencia a la conducción, como puede ser el freno automático de emergencia o corregir la trayectoria en caso de detectar una colisión inminente.

Una de sus principales ventajas es su alta fiabilidad en condiciones climáticas adversas, como lluvia, y esto los convierte en sensores clave en cualquier nivel de autonomía. Se espera que con la implementación de nuevas tecnologías los radares sean aún más precisos [6].

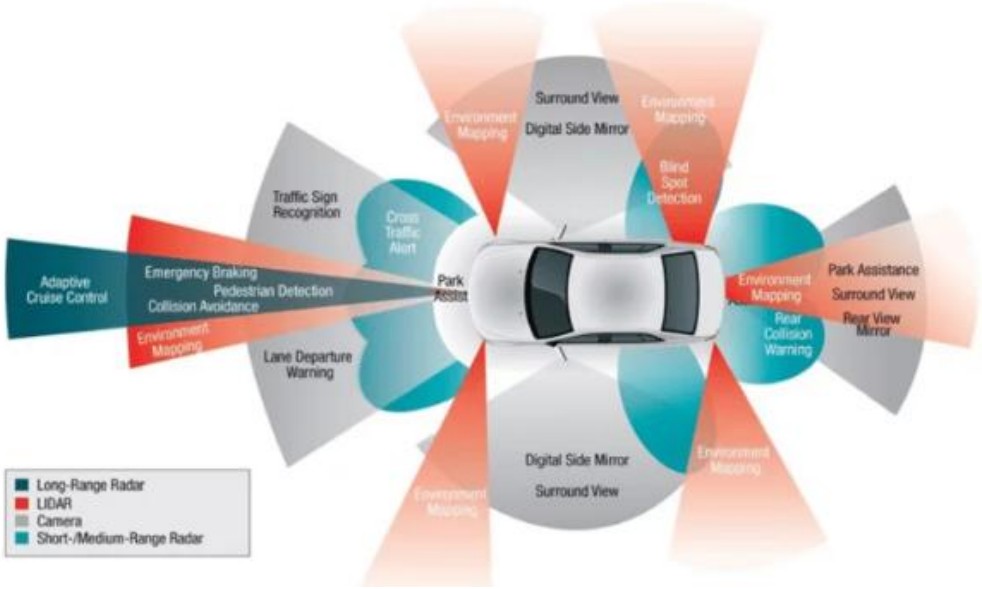


Ilustración 2. Sensores acoplados a un vehículo autónomo [7]

Sonar

Los Sonar, también conocidos como sensores de ultrasonido, en inglés *SOund Navigation And Ranging*, funcionan de manera similar al radar, pero en este caso utilizan ondas sonoras de alta frecuencia y no ondas electromagnéticas. Estas ondas, casi imperceptibles para el oído humano, rebotan en objetos cercanos al vehículo y permiten al sistema calcular la distancia al objeto en función del tiempo de retorno del eco. Se emplean principalmente en situaciones de baja velocidad, debido a su mayor precisión a corta distancia. Estas situaciones pueden ser situaciones de aparcamiento asistido, o en la detección de vehículos muy próximos. Muchos fabricantes los están integrando actualmente en sistemas de asistencia. En el caso de Tesla, algunos modelos incorporan sensores ultrasónicos que se encuentran distribuidos por todo el vehículo ofreciendo una cobertura 360º, mejorando la detección lateral y frontal, y complementando la funcionalidad del radar [6].

Cámaras

Los vehículos autónomos integran varias cámaras en distintos puntos de la carrocería estratégicamente distribuidos para generar una imagen 3D de los objetos cercanos al vehículo. Estas cámaras pueden estar en el maletero, en la parte frontal, en los laterales, y actúan de modo complementario a otros sensores, ya que ofrecen información visual muy importante.

Estos sistemas están basados en lentes monofocales, y cuentan con dos cámaras de visión únicas, muy similar a como funciona la vista humana.

Gracias a estas cámaras, se pueden detectar tanto vehículos cercanos, como peatones, ciclistas, e incluso señales de tráfico. Los sistemas más avanzados pueden llegar a combinar las cámaras con algoritmos que procesan las imágenes, de manera que no solo detecten los objetos cercanos al vehículo, sino que también los identifique y pueda predecir sus posibles movimientos.

Ilustración 3. Ejemplo de cámara trasera de vehículo [8]

2.1.2 Visión artificial en la conducción autónoma

La visión artificial es una disciplina científica que está formada por las técnicas de captación, tratamiento, procesamiento y análisis de imágenes. El objetivo de esta es extraer información importante de una imagen y así responder a preguntas, como: "¿Qué es lo aparece en la imagen?" o "¿De qué color es el objeto que sale en la imagen?". Gracias a esta información, los sistemas pueden permitir o restringir accesos a ciertos lugares, o pueden proporcionar información turística del sitio que se visita. Estos son solo dos ejemplos de los muchos campos en los que se pueden emplear, entre ellos:

- **Seguridad**: Se puede utilizar la visión artificial para controlar accesos, como por ejemplo el sistema *FaceID* de iPhone, que permite desbloquear el móvil en función de la cara reconocida.
- **Industria**: Se pueden comparar imágenes de objetos fabricados y automatizar procesos de control de calidad.
- **Comercio**: Se puede utilizar imágenes de casas para analizar el aspecto que podrían tener ciertas obras o ciertos muebles sin tener que gastar un dinero innecesario.
- **Educación**: Se pueden desarrollar programas que puedan leer fórmulas matemáticas o ayudar en tareas que son difíciles de comprender para el humano.

Para conseguir todo esto, la visión artificial se apoya en 3 pilares, concretamente la adquisición, el procesamiento y el análisis de imágenes del mundo real.

A través de la adquisición de imágenes, el mundo que vemos o la imagen captada se traduce en datos binarios, descomponiendo la imagen en 0s y 1s.

Con el procesamiento de imágenes, se utilizan algoritmos matemáticos que destacan ciertas características de las imágenes. Gracias a distintas operaciones como el recorte, el escalado, el cambio de espacio de colores y la aplicación de filtros de imágenes, se puede centrar la atención en determinados objetos de las imágenes.

Por último, el análisis de imagen, basado en algoritmos matemáticos, parte de los resultados del procesamiento de imágenes, y realizará un reconocimiento de objetos, su clasificación, o su identificación y seguimiento [9].

2.1.2.1 Detección de objetos. Tipos de métodos utilizados

La identificación de objetos pequeños, como peatones, señales de tráfico o vehículos a larga distancia, plantea retos significativos en la visión artificial. Su reducido tamaño en la imagen, su alta densidad y variabilidad de representación, su orientación o las distintas condiciones de iluminación hacen que los modelos clásicos basados en características manuales, como HOG resulten insuficientes para detectar objetos correctamente.

Con la irrupción de las redes neuronales convolucionales (CNN), se introdujeron arquitecturas capaces de aprender representaciones jerárquicas directamente de los datos, y esto les ha permitido adaptarse mejor a la detección de objetos de pequeñas dimensiones en la imagen (un objeto enorme puede ser considerado pequeño si sale muy lejos en la imagen). Las capas convolucionales y de *pooling* permiten extraer características robustas y que discriminan al objeto, y esto facilita la detección de los objetos incluso en condiciones adversas [10].

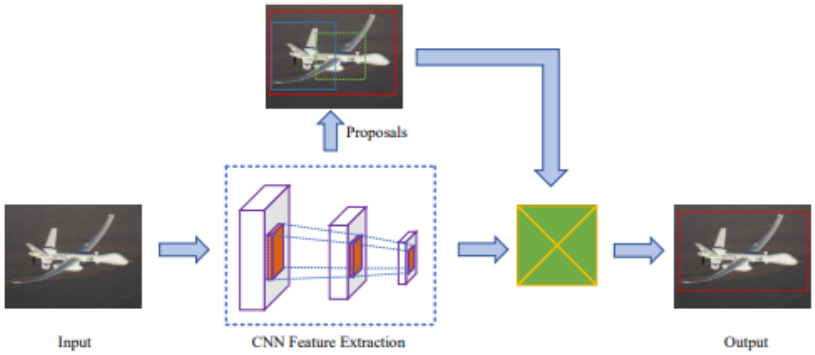


Ilustración 4. Proceso de detección de objetos a través de CNN [10]

Actualmente, los enfoques basados en aprendizaje profundo para detección de objetos incluyen:

- **Métodos de detección basados en anchors**: Producen cuadros candidatos o cajas ancla que se filtran por clasificación y regresión. Estos métodos parten de arquitecturas de redes neuronales convolucionales y mejoran la detección de objetos pequeños, pero tienen un entrenamiento lento. Entre estos se pueden clasificar los detectores de dos etapas y los detectores de una sola etapa.
    - **Detectores de dos etapas**
        - En la primera etapa, se extraen los *frames* o regiones candidatas, es decir, regiones donde puede haber objetos. En la segunda etapa, se pasa cada región candidata a una red que clasifica que objeto es y ajusta las cajas delimitadoras.
        - Tienen mucha precisión y suelen detectar bien, pero son más lentos.

        - Algunos ejemplos son R-CNN, SPPNet, Fast R-CNN, Faster R-CNN, FPN, entre otros muchos detectores [10].

    - **Detectores de una sola etapa**
        - Estos detectores no generan regiones candidatas, sino que se realiza una sola pasada sobre la imagen y la red tiene que predecir los objetos que hay, donde están y la probabilidad asociada.
        - Estos detectores son mucho más rápidos y simples, pero tienen menor precisión.
        - Algunos ejemplos son YOLO (versiones v1–v9), que han evolucionado hacia mejor precisión y velocidad, SSD, RetinaNet y EfficientDet [10].

- **Métodos anchor Free**: Estos métodos no utilizan anchors predefinidos o cajas ancla, sino que directamente detectan mediante puntos clave y centroides.
  Estos métodos tienen menos parámetros manuales y menos cajas inútiles, y esto mejora precisión y eficiencia. Sin embargo, objetos muy pequeños presentan problemas y es necesaria una buena asignación de etiquetas. Algunos ejemplos son Centernet, FCOS y TTFNet [10].

- **Métodos basados en Transformers:** Aplican el paradigma *Transformer*, inicialmente usado en NLP, en inglés *Natural Language Processing*, a visión por computador.

  El desarrollo de este Trabajo de Fin de Máster está orientado a conocer, investigar, aplicar y evaluar estos métodos basados en Transformadores, donde entran los *Large Vision Models* o LVM.

  Se pueden encontrar los siguientes métodos:
    - DETR: trata la detección como una tarea de predicción de conjunto sin necesidad de *anchors*.
    - YOLOS: adapta ViT (*Vision Transformers*) para detección de objetos, demostrando su capacidad de generalización.
    - Swin Transformer**:** son una mejora sobre los *Transformers* clásicos aplicando ventanas deslizantes jerárquicas para mejorar eficiencia en visión [10].

Además, para abordar específicamente la detección de objetos pequeños, se han propuesto distintas estrategias para mejorar estos detectores:

1. Predicción multiescala: Utiliza características extraídas en diferentes resoluciones, y de esta manera se detectan objetos en varios tamaños.
2. Mejora de la resolución de características: Emplea técnicas que recuperan detalles muy específicos.
3. Información contextual: Aprovecha el entorno del objeto para mejorar la detección cuando el objeto ofrece pocas pistas visuales.
4. Aumento de datos: Utiliza *data augmentation*, es decir, se aumentan los datos, específicamente para mejorar la robustez del modelo frente a variaciones.

5. Arquitecturas y entrenamientos especializados: Redes diseñadas o entrenadas específicamente para mejorar la detección de objetos pequeños.
6. Corrección de discontinuidades en bordes: Métodos que corrigen errores en segmentación o detección causados por bordes no alineados [10].

2.1.2.2 Redes de neuronas profundas

El aprendizaje profundo o *Deep Learning* se centra en identificar qué componentes modificables de un sistema de aprendizaje son responsables de su éxito o fracaso, y cómo ajustar sus pesos para mejorar su comportamiento, refiriéndose a la cantidad de neuronas y capas, y los pesos entre ellas.

Una red neuronal está compuesta por unidades de procesamiento llamadas neuronas, que se activan mediante entradas o conexiones ponderadas desde otras neuronas. El objetivo de la tarea del aprendizaje es actualizar esos pesos o conexiones ponderadas para que la red neuronal consiga un comportamiento deseado respecto a una determinada tarea. Este proceso de aprendizaje puede requerir múltiples etapas de procesamiento de tipo no lineal. El método de *backpropagation* o retropropagación, es clave para el aprendizaje supervisado y no supervisado, ya que permite actualizar los pesos "hacia atrás" gracias al método del descenso de gradiente aplicado a las neuronas. Las redes neuronales profundas, contienen muchas de estas etapas de procesamiento no lineal, a diferencia de las redes poco profundas [11].

Estas redes neuronales profundas, también llamadas redes neuronales convolucionales, son redes neuronales específicamente adaptas para visión artificial, ya que pueden manejar entradas de altas dimensiones como imágenes e incluso videos. Estas redes disponen de muchas capas convolucionales y capas de *pooling,* que realizan distintas tareas respectivamente. Las capas convolucionales son un grupo de mapas de activación que extraen características, llamadas *features,* de la imagen, ya que aplican un filtro a una imagen utilizando matrices. Por otro lado, la capa de *pooling* permite reducir las dimensiones de la imagen y resumir esas características por grupos.

Además, disponen de unas funciones, denominadas ReLU, que permiten aplicar la no linealidad, al utilizarse como función de activación de las neuronas [12].

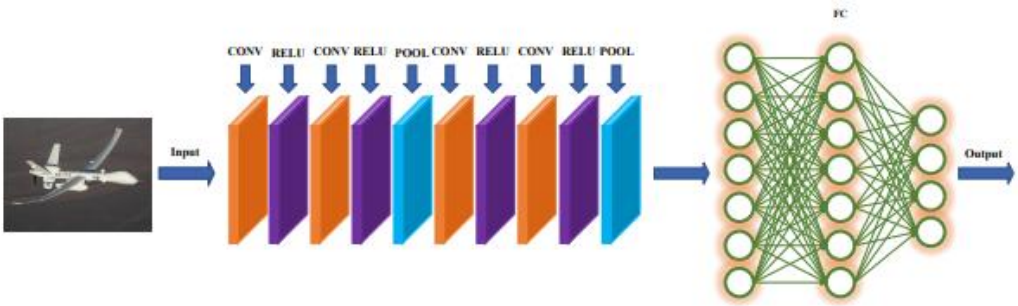


Ilustración 5. Diagrama para la detección de objetos en una CNN [10]

Las redes convolucionales disponen de una red neuronal situada al final de la arquitectura, denominadas *Fully Connected Layer*, que permiten aplanar el vector de entrada que recibe. Esta

se trata de una red neuronal estándar para la tarea de clasificación [12]. Por tanto, la estructura típica de una red convolucional está formada por esta arquitectura en el orden seguido:

- Capa de entrada
- Red convolucional
- Función ReLU
- Pooling
- Fully Connected Layers
- Softmax

La arquitectura típica y el funcionamiento de una red convolucional se puede ver en la siguiente imagen:

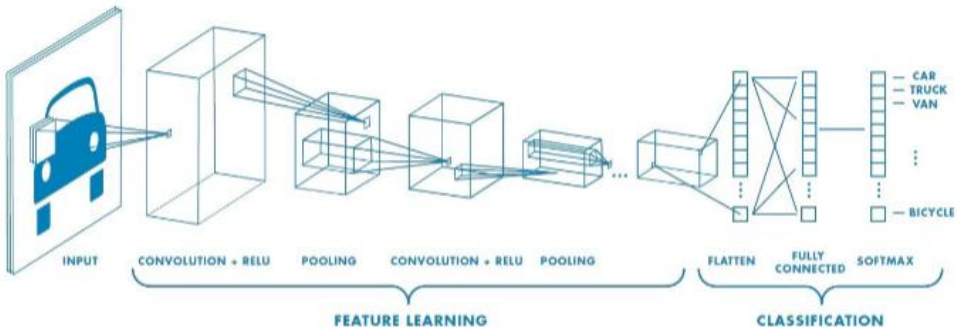


Ilustración 6. Capas de una Red Neuronal Convolucional [12]

2.1.2.3 Principales modelos de visión artificial: YOLO, Faster R-CNN.

Como se ha indicado anteriormente, los modelos de visión artificial para detección de objetos suelen clasificarse en dos enfoques principales dentro de los detectores basados en anclas, concretamente en detectores de dos etapas y detectores de una sola etapa.

Faster R-CNN: Detectores de dos etapas

Faster R-CNN es un clásico dentro de los detectores de dos etapas. Combina una *Region Proposal Network*, que genera propuestas de regiones de forma rápida, con una segunda red que clasifica y refina estas regiones en una sola arquitectura con convoluciones compartidas. Gracias a esta integración, el modelo alcanza una precisión significativamente mejor que R-CNN o Fast R-CNN, reduciendo el tiempo por imagen a aproximadamente 5 FPS con VGG-16. Aunque sigue siendo más lento que los modelos de una sola etapa, su precisión en la detección de objetos pequeños y complejos es muy superior [13].

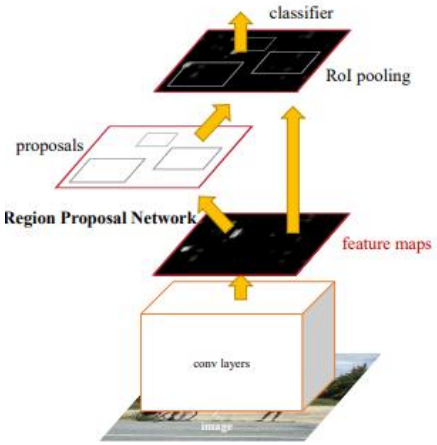


Ilustración 7. Funcionamiento de Faster R-CNN [13]

YOLO: Detectores de una sola etapa

YOLO o *You Only Look Once* es uno de los modelos más representativos de los detectores de una sola etapa. A diferencia de los métodos de dos etapas, YOLO trata la detección como un problema de regresión, prediciendo las coordenadas de los objetos y sus probabilidades de clase a partir de la imagen.

El modelo divide la imagen en una cuadrícula y, para cada celda, asigna una probabilidad a las cajas delimitadoras y las clases correspondientes. Esta estructura le permite incorporar información del contexto de la imagen y esto hace que se reduzcan los falsos positivos sobre el fondo respecto a otros modelos.

Una de sus principales ventajas es la alta velocidad de procesamiento en tiempo real. En su versión original, YOLO alcanzaba 45 FPS y su versión ligera, Fast YOLO, superaba los 150 FPS. No obstante, su principal desventaja en las primeras versiones era una menor precisión en la localización de objetos pequeños [14].

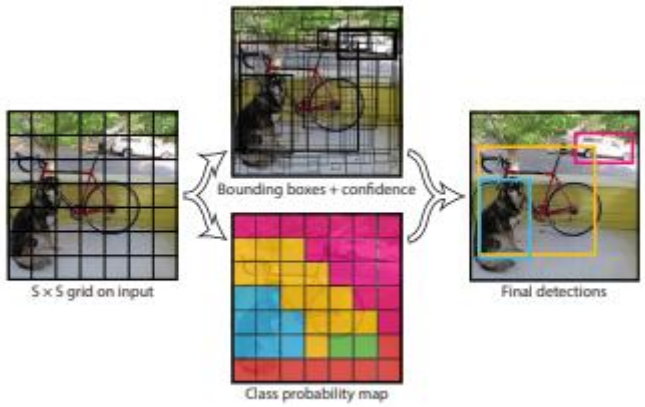


Ilustración 8. Funcionamiento de YOLO para detección de objetos [14]

Así pues, Faster R-CNN destaca cuando la precisión es lo que se buscan principalmente, mientras que YOLO resulta más adecuado cuando se prioriza la velocidad y se requiere operar en tiempo real, o incluso en hardware con recursos limitados.

2.1.3 Large Vision Models (LVM)

Los LVM son modelos multimodales que combinan imágenes y texto como entrada para realizar tareas de procesamiento de lenguaje natural con información visual. Son una evolución de los LLM, *Large Language Models*, con la capacidad añadida de ver una imagen o video.

La mayoría de los LVM se basan en transformadores de gran escala, aprovechando su capacidad para procesar muchos datos y generar representaciones fuertes en detección de objetos.

Estos modelos pueden realizar diversas tareas, entre ellas responder preguntas sobre imágenes, generar descripciones automáticas de imágenes, reconocimiento de comandos mediante instrucciones, comprensión de documentos e imágenes, e interacciones hombre-máquina, entre otras muchas tareas.

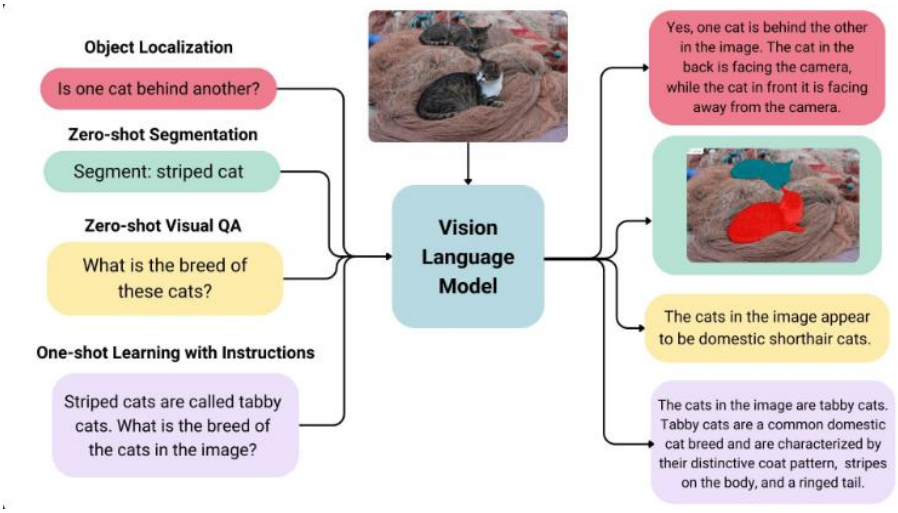


Ilustración 9. Tareas realizadas por LVMs [15]

En la imagen anterior, se pueden ver detalladas las distintas tareas que pueden realizar los LVM.

Los LVM siguen una arquitectura típica compuesta por tres bloques clave:

- **Image encoder:** Convierte la imagen en una representación vectorial. Suelen usar modelos como CLIP, ViT o ConvNeXt.
- **Proyector multimodal:** Une la representación de imagen con la de texto. Es una red neuronal que alinea ambos espacios.
- **Text decoder:** Genera texto a partir de la fusión imagen y texto. Aquí entra un modelo tipo LLM, como LLaMA, GPT o Vicuna.

Algunos modelos como LLaVA congelan el encoder y decoder y solo entrenan el proyector. Otros, como KOSMOS-2, entrenan todo el modelo de forma conjunta, pero resultan más costosos [15].

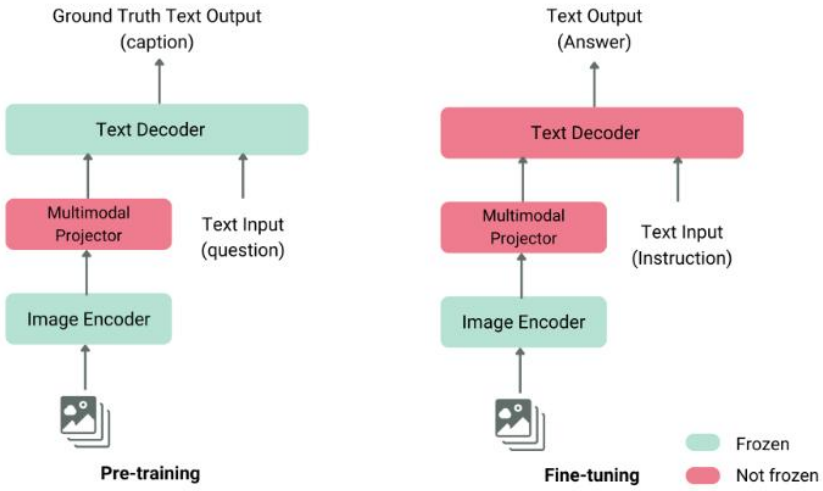


Ilustración 10. Estructura típica de un Language Vision Model [15]

En los últimos años, los *Large Vision Models* y los *Large Language Models*, especialmente en su forma multimodal, MLLM, han comenzado a desempeñar un papel emergente en la conducción autónoma. Aunque los LLM ya demuestran capacidades notables en tareas complejas de razonamiento lógico, su integración en vehículos autónomos aún se encuentra en una fase incipiente. La combinación de lenguaje natural con datos visuales y de sensores tiene el potencial de transformar los sistemas de conducción autónoma actuales. Los MLLM permiten una comprensión del entorno más precisa al fusionar texto con información visual, mejorando la percepción, la planificación de movimiento y el control. Estos modelos también abren nuevas vías para una interacción natural y personalizada entre el pasajero y el vehículo. A medida que se avanza hacia los niveles SAE 4 y 5, los LVM y MLLM prometen habilitar vehículos más adaptativos, explicables y centrados en el usuario [16].

2.1.4 Métricas de evaluación en la detección de objetos

La evaluación del rendimiento de los modelos visuales para detectar objetos es clave para comparar diferentes enfoques y medir mejoras frente a otros métodos existentes. En la tarea de la detección de objetos, los detectores no solo deben identificar la presencia de los objetos en una imagen y clasificarlos, sino también localizar con precisión su posición mediante *bounding boxes*, o cajas limitadoras. Para ello, se han establecido métricas específicas que cuantifican la exactitud en la clasificación de cada objeto y la coincidencia espacial entre las cajas predichas y las reales, a lo que se conoce como *ground-truth*. Actualmente, se ha impulsado el desarrollo de estas métricas, siendo el *Intersection over Union* (IoU) y el *mean Average Precision* (mAP) los indicadores más utilizados para evaluar la calidad de las detecciones en distintos *datasets* [17].

2.1.4.1 Intersection over unión (IoU)

Una de las métricas más importantes se le conoce como *intersection over union* o intersección sobre unión, en español. Esta medida está basada en el índice de Jaccard y cuantifica la superposición entre dos conjuntos. Se calcula como el área de intersección entre la caja predicha, declarada como B_{gt}, y la real, declarada como B_p, dividida entre el área de unión.

$$IoU = \frac{\text{Área}(\text{Bp} \cap \text{Bgt})}{\acute{A}rea(\text{Bp} \cap \text{Bgt})}$$

A continuación, se muestra una representación visua de la división del área de intersección entre el área de unión.

$$\text{IOU} = \frac{\text{area of overlap}}{\text{area of union}} =$$

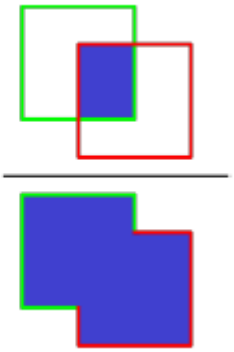


Ilustración 11. Cálculo y representación de la métrica Intersection Over Union [17]

Un valor de IoU = 1 indica una coincidencia perfecta entre la caja predicha y la real, mientras que IoU = 0 significa que no hay superposición. Cuanto más se acerque el IoU a 1, mejor se considera la detección. Dado que los detectores también clasifican cada caja dentro de las clases anotadas, solo se comparan las cajas predichas y reales de la misma clase.

Además, esta métrica permite establecer un umbral de intersección, de manera que dicho umbral determine qué detecciones se consideran correctas. Un umbral de casi 1 es muy estricto, ya que requiere que las superposiciones sean casi perfectas, mientras que uno más bajo permite

aceptar superposiciones parciales entre la caja real y la caja predicha. Es importante declarar que los umbrales de IoU se expresan en porcentaje, siendo los más comunes 50% y 75%.

Este criterio es fundamental para definir las métricas más relevantes en detección de objetos [17].

2.1.4.2 Precision y Recall

En detección de objetos y en descripción textual de imágenes, dos conceptos fundamentales para evaluar un modelo son la precisión (*precision*) y la recuperación (*recall*).

La precisión mide la capacidad del detector para identificar solo los objetos relevantes, es decir, el porcentaje de predicciones correctas entre todas las predicciones realizadas. Por otro lado, el *recall* mide la capacidad del detector para encontrar todos los objetos presentes en la imagen, también llamado *ground-truth*, es decir, el porcentaje de predicciones correctas respecto a todos los objetos reales [17].

Para calcular estas métricas, cada predicción se clasifica como:

- **True Positive (TP):** Detección correcta de un objeto real. El clasificador ha detectado como positiva una detección que realmente era positiva.
- **False Positive (FP):** Detección incorrecta de un objeto inexistente o mal localizada. El clasificador ha detectado como positivo, una clase que realmente era negativa.
- **False Negative (FN):** Objeto real no detectado. El clasificador ha detectado como negativo, una clase que en realidad era positiva.
- **True Negative (TN):** No se ha detectado el objeto porque no había objeto. El clasificador ha detectado como negativa una detección que realmente era negativa.

En la siguiente imagen se puede visualizar estos conceptos, organizados en una tabla que permite organizar las detecciones de distintos clasificadores. A esta tabla se la conoce como matriz de confusión.

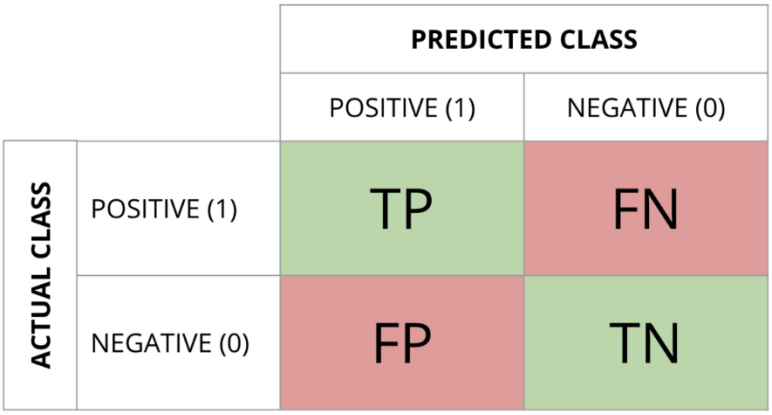


Ilustración 12. Matriz de confusión [18]

Una vez declarados estos conceptos, se puede comenzar a declarar las métricas de *precision* y *recall* formalmente. Como se ha mencionado, la precisión permite identificar los objetos correctos entre todas las predicciones, y por tanto se puede definir como:

$$Precision = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

Por otro lado, el *recall* permite medir la capacidad de encontrar objetos, y por tanto se puede definir como:

$$Recall = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

Estas métricas permiten evaluar tanto la exactitud de las predicciones como la capacidad de encontrar todos los objetos, y son la base para calcular métricas más completas como el *Average Precision* y el *mean Average Precision* [17].

2.1.4.3 Average Precision

El *Average Precision* (AP) es una métrica que mide el rendimiento de un detector al integrar la relación entre *precision* y *recall*, a lo largo de diferentes umbrales de confianza. Es importante considerar que se consideran positivas aquellas detecciones que superen ese umbral de confianza.

Al variar este umbral, el *recall* tiende a disminuir mientras que la *precision* puede variar, generando curvas que pueden presentar un comportamiento no monótono. Para evitar esto y estabilizar la curva *precision-recall*, se le aplica una interpolación que garantiza su monotonía decreciente, permitiendo calcular el área bajo la curva.

El valor de AP se calcula como el área bajo dicha curva monótona decreciente, evaluada en puntos de *recall* de referencia. En la práctica, un detector con AP alto indica que mantiene buena precisión incluso al aumentar el *recall*, mostrando un equilibrio eficiente entre exactitud de las predicciones y capacidad de encontrar todos los objetos [17].

2.1.4.4 Mean Average Precision

Teniendo en cuenta que el *Average Precision* se calcula de forma individual para cada clase, para obtener una métrica global en conjuntos de datos con múltiples clases, se utiliza el *mean Average Precision (mAP)*, que corresponde al promedio de los AP de todas las clases, y se calcula de la siguiente manera:

$$mAP = \frac{1}{\mathrm{C}} x \sum_{i=1}^{C} APi$$

donde C es el número de clases y APi es el AP de la clase i. Es decir, es una métrica que mide el rendimiento global de un detector de objetos. Se calcula a partir de los valores de AP, y como se ha mencionado, resume cómo de bien se equilibra la *precision* y el *recall* en cada clase, promediando dichos valores para obtener una medida única [17].

2.1.4.5 Average Recall

El *Average Recall* (AR) es una métrica que mide la capacidad de un detector para localizar correctamente los objetos, evaluando su rendimiento en distintos umbrales entre las cajas predichas y las de referencia.

A diferencia del AP, no tiene en cuenta el nivel de confianza de las detecciones, ya que esta métrica asume un umbral de confianza cero, y se centra únicamente en la proporción de objetos que se detectan correctamente. El cálculo del AR consiste en promediar los valores de *recall* obtenidos para múltiples valores de IoU, normalmente en el intervalo [0.5, 1].

En la práctica, *datasets* como COCO reportan el AR como el promedio del *recall máximo* alcanzado en diferentes umbrales de IoU, ofreciendo así una métrica del rendimiento del detector en cuanto a exhaustividad y precisión [17].

2.1.4.6 Mean Average Recall

Por último, conviene también incluir la definición del *Mean Average Recall*. De igual manera que el *Mean Average Precisión*, se trata de una métrica global que se calcula tomando los promedios de los valores de AR para las clases. Como el AR se calcula de forma individual para cada clase, mAR se obtiene promediando los valores de AR entre todas las clases del dataset. De esta forma, se consigue una única métrica que refleja el rendimiento medio del detector en términos de exhaustividad de la detección y localización de objetos [17].

2.2 Contexto y justificación

Los vehículos son cada vez más utilizados en todo el mundo, y conducir se ha convertido en una tarea repetitiva, peligrosa y muchas veces aburrida. En este contexto, el creciente interés en desarrollar vehículos autónomos ha potenciado la necesidad de desarrollar sistemas de percepción visual mejorados integrados en los vehículos, empleando para ello tecnologías avanzadas en visión artificial.

A pesar de los avances, la conducción autónoma se enfrente a numerosos desafíos, especialmente en escenarios complejos o entornos nuevos, donde es importante una toma de decisiones rápida, efectiva y en tiempo real. Dentro de este marco, los *Large Vision Models* se presentan como una de las soluciones, debido a su alta capacidad de generalización en tareas visuales.

Este Trabajo Fin de Máster se justifica en la necesidad de explorar, evaluar y adaptar los LVM al dominio de la conducción autónoma, con el objetivo de analizar sus ventajas y sus limitaciones, y así obtener conclusiones acerca de la aplicabilidad en entornos reales. Por tanto, el propósito de este Trabajo Fin de Máster es evaluar estos modelos visuales y aportar conocimiento útil sobre su funcionamiento y potencial, contribuyendo a la mejora global de los sistemas de percepción visual, tanto en tareas de detección de objetos como en la comprensión de la imagen.

2.3 Planteamiento del problema

Actualmente, se han logrado avances en los sistemas de conducción autónoma. Sin embargo, existen muchas limitaciones que dificultan su integración actual en entornos reales. El estado del arte ha demostrado que los modelos tradicionales son eficientes en entornos conocidos, pero presentan dificultades generalizando los resultados en entornos no vistos. En este sentido, los *Large Vision Models* emergen como una alternativa prometedora para la visión artificial tradicional ya que han sido entrenados con una gran cantidad de datos, y por tanto tienen una capacidad de generalización mucho mayor.

Estos modelos también presentan limitaciones. Entre esas limitaciones se encuentran el elevado coste computacional asociado, la necesidad de entrenarlos con grandes conjuntos de datos, la dificultad de integrarlos en un hardware limitado, y sobre todo, el tiempo de inferencia que presentan, que son muy elevados en comparación con modelos tradicionales [19].

Esta situación evidencia, por un lado, la falta de estudios que evalúen la efectividad de los LVM, y por otro, la necesidad de adaptar estos modelos a los requisitos que demanda la conducción autónoma, donde son críticas la precisión, la eficiencia y la capacidad de respuesta en tiempo real.

Capítulo 3. OBJETIVOS

En este capítulo se presentan el objetivo general y los objetivos específicos que guían el desarrollo de este Trabajo Fin de Máster. Asimismo, durante el capítulo se detallan los beneficios esperados y la posible contribución del proyecto al ámbito de la conducción autónoma.

3.1 Objetivos generales

El objetivo general de este trabajo es explorar las capacidades de los modelos visuales de gran escala (LVMs) en tareas relacionadas con la percepción visual del entorno de los vehículos autónomos, con el fin de analizar sus limitaciones técnicas y su viabilidad práctica en entornos reales.

3.2 Objetivos específicos

A partir del objetivo general planteado, se definen los siguientes objetivos específicos que permiten orientar el proyecto, y definen los límites del alcance de este Trabajo Fin de Máster.

1. **Estudio profundo de las arquitecturas de LVMs:**
    - Analizar los principales modelos de visión artificial más utilizados actualmente y situar los LVM dentro de ese contexto.
    - Examinar los componentes internos de arquitecturas representativas de LVMs, así como algunos modelos LVM principales.
    - Evaluar las métricas de calidad utilizadas para medir el desempeño de los modelos.
2. **Comparación de LVMs con enfoques tradicionales de visión artificial:**
    - Realizar una revisión comparativa entre un modelo LVM y un modelo de visión artificial más tradicional basado en redes convolucionales, concretamente YOLOv11.
    - Analizar las diferencias en precisión de detección, tiempos de inferencia, velocidades de procesamiento y capacidad de generalización.
    - Evaluar los requerimientos hardware y las limitaciones de inferencia en ambos enfoques.
3. **Analizar el rendimiento de distintos modelos LVMs:**
    - Implementar un conjunto de datos con imágenes anotadas, que permitan analizar distintos LVM en igualdad de condiciones.
    - Implementar y probar distintos modelos LVM para la tarea de la detección de objetos.
    - Implementar y probar distintos modelos LVM para la tarea de descripción semántica de imágenes.
    - Obtener resultados comparativos en métricas cuantitativas como tiempo de inferencia, FPS, número de detecciones, AP y AR.
4. **Evaluación de limitaciones prácticas de los LVM en la conducción autónoma:**

- Analizar los resultados obtenidos para la tarea de detección de objetos e identificar sus limitaciones y fortalezas.
- Analizar los resultados obtenidos para la tarea de descripción semántica de imágenes e identificar sus limitaciones y fortalezas.
- Extraer conclusiones sobre la aplicabilidad de los modelos LVM en entornos de conducción autónoma.

5. **Identificación de retos a los que se enfrentan los LVMs y detectar posibles futuras líneas de investigación:**
   - Identificar los principales desafíos que se encuentran en el uso de LVMs en la conducción autónoma.
   - Proponer posibles líneas futuras de mejora, como modelos híbridos, técnicas de compresión, o *fine-tunning* adaptado a entornos de conducción.
   - Establecer un marco base para futuras investigaciones que busquen aplicar LVMs en sistemas autónomos de cualquier clase.

3.3 Beneficios del proyecto

Este Trabajo Fin de Máster aporta beneficios tanto a nivel científico como a nivel técnico, siguiendo los objetivos específicos establecidos. En primer lugar, este proyecto contribuye a mejorar la comprensión del potencial de los LVMs, pero también de sus limitaciones, sobre todo en el contexto de la conducción autónoma, un ámbito de aplicación bastante exigente que requiere soluciones precisas, eficientes, rápidas y escalables en tiempo real.

La investigación y el análisis detallado de las arquitecturas, los principales modelos, rendimientos y limitaciones proporciona una base sólida de conocimiento para otros investigadores que trabajan en sistemas autónomos aplicando visión artificial. Además, la comparación de los modelos estudiados con modelos tradicionales permite identificar ventajas competitivas y posibles áreas de mejora para los LVMs, como latencia, precisión, o coste computacional.

En conjunto, el proyecto acelera el proceso de estudio e investigación de estas tecnologías emergentes en un dominio tan relevante como es la conducción autónoma, fomentado el uso informado y eficiente de los *Large Vision Models*.

Capítulo 4. DESARROLLO DEL PROYECTO

En este capítulo se presenta el proceso completo de desarrollo del proyecto, desde su planificación inicial hasta la obtención de los resultados finales. Se detallan las distintas fases que han compuesto el trabajo, organizadas en un diagrama de Gantt, junto con la metodología y las herramientas utilizadas. Asimismo, se abordan aspectos relevantes como los recursos empleados, la evaluación económica del proyecto y su viabilidad. Finalmente, se exponen los principales resultados obtenidos.

4.1 Planificación del proyecto

En esta sección se recoge la planificación real del proyecto, incluyendo un cronograma de Gantt con las diferentes actividades realizadas a lo largo del tiempo. Se describen de manera breve las tareas llevadas a cabo, su duración estimada y el esfuerzo invertido en cada una de ellas.

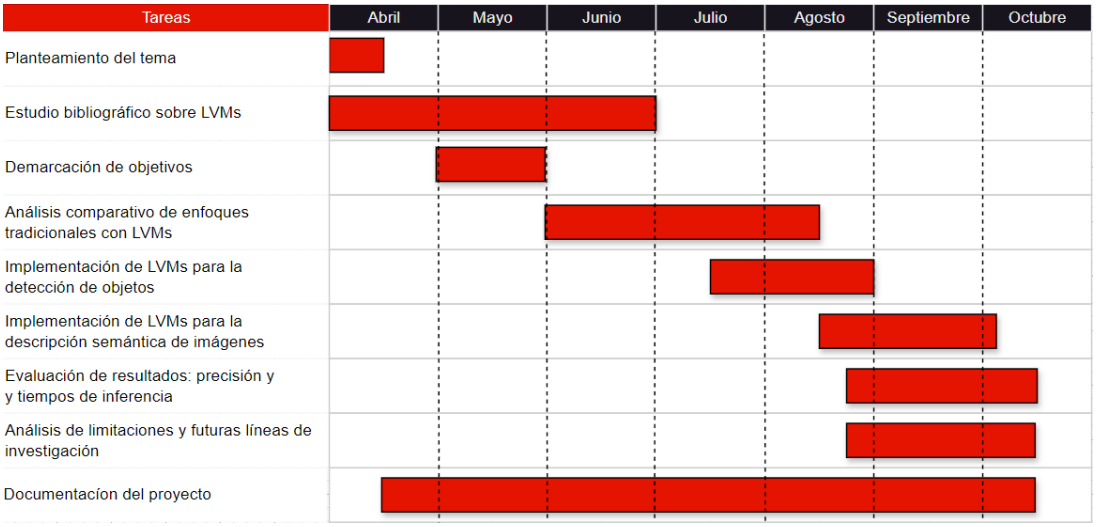


Ilustración 13. Diagrama de Gantt con la planificación del proyecto

Como se puede ver en la ilustración, el proyecto se divide en 9 tareas, repartidas entre los meses de abril y octubre.

4.2 Descripción de la solución, metodologías y herramientas empleadas

En esta sección se detalla la metodología seguida durante el desarrollo de este Trabajo Fin de Máster. Aquí se explica todo el desarrollo del proyecto, incluyendo el enfoque metodológico seguido, la configuración del entorno de estudio, la comparativa con enfoques tradicionales, la implementación de distintos modelos LVM para la detección de objetos y descripción semántica de imágenes, y un breve apartado analizando el uso de la inteligencia artificial.

4.2.1 Enfoque metodológico

Este proyecto ha seguido una metodología exploratoria, comparativa y experimental, y se estructura en distintas fases:

1. **Revisión del estado del arte y análisis teórico:** Se inició el proyecto con una revisión exhaustiva de literatura científica, artículos técnicos y documentación oficial de algunos de los modelos más representativos de los LVMs. Esta revisión incluyó tanto los fundamentos teóricos como los aspectos prácticos relacionados con su aplicación en tareas de percepción visual.
   Además, ha sido importante el estudio del contexto de la visión artificial para situar los modelos dentro de este ámbito.

2. **Comparativa entre enfoques tradicionales y LVMs:** Se seleccionaron modelos clásicos de visión por computador basados en CNNs como línea base comparativa, concretamente YOLOv11. Estos modelos se han obtenido de la librería Ultralytics. Se implementaron y evaluaron en tareas de detección para establecer diferencias claras respecto a precisión, latencia y eficiencia con un modelo LVM muy utilizado actualmente.

3. **Implementación y pruebas de LVMs para tareas de visión:** Primero, se integró un conjunto de datos con anotaciones COCO para poder utilizar y probar los modelos LVM. Posteriormente, se integraron modelos preentrenados disponibles en plataformas como *Hugging Face*. Estos modelos han sido seleccionados en función de las tareas que se iban a realizar durante las pruebas, concretamente:
   - Detección de objetos
   - Descripción semántica de imágenes

4. **Evaluación y análisis de resultados:** Se recopilaron métricas clave como *precision*, *recall*, FPS, memoria usada y latencia, entre otras muchas métricas. Se aplicaron técnicas de visualización y análisis estadístico para mostrar gráficas comparativas con los resultados obtenidos. En base a la tarea realizada por los modelos, las métricas analizadas han presentado diferencias en la detección de objetos y en la descripción semántica de imágenes.

5. **Identificación de limitaciones y propuestas de mejora:** A partir de la evaluación experimental, se discutieron los principales retos técnicos que presentan los LVMs. Finalmente, se propusieron líneas futuras de investigación orientadas a mejorar su viabilidad en entornos autónomos.

Esta metodología ha permitido no solo estudiar el rendimiento técnico de los modelos, sino también entender su aplicabilidad real, sus restricciones y sus oportunidades de mejora en sistemas complejos como los vehículos autónomos.

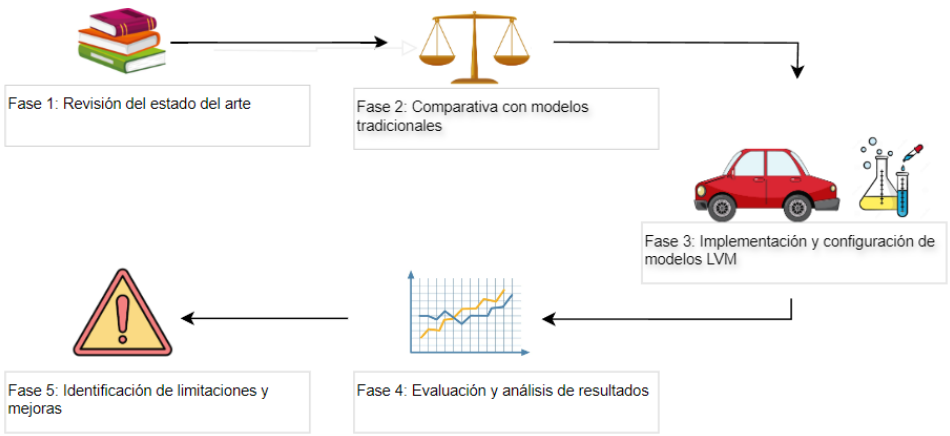


Ilustración 14. Diagrama visual con la metodología seguida

4.2.2 Configuración del entorno de estudio

Antes de proceder con el estudio exhaustivo de los modelos, es necesario definir claramente las tareas que los modelos deberán abordar, los tipos de modelos que serán comparados en función de esas tareas, las fuentes de datos utilizadas para su evaluación y la configuración del entorno donde se realizarán todas las pruebas.

Tareas por realizar:

Las principales tareas consideradas para la evaluación de los modelos incluyen la detección y clasificación de objetos en imágenes. A estas tareas se les denomina *zero-shot object detection.* Además, también se evaluarán tareas de descripción semántica de imágenes, de manera que los modelos deben proporcionar texto a partir de una imagen de entrada y un *prompt* de texto. Las tareas se centrarán únicamente en vehículos, excluyendo señales de tráfico y peatones.

Modelos por comparar:

En función de la tarea realizada, se seleccionarán unos modelos u otros. Para el análisis comparativo con enfoques tradicionales, se evaluará el rendimiento de modelos detectando objetos. En este contexto, se evaluarán los modelos YOLOv11 y GroundingDINO.

En el estudio e implementación de distintos modelos, se abordarán dos tareas distintas, como se ha mencionado anteriormente.

- Para la tarea de detectar objetos, se compararán modelos basados en arquitecturas modernas de *Large Vision Models*, tales como GroundingDINO, OWLv2 y OmDet-Turbo.
- Para la tarea de la descripción semántica de imágenes se evaluarán modelos basados en arquitecturas *image-text-to-image*, que utilizan un *encoder visual* y un LLM para devolver una descripción de la imagen. Estos modelos serán LLaVAv1.6 y Gemma-3.

4.2.2.1 Preparación del entorno

Una vez definidos los objetivos y los modelos a evaluar, se procede a la configuración del entorno de desarrollo.

El trabajo se desarrollará en Python 3.10, junto con *frameworks* especializados en aprendizaje profundo como PyTorch o TensorFlow. Además, para la fase de implementación y experimentación, se utilizará Jupyter Notebooks, que permite una ejecución interactiva y visualización directa desde un enfoque modular.

Para complementar, se instalarán bibliotecas orientadas a la visión por computador, incluyendo transformers, torchvision, datasets, matplotlib y numpy, entre otras muchas.

La instalación de estas dependencias se realizará mediante el gestor de paquetes pip, usando el proceso de conjuración seguido en la siguiente imagen:

```
PS C:\Users\Tito\Desktop\Master\TFM\development> py --version
Python 3.13.2
PS C:\Users\Tito\Desktop\Master\TFM\development> py -m pip install torch torchvision transformers datasets matplotlib
```

Ilustración 15. Preparación del entorno en Python

4.2.2.2 Obtención de datos

Para probar y comparar los modelos, es imprescindible contar con conjuntos de datos adecuados. Por ello, se seleccionarán imágenes de diferentes repositorios públicos disponibles en Kaggle, los cuales contienen miles de imagenes de carreteras, vehículos y señales de tráfico.

Como ejemplo, uno de los repositorios principales empleados será el *Road Vehicle Images Dataset* [20], que proporciona una amplia variedad de imágenes útiles para las tareas de detección planteadas.

Este *dataset* contiene numerosas imágenes capturadas en carreteras de Bangladesh, y está específicamente orientado para ser utilizado con modelos de detección como YOLO. Las imágenes vienen separadas en un conjunto de entrenamiento y otro de validación. Además, cada imagen viene acompañada de un archivo de anotaciones en formato YOLO, que incluye las coordenadas normalizadas de los objetos detectados y sus respectivas clases. El conjunto contempla un total de 21 clases, concretamente:

['ambulance', 'army vehicle', 'auto rickshaw', 'bicycle', 'bus', 'car', 'garbagevan', 'human hauler', 'minibus', 'minivan', 'motorbike', 'pickup', 'policecar', 'rickshaw', 'scooter', 'suv', 'taxi', 'three wheelers -CNG-', 'truck', 'van', 'wheelbarrow']

Gracias a la variedad de vehículos que dispone el *dataset*, resulta muy valioso para evaluar la capacidad de generalización de los modelos, así como la precisión en entornos urbanos y realistas. La disponibilidad de anotaciones en el formato requerido permite su reutilización tanto para tareas de detección de objetos como para segmentación y clasificación supervisada.

Un ejemplo de las imágenes que contiene este *dataset* se puede ver en la siguiente imagen:

Ilustración 16. imagen perteneciente al primer dataset, representando un autobús en una carretera.

Cómo se ha mencionado, junto con cada imagen se proporciona un fichero de texto que contiene información sobre los objetos en la imagen. En el fichero, cada fila representa un objeto y contiene el identificador de la clase correspondiente al coche, la posición en X y en Y, el ancho en pixeles y el alto en pixeles. La estructura mencionada es la siguiente:

<class_id> <x_center> <y_center> <width> <height>

Un ejemplo de esta anotación se puede ver en la siguiente imagen, donde se indica la clase de vehículo como 4, correspondiente a un autobús en este caso.

```
1    4 0.521875 0.4929906542056075 0.121875 0.21495327102803738
```

Ilustración 17. Fichero ejemplo con las etiquetas correspondientes a los objetos encontrados

A partir de esta información, es posible visualizar los bordes o *bounding boxes* de los objetos detectados. Para ello, se ha desarrollado un pequeño script en Python utilizando la librería cv2 de OpenCV, que permite superponer las cajas sobre las imágenes originales. El resultado se muestra a continuación:

Ilustración 18. Visualización de objeto detectado, con bordes delimitadores y detección de clase

Además de este *dataset*, se ha decido utilizar el *dataset* COCO, o *Common Objects in Context*, concretamente el conjunto de datos del 2017. Este conjunto de datos dispone de un conjunto de entrenamiento, validación y prueba. Para la tarea de este Trabajo Fin de Máster, se utilizará el de validación, ya que sus anotaciones son públicas y es el estándar para la evaluación de los distintos modelos. Se trata de un conjunto de datos a gran escala utilizado para segmentación y detección de objetos principalmente, aunque se puede usar para muchas tareas [21].

Este *dataset* es muy interesante, ya que no solo contiene imágenes de carreteras y vehículos sino de muchísimas más cosas como animales, personas, objetos cotidianos como sillas, etc. El uso de COCO Dataset permite la comparación directa de las métricas de rendimiento de los distintos modelos para que sea justa y universal.

4.2.3 Comparativa con enfoques tradicionales

4.2.3.1 Implementación de GroundingDINO para detección de objetos

Para evaluar LVMs en tareas de detección de objetos, se deben implementar aquellos modelos que estén pensados para detectar objetos, y no modelos pensados para responder preguntas. Para realizar estas tareas, se ha decidido implementar el modelo GroundingDINO, un detector de objetos *zero-shot* con alto desempeño en mAP, capaz de generar etiquetas automáticamente. Aunque su velocidad de procesamiento es menor que la de YOLO y no es adecuado para escenarios de tiempo real, las etiquetas generadas pueden usarse para entrenar modelos como YOLO, manteniendo o incluso mejorando la precisión de detección [22].

Una vez implementado el modelo, se ha procedido a probarlo utilizando la misma imagen utilizada con los modelos anteriores. El resultado obtenido por el modelo es el siguiente:

Ilustración 19. Salida del modelo Grounding DINO para autobus

Para probar su robustez, se ha decido ir un paso más allá y probar imágenes con más vehículos y distintas condiciones meteorológicas. El resultado obtenido se puede ver en la siguiente imagen:

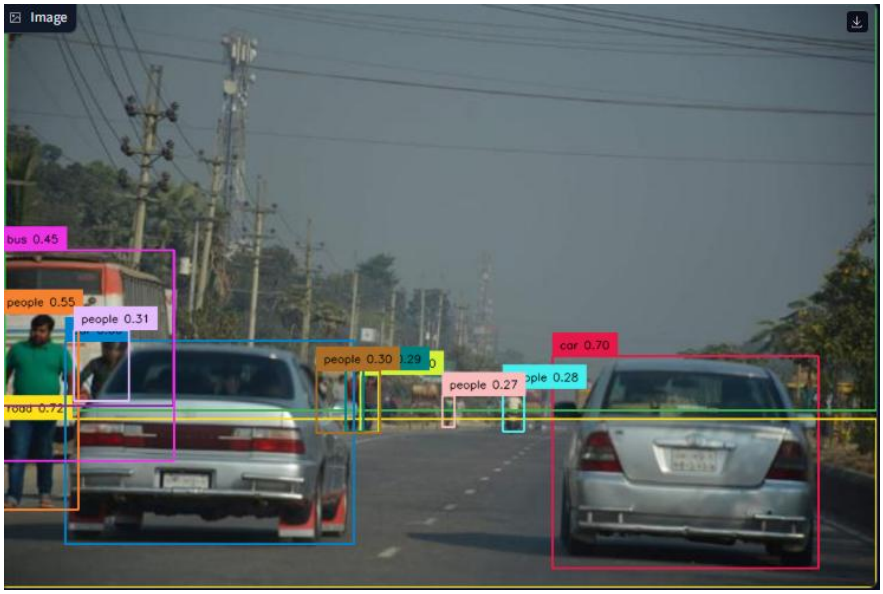


Ilustración 20. Salida del modelo GroundingDINO para imagen compleja

Cómo se puede observar en la imagen anterior, el modelo GroundingDINO ha sido capaz de detectar los coches de la imagen, los buses, las personas, la carretera e incluso el cielo.

Una particularidad de este modelo es que se puede combinar con modelos de segmentación como Grounding-SAM, que permite detectar y segmentar los objetos [23]. Para realizar una prueba, se ha utilizado otra imagen que contiene dos vehículos en una carretera rodeada de árboles. El resultado se puede observar en la siguiente imagen:

Ilustración 21. Salida del modelo Grounding-SAM para detección de objetos

En la imagen se puede ver no solo la detección de los objetos mencionados, sino que también se puede ver la segmentación de dichos objetos en la propia imagen. El modelo proporciona una herramienta para ver únicamente la segmentación de los objetos detectados, y se puede ver en la siguiente imagen:

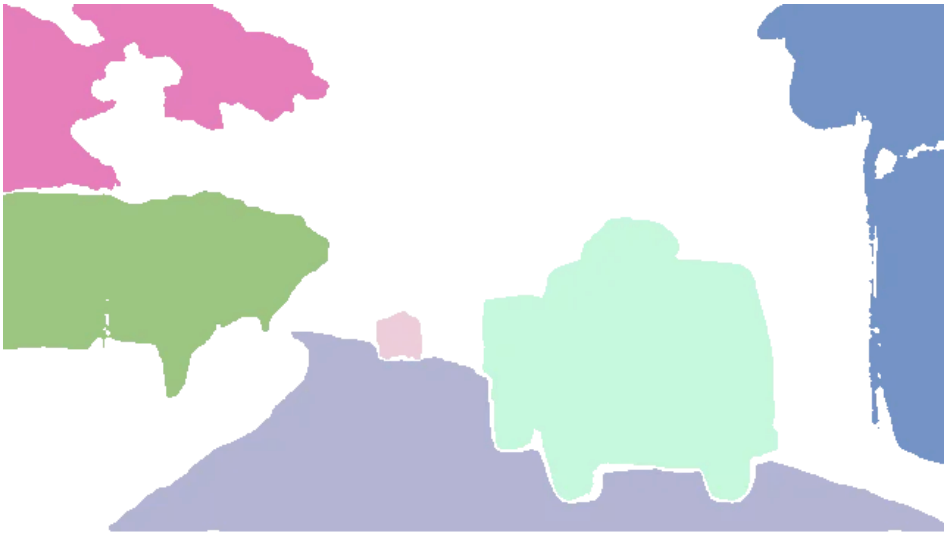


Ilustración 22. Salida del modelo Grounding-SAM para segmentación de objetos

4.2.3.2 Implementación del modelo clásico YOLO

En este apartado, se va a detallar la implementación del modelo YOLO para poder realizar una comparación con el modelo LVM implementado.

Inicialmente, se descargó un modelo preentrenado de la familia YOLO, concretamente la versión YOLO11n, que ofrece un buen equilibrio entre precisión y velocidad [24]. Para ello, se utilizó la librería oficial ultralytics para cargar el modelo.

Una vez implementado el modelo, se probó con una imagen ya utilizada en este Trabajo Fin de Máster. El resultado fue satisfactorio, pues se puede ver que la tarea de detección de vehículos se realizó exitosamente, y la tarea de clasificación de dicho objeto entre las 21 clases del *dataset* también, con una certeza de acierto del 91%. Se puede ver dicho resultado en la siguiente imagen:

Ilustración 23. Salida del modelo YOLOv11 para imagen de ejemplo

4.2.3.3 Comparativa de YOLO con LVM

Comparativa teórica

Antes de comparar el modelo tradicional YOLO con el modelo LVM, concretamente GroundingDINO, conviene entender ciertas diferencias teóricas que presentan estos modelos.

- **Arquitectura**

Los modelos tradicionales como YOLO parten de la arquitectura clásica YOLO basada en CNNs, y están optimizados para velocidad y eficiencia. Por otro lado, los LVMs están basados en transformadores multimodales que han sido preentrenados con pares texto-imagen. Estos modelos están diseñados para obtener una gran capacidad de generalización y flexibilidad, frente a la velocidad, que es más limitada [25].

- **Capacidad de Vocabulario**

Los modelos YOLO, sobre todo los más tradicionales, presentan un vocabulario muy cerrado, mientras que modelos YOLO más nuevos presentan un vocabulario abierto. Esto quiere decir que solo detectan las clases vistas durante el entrenamiento. Por el contrario, los LVMs han sido entrenados con visión-lenguaje a gran escala, y por tanto permiten detectar categorías que nunca ha visto el modelo a través de descripciones textuales, es decir, detección zero-shot [25].

- **Robustez**

El modelo YOLO, a pesar de ser eficiente, suele presentar menores valores mAP a pesar de su velocidad, mientras que LVMs como GroundingDINO presentan valores mAP muy altos [26]. Cabe recordar que la métrica mAP, también denominada Precisión Media Promedio, es una de las métricas más utilizadas para valorar modelos de visión artificial y que mide de manera global qué tan bien un modelo detecta y clasifica objetos en imágenes [25].

- **Eficiencia**

Los modelos YOLO mantienen una filosofía de eficiencia y efectividad en tiempo real, mientras que los LVMs requieren más cómputo y memoria y suelen ser menos prácticos en escenarios de tiempo real [26].

Comparativa práctica

A la hora de comparar los modelos, es importante tener en cuenta tanto sus diferencias teóricas como sus aspectos prácticos y mediables cuantitativamente. Los aspectos cuantitativos que se van a comparar en este apartado son: mAP, tiempo de inferencia por imagen, tamaño del modelo y uso de memoria.

- **mAP**

El mAP es la métrica más utilizada y completa para evaluar el rendimiento de los modelos de detección de objetos. Para este apartado, basta con entender que el mAP es una métrica que permite analizar cuán preciso es un modelo para así poder comparar la efectividad de sus predicciones. Es como la nota final que resume la calidad de un modelo, ya que tiene en cuenta tanto la precisión de las cajas delimitadoras como la confianza en las clasificaciones.

Para obtener las métricas de rendimiento y realizar una comparación cuantitativa entre YOLO y GroundingDINO, se ha seguido una metodología de dos fases:

1. **Fase 1: Generación de predicciones.** Se han desarrollado dos scripts independientes en Python, uno para cada modelo. El objetivo de estos scripts es procesar cada una de las 5,000 imágenes del conjunto de validación de COCO y generar un archivo de predicciones en formato JSON.
    - **Para el modelo YOLO:** Se ha cargado el modelo preentrenado YOLO11n mediante la librería ultralytics. Posteriormente, se cargan las imágenes de validación, y para cada una, se ha realizado la inferencia, se han convertido las coordenadas al formato correspondiente, y todo ello se ha almacenado en un archivo JSON compatible con el formato COCO.
    - **Para el modelo GroundingDINO:** A diferencia de YOLO, que está entrenado para clases específicas, GroundingDINO es un modelo de detección *zero-shot*. Se ha cargado el modelo GroundingDINO-tiny y su *processor*, y se han definido los nombres de las categorías de COCO como etiquetas de texto que el modelo intentará reconocer. Después, para cada imagen, se preprocesa y se combina

con las etiquetas, se pasa por el modelo, y se obtienen la caja y los *scores*, que se almacenan en un diccionario para generar un JSON compatible con COCO.

2. **Fase 2: Evaluación de rendimiento.** Una vez obtenidos los dos archivos JSON con las predicciones en el formato de COCO, se ha utilizado un tercer script de Python para evaluar el rendimiento. Este script emplea la biblioteca oficial de COCO API, pycocotools, que está diseñada específicamente para este tipo de evaluación.

   La API compara las predicciones de cada archivo JSON con las anotaciones de verdad del conjunto de datos COCO.

   El resultado de esta comparación es un conjunto de métricas de rendimiento, entre ellas AP y AR, que son fundamentales para una evaluación completa.

En las siguientes figuras se presentan los resultados de detección de objetos obtenidos con los modelos detallados.

```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.330
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.441
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.363
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.130
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.360
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.517
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.271
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.380
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.383
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.145
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.415
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.602
```

Ilustración 24. Resultados de YOLO en la detección de objetos

```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.425
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.560
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.462
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.273
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.449
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.571
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.336
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.503
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.519
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.344
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.544
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.672
```

Ilustración 25. Resultados de GroundingDINO en la detección de objetos

La primera figura muestra los resultados del modelo YOLOv11, mientras que la segunda figura corresponde al modelo GroundingDINO en su versión reducida. Se pueden observar diferencias en la precisión de las cajas delimitadoras y en la detección de objetos pequeños y medianos, donde GroundingDINO tiende a mostrar unos mejores resultados.

A continuación, se presenta una tabla comparativa de las métricas de evaluación para ambos modelos, que permite cuantificar sus diferencias y obtener conclusiones comparando los resultados de cada métrica.

Métrica	GroundingDINO	YOLOv11	Observaciones
AP @[IoU=0.50:0.95] (All)	0.425	0.330	GroundingDINO tiene un AP superior en todas las detecciones combinadas, lo que indica mejor precisión promedio global.
AP @[IoU=0.50] (All)	0.560	0.441	GroundingDINO detecta más objetos correctamente cuando se permite una tolerancia del 50% IoU.
AP @[IoU=0.75] (All)	0.462	0.363	GroundingDINO mantiene mejor precisión cuando se requiere un ajuste más preciso de las cajas.
AP Small	0.273	0.130	GroundingDINO detecta mucho mejor los objetos pequeños, casi el doble que YOLO.
AP Medium	0.449	0.360	GroundingDINO detecta mucho mejor los objetos medianos.
AP Large	0.571	0.517	GroundingDINO sigue ligeramente por delante detectando objetos grandes.
AR MaxDets=100	0.519	0.383	La capacidad de recordar (*recall*) objetos es significativamente mayor en GroundingDINO.
AR Small	0.344	0.145	GroundingDINO encuentra muchos más objetos pequeños que YOLO.
AR Medium	0.544	0.415	GroundingDino presenta mejor recuperación de objetos medianos.
AR Large	0.672	0.602	GroundingDino presenta mejor recuperación de objetos grandes, aunque la diferencia es menor.

Tabla 2. Resultados comparativos entre YOLOv11 y GroundingDINO

Analizando la tabla, se puede observar que GroundingDINO supera a YOLOv11 en todas las métricas de *precision* y *recall*. Las diferencias más notables se encuentran en la detección de

objetos pequeños, donde GroundingDINO alcanza casi el doble de AP y AR en comparación con YOLO.

Para poder visualizar los resultados, se van a representar en una gráfica las principales métricas para representar el AP de un modelo.

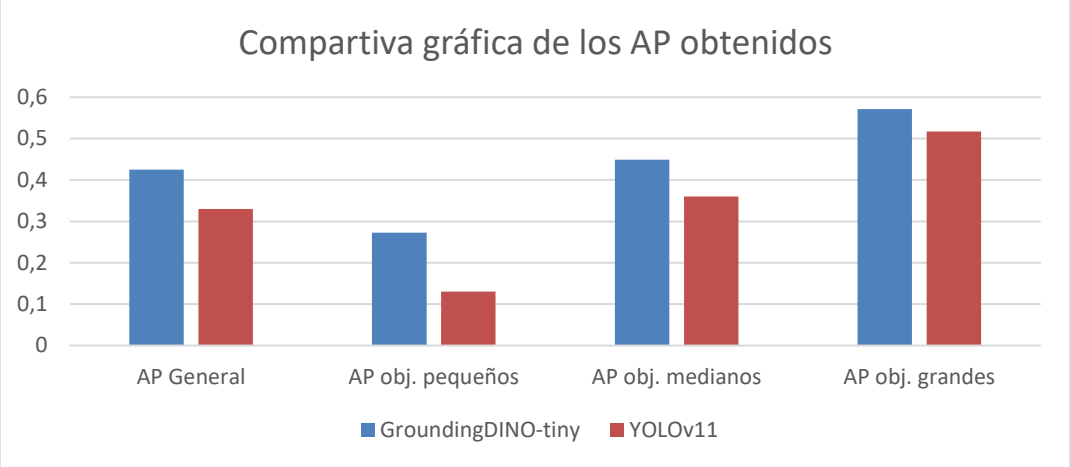


Ilustración 26. Comparativa de los principales AP obtenidos por DINO y YOLO

Como se puede ver en la gráfica, la barra azul, representando los AP obtenidos por el modelo GroundingDINO-tiny, es siempre mayor a la de YOLO, indicando que el primer modelo es más preciso.

A continuación, se puede visualizar una gráfica representando los principales AR obtenidos por los modelos.

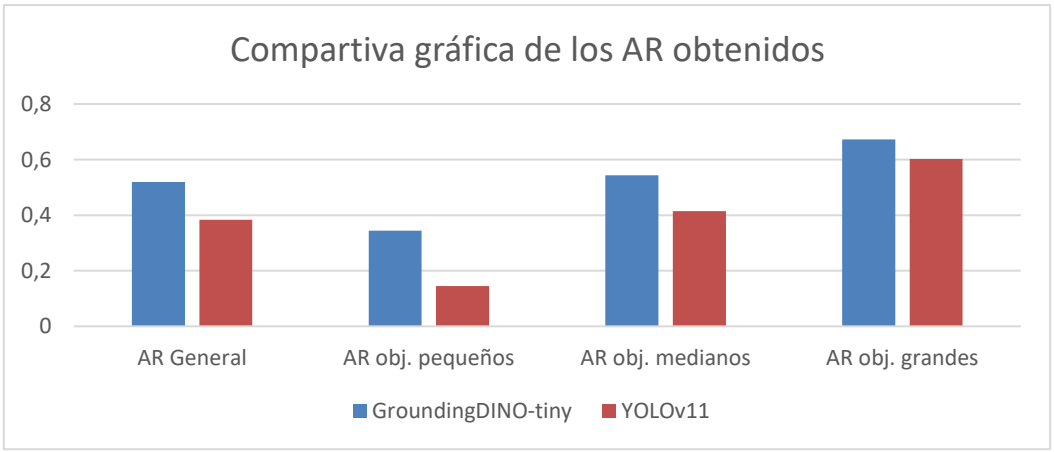


Ilustración 27. Comparativa de los principales AR obtenidos por DINO y YOLO

En esta gráfica también se puede ver que los AR obtenidos, es decir, la capacidad para recuperar objetos también es mayor para todos los objetos detectados en DINO, indicando que tiene mayor *recall* en general.

Al examinar los resultados cuantitativos y la posterior tarea de inferencia, se puede determinar que ambos modelos presentan ventajas y desventajas. GroundingDINO muestra mayor robustez

en la detección de objetos, especialmente en casos difíciles y objetos pequeños, y mantiene un buen equilibrio entre *precision* y *recall*.

Por su parte, YOLOv11 destaca por su velocidad de inferencia, mucho más rápida que la de DINO. A pesar de obtener resultados inferiores en general, YOLOv11 presenta un buen desempeño en la detección de objetos grandes y medianos.

- **Tiempo de inferencia**

Para calcular los tiempos, se ha seguido un proceso secuencial seguido de un proceso iterativo. Primero, se ha cargado el modelo, y se le ha pasado la imagen para preparar la GPU. Posteriormente, se realizan 100 iteraciones del modelo con la misma imagen para medir tanto el tiempo de inferencia total, como los FPS o *Frames Per Second*, y que permite dar una idea de la velocidad de procesado general.

La medición se ha realizado con una imagen de 640 píxeles de ancho x 448 píxeles de alto y con 3 canales de color: rojo, verde y azul.

Para el modelo YOLO, se ha obtenido un tiempo medio por imagen de 0.0286s, o lo que es lo mismo 28.63ms, y unos 34.92 FPS, que significa que se han procesado esa cantidad de imágenes o *frames* cada segundo.

Para el caso de GroundingDINO, se ha obtenido un tiempo medio por imagen bastante mayor, concretamente 0.436s y unos 2.29 FPS, que significa que se han procesado dos imágenes por segundo.

En la siguiente tabla se puede ver la comparación de los tiempos de inferencia, así como los FPS logrados con cada uno de los modelos:

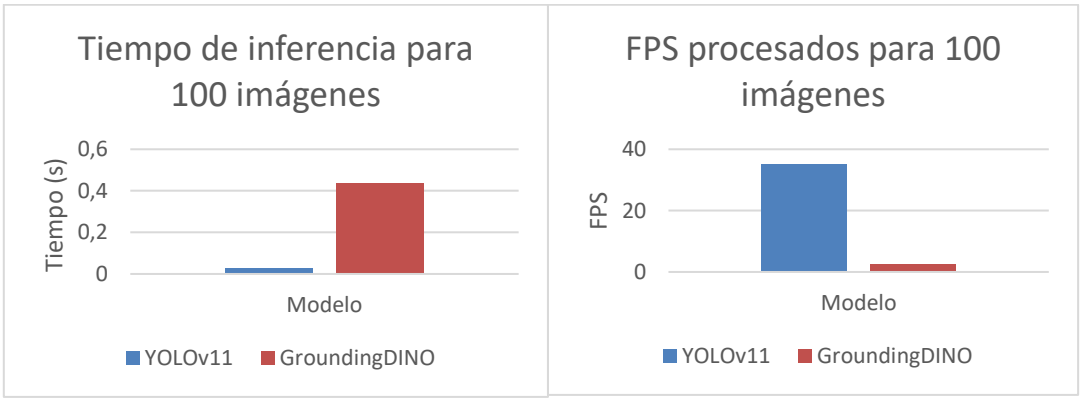


Ilustración 28. Gráficas comparativas de tiempos de inferencias y FPS

Esto verifica la teoría de que los modelos LVM son generalmente más lentos que los modelos tradicionales basados en CNNs como YOLO, que son más utilizados para detección de objetos en tiempo real.

- **Tamaño del modelo**

El tamaño del modelo es otro parámetro importante porque permite comparar y evaluar la viabilidad del despliegue del modelo, y si es compatible con dispositivos con memoria limitada o para tiempo real, como puede ser un vehículo.

El tamaño del modelo YOLOv11 es de 5.35MB, un tamaño bastante reducido y que se puede almacenar en un vehículo, mientras que el modelo GroundingDINO, en su versión reducida, tiene un tamaño en disco de 657MB, un tamaño ya considerable para un modelo, y que supone un almacenaje mucho mayor que el modelo YOLO, a pesar de ser un modelo también almacenable en un vehículo.

Además, también se ha decido considerar el número de parámetros a la hora de determinar el tamaño del modelo, para cuantificar cuánto espacio físico requieren estos modelos.

Para el caso de YOLOv11, cuenta con 2.624.000 parámetros, que es un número bastante elevado. Sin embargo, el modelo GroundingDINO en su versión reducida cuenta con 172.249.090 parámetros, que son 65 veces más que el primer modelo. Esta diferencia tanto en el número de parámetros como en el propio tamaño arroja una idea de la robustez y solidez del modelo LVM frente a YOLO.

A continuación, se muestran unas gráficas comparativas de las métricas analizadas.

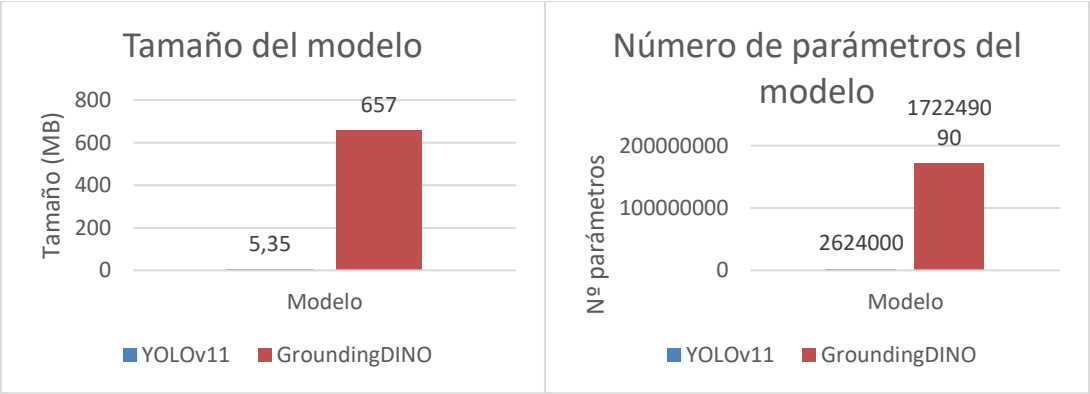


Ilustración 29. Gráficas comparativas del tamaño de los modelos y número de parámetros

- **Uso de memoria**

Además del tamaño en disco y el número de parámetros, otro aspecto clave a la hora de estudiar la viabilidad de un modelo y su aplicabilidad en entornos reales es el uso de memoria durante la inferencia, concretamente en la GPU, que es donde se suelen ejecutar estas tareas dentro de los dispositivos. Esta métrica permite conocer cuánta VRAM (memoria de vídeo) consumen los modelos al procesar imágenes.

Para obtener esta información, se ha desarrollado un script en Python que ejecuta una inferencia sobre una imagen de entrada y registra el pico de memoria utilizado por la GPU durante el proceso.

El modelo YOLOv11 ha utilizado 61,34 MB de memoria GPU en su momento pico, mientras que el modelo GroundingDINO ha alcanzado los 1515 MB, una cifra significativamente más alta.

A continuación, se muestra una gráfica comparativa de la métrica de memoria GPU consumida en su momento pico.

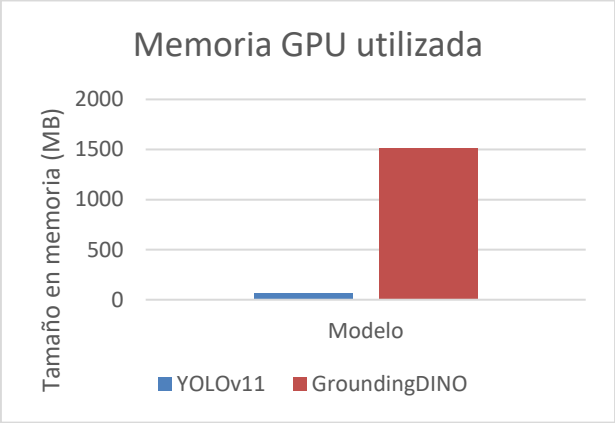


Ilustración 30. Gráfica comparativa de la memoria GPU utilizada

A través de esta comparación, se ha podido analizar y concluir que los modelos tradicionales como YOLO son modelos mucho más pequeños, rápidos y portables que los *Large Vision Models*. Sin embargo, presentan peores prestaciones en términos de precisión, debido a su menor robustez y solidez en su arquitectura que los modelos LVM.

4.2.4 Estudio de distintos modelos LVM

Entre los *Large Vision Models* existen numerosas tareas a ejecutar posibles, y dependiendo de la tarea que se desee realizar, se seleccionará un modelo u otro. En este apartado se analizan distintos enfoques dentro de los LVMs, como LLaVA, que permite generar texto a partir de imágenes o incluso a partir de combinaciones texto-imagen, y los modelos de detección de objetos en modo *zero-shot,* que tienen como objetivo identificar y clasificar elementos visuales sin necesidad de entrenamiento previo.

4.2.4.1 Plataforma HuggingFace

Todos los modelos LVM seleccionados han sido obtenidos a través de *Hugging Face* [27], una plataforma enfocada en inteligencia artificial que ofrece un amplio repositorio de modelos preentrenados, *datasets* y herramientas de fácil integración. *Hugging Face* se ha consolidado como un recurso que permite el acceso a modelos de visión por computador, procesamiento del lenguaje natural y multimodalidad. Esto facilita la comparación directa de distintos modelos y arquitecturas bajo un mismo marco experimental.

La plataforma de *Hugging Face* ofrece una gran variedad de herramientas enfocadas en la inteligencia artificial. Entre ellas destaca la pestaña de modelos, que permite descargar y utilizar redes preentrenadas. Además, proporciona acceso a *datasets* con los que se pueden probar y entrenar modelos, y una sección denominada “Spaces”, un entorno de trabajo ya estructurado que permite ejecutar y evaluar modelos directamente desde la web sin tener que descargarlos localmente.

A continuación, se puede ver una captura de la página principal de *Hugging Face*, con todas las opciones que ofrece.

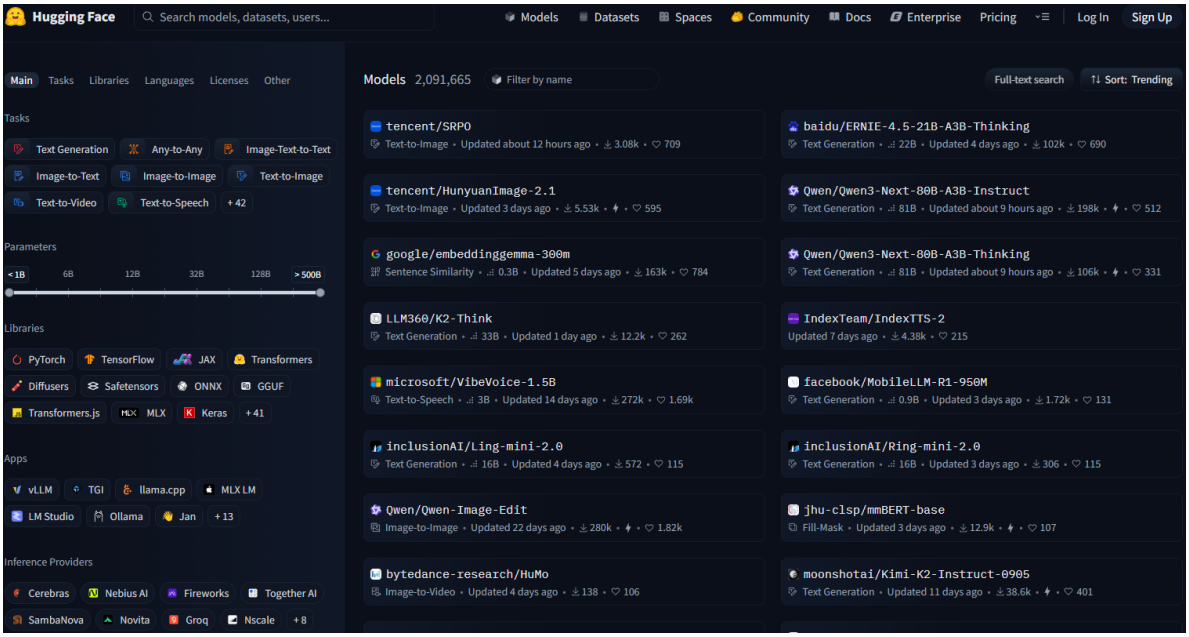


Ilustración 31. Página principal de Hugging Face [27]

Lo más interesante de esta pestaña es el panel lateral izquierdo, que permite aplicar filtros para encontrar modelos específicos según la tarea que deben desempeñar estos modelos. Entre las principales categorías se encuentran: detección de objetos, clasificación de imágenes, segmentación de imágenes, generación de texto, entre muchas otras.

En el caso concreto de este Trabajo Fin de Máster, al tratarse de un estudio enfocado en la detección de vehículos en carretera, se seleccionó la categoría de *Zero-Shot Object Detection*, así como *Image captioning* e *image-text-to-text,* para obtener descripciones textuales de imágenes [28].

4.2.4.2 Image captioning - Implementación de LLaVA

Para tareas como descripción de imágenes o responder a preguntas, uno de los modelos más utilizados dentro de los *Large Vision Models* es LLaVA, también denominado *Large Language-and-Vision Assistant*. LLaVA combina el codificador visual CLIP ViT-L/14 con un LLM llamado Vicuna mediante una matriz de proyección que permite alinear las representaciones visuales con el espacio del lenguaje. Esta arquitectura permite integrar visión y lenguaje, y ofrece muchas capacidades de razonamiento multimodal, tales como *captioning*, *visual question answering* y dialogo visual. Una de las ventajas de estos *Large Vision Models* es que no es necesario reentrenarlos constantemente, a diferencia de otros tipos de modelos de visión artificial [29]. La arquitectura del sistema es la siguiente:

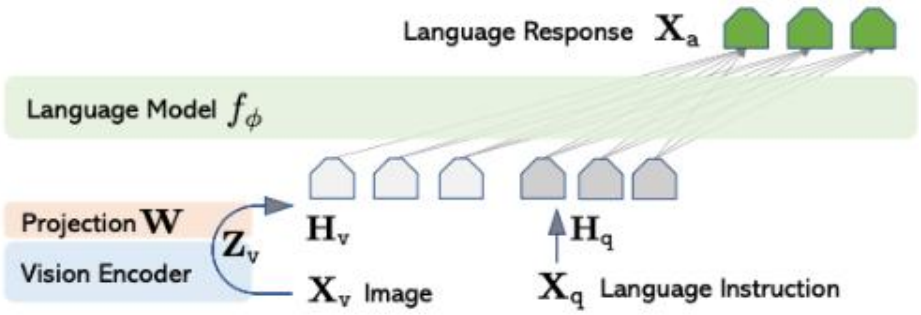


Ilustración 32. Arquitectura de LLaVA [29]

El modelo que se va a implementar concretamente es el modelo LLaVA v1.6 Mistral 7B, y su implementación se puede ver en la siguiente imagen:

```python
from transformers import LlavaNextProcessor, LlavaNextForConditionalGeneration
import torch
from PIL import Image
import requests

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Cargar el modelo y procesador
processor = LlavaNextProcessor.from_pretrained("llava-hf/llava-v1.6-mistral-7b-hf")
model = LlavaNextForConditionalGeneration.from_pretrained(
    "llava-hf/llava-v1.6-mistral-7b-hf",
    torch_dtype=torch.float16,
    low_cpu_mem_usage=True
)
model.to(device)
```

Ilustración 33. Inicialización del modelo LLaVA v1.6 Mistral 7B

El flujo de implementación de este modelo se puede visualizar en la siguiente imagen:

Ilustración 34. Flujo de trabajo del modelo LLaVA

Una vez analizado el flujo del *pipeline*, se va a probar un ejemplo de *prompt* para analizar su respuesta.

Por ejemplo, para la pregunta "¿Qué se muestra en la siguiente imagen?", LLaVA devuelve una descripción completa de la escena, incluyendo objetos, contexto y entorno. Este tipo de salida permite demostrar la capacidad de visión-lenguaje. La estructura del *prompt* y la respuesta que se va a probar se puede ver en la siguiente tabla:

Pregunta	Imagen	Respuesta
"What is shown in this image?"		"The image shows a road with a bus driving down it. The road is lined with trees, and the bus appears to be a public transit vehicle, possibly a city bus. The setting suggests a rural or semi-rural area, as indicated by the greenery and the absence of high-rise buildings. The sky is overcast, and the lighting suggests it might be late afternoon or early evening."

Tabla 3. Ejemplo de prompt proporcionado sobre LLaVA

Como se puede observar en la tabla anterior, el modelo LLaVA devuelve una respuesta muy completa respondiendo qué se ve en la imagen. El modelo responde que en la imagen aparece una carretera, con un bus recorriéndola. La carretera está rodeada de árboles, y el bus parece un vehículo de transporte público, probablemente un bus de ciudad. También detalla que se trata de una zona rural o semirrural, mostrada por las zonas verdes, y por la ausencia de edificios altos. El cielo se encuentra nublado, y la iluminación sugiere que la imagen puede estar tomada a última hora de la tarde o primera de la noche.

A diferencia de los modelos basados en la detección de objetos, LLaVA no genera directamente *bounding boxes* asociados a los objetos de la imagen. Esto limita mucho la posibilidad de aplicar métricas cuantitativas clásicas, pero se pueden obtener respuestas en formato COCO para poder comparar con las anotaciones obtenidas, y poder medir *recall* y *precision*, así como el balance a través del F1-score.

Los resultados obtenidos por el modelo se podrán ver en el apartado de resultados.

4.2.4.3 Image captioning - Implementación de Gemma-3-4b-it

Gemma es un modelo de la familia de modelos multimodales desarrollados por Google DeepMind, derivado de los modelos Gemini. Es capaz de procesar texto e imágenes de entrada, y generar texto de salida, con soporte para muchos idiomas. Su tamaño relativamente pequeño lo hace muy adecuado para entornos con recursos limitados, como la conducción autónoma.

Para el desarrollo de este Trabajo Fin de Máster, se ha decidido implementar el modelo Gemma-3-4b-it para la identificación de vehículos en imágenes de COCO. Este modelo, al igual que LLaVA, tendrá que generar descripciones semánticas, denominadas *captions*, con instrucciones precisas.

Cabe destacar que, para poder utilizar el modelo, fue necesario disponer de un token de acceso de *Hugging Face* y aceptar los términos de uso oficiales de Gemma, proporcionados por Google, que restringen su uso a fines académicos y prohíben su redistribución de pesos o su uso

comercial. Se ha decidido utilizar este modelo debido al carácter investigador del proyecto, y por su fama de buen rendimiento ante entornos no vistos.

Gemma ha demostrado un buen equilibrio entre precisión y velocidad de inferencia, pues suele ser más rápido que LLaVAv1.6, que es más grande. Esta velocidad también supone una ligera reducción de la precisión en la detección. Gracias a su capacidad multimodal y a su eficiencia, se ha convertido en una opción práctica en tareas de tratamiento y generación de texto a partir de imágenes [30].

La carga del modelo se puede ver en la siguiente imagen:

```
# ----------------------------
# Cargar modelo Gemma
# ----------------------------
model_id = "google/gemma-3-4b-it"
model = Gemma3ForConditionalGeneration.from_pretrained(model_id, device_map="auto").eval()
processor = AutoProcessor.from_pretrained(model_id)
```

Ilustración 35. Carga del modelo Gemma-3

4.2.4.4 Detección de objetos - Implementación de LVMs

El objetivo de esta sección es estudiar tres *Large Vision Models* orientados a detectar objetos en escenarios *zero-shot.*

Estos modelos están caracterizados por realizar detecciones *open-vocabulary*, lo que quiere decir que puede localizar objetos en imágenes a través de texto.

A diferencia de otros modelos LVM, estos no generan salidas textuales, sino que generan *bounding boxes* asociados a los objetos detectados, y esto permite evaluar métricas como *precision*, *recall* y mAP.

Los modelos seleccionados son los siguientes:

- **Grounding-DINO**

El modelo GroundingDINO, tanto en su versión *base* [31], como en su versión *tiny* [32], amplía un modelo de detección de objetos *zero-shot* mediante la incorporación de un codificador de texto, lo que le permite realizar *open-set object detection*. Esto significa que puede detectar en imágenes objetos descritos mediante texto, sin necesidad de entrenamiento específico con clases predefinidas. Destaca por sus resultados, alcanzando métricas como 52.5 AP en COCO sin entrenamiento supervisado en esas categorías [23].

- **OmDet-Turbo-Swin-Tiny**

El modelo OmDet-Turbo-Swin-Tiny está diseñado para detección *open-vocabulary* en escenarios *zero-shot*, combinando un *backbone* Swin-Tiny con un módulo de fusión eficiente, que permite una inferencia rápida y precisa. Su uso principal es la detección de objetos sin necesidad de

clases predefinidas, y ha sido entrenado sobre conjuntos de datos muy diversos, y esto lo hace capaz de generalizar clases no vistas [33].

- **OWLv2**

El modelo OWLv2 es una evolución de OWL-ViT y pertenece a la categoría de modelos de detección *open-vocabulary* y *zero-shot*. Su funcionamiento permite detectar objetos en una imagen a partir de descripciones en lenguaje natural.

La arquitectura utiliza como *backbone* un CLIP entrenado desde cero, con un *transformer* ViT-B/16 como codificador de imágenes y un *transformer* enmascarado como codificador de texto. La clave del enfoque está en sustituir las capas de clasificación fija por *embeddings* de nombres de clase generados por el modelo de texto, y esto permite la detección de clases no vistas [34].

4.2.4.5 Metodología de las pruebas

En este apartado, se van a detallar todos los pasos seguidos para la evaluación de los modelos LVM seleccionados en el apartado anterior, tanto los modelos de detección de objetos como los de imagen a texto. El objetivo es garantizar que la comparación sea justa y objetiva con el rendimiento de cada modelo. Para ello, se va a definir el *dataset* empleado, las tareas de evaluación, las métricas utilizadas y el procedimiento seguido.

Dataset de evaluación

El *dataset* es muy importante a la hora de comparar modelos, ya que una incorrecta selección del *dataset* puede proporcionar métricas desbalanceadas o erróneas. Es importante que el *dataset* seleccionado sea compatible con el enfoque *zero-shot*, y que sea útil por tener escenarios de tráfico y poder ser utilizado tanto para detección de objetos como para descripción semántica de imágenes.

Para estas pruebas, se ha decidido seleccionar el mismo subconjunto de COCO, como en el apartado anterior, correspondiente a los datos de validación del año 2017. Se ha decidido seleccionar este *dataset* ya que muchos modelos se evalúan en COCO, y esto asegura una base común entre ellos. Este *dataset* será utilizado tanto para la tarea de detección de objetos como para la tarea de descripción de imágenes.

Tareas de evaluación

Las tareas que se deben realizar para evaluar el rendimiento son dos concretamente:

- Detección de objetos en escenarios *zero-shot*, donde el modelo debe localizar instancias de clases en la imagen y clasificarlas.
- Descripción textual de imágenes, donde el modelo debe generar un texto que represente lo que se ve visualmente, intentando identificar los vehículos.

Los objetos que el modelo tendrá que detectar y describir serán únicamente vehículos, entre ellos coche, bus, bicicleta, moto y camión.

Métricas utilizadas

Debido a la distinta naturaleza de los LVMs a comparar, según la tarea que realicen se aplicarán unas métricas u otras.

Para el caso de los LVM orientados a detección de objetos, las métricas serán:

- Average Precision (AP)
- Average Recall (AR)
- Frames por segundo (FPS)
- Tiempo de inferencia
- Numero de detecciones

Para el caso de descripción de imagen a texto, las métricas serán métricas como:

- Precision
- Recall
- F1-score
- Frames por segundo (FPS)
- Tiempo de inferencia
- Numero de detecciones

Procedimiento de las pruebas

El proceso de evaluación de los modelos se ha desarrollado siguiendo unos pasos estructurados para garantizar la reproducibilidad de los resultados y la comparación entre ellos. Para poder reproducir los resultados, cada prueba se ha realizado sobre una celda de un Jupyter notebook.

Cabe destacar que los pasos que se van a declarar a continuación son generales a todos los modelos, por lo que, tanto para detección de objetos como para descripción semántica, los pasos seguidos, a pesar de tener diferencias evidentes, son los mismos.

1. **Selección y preparación de los datos**
   - Se emplea el conjunto de validación del *dataset* COCO 2017
   - A partir de las anotaciones oficiales, se filtran aquellas imágenes que contienen al menos un objeto de interés entre: *car, bus, truck, bicycle, motorcycle.*
2. **Carga de los modelos LVM**
   - Se descarga el modelo correspondiente desde *Hugging Face*.
   - Se prepara el *processor* y el modelo preentrenado adecuado según la tarea (detección o *captioning*).
   - El modelo se transfiere al dispositivo disponible, en este caso a la GPU.
3. **Definición de clases textuales (*prompts*)**
   - En el caso de modelos de detección, se define un *prompt* con los nombres de las clases de interés.
   - Para el caso de *captioning*, no se definen clases, sino que se define un *prompt* pidiendo que nombre los vehículos que puede ver en la imagen en base a las clases definidas.
4. **Proceso de inferencia sobre las imágenes**

- Para cada imagen, se prepara la entrada, se ejecuta la inferencia y se procesan las predicciones para ajustarlas al formato y dimensiones de salida.
- En el caso de detección, se obtienen *bounding boxes*, clases y probabilidades.
- En el caso de *captioning*, se obtiene un texto descriptivo de la imagen, que tendrá concretamente una lista de objetos vistos.

5. **Almacenamiento y visualización de resultados**
   - Los resultados se almacenan en formato estructurado JSON.
   - Para comprobar el funcionamiento, se realiza una inspección visual con un ejemplo de salida.
6. **Evaluación cuantitativa con métricas COCO**
   - Para la detección, se emplea la librería COCOeval para obtener métricas como AP y AR en distintos umbrales de intersección para el caso de detección de objetos.
   - Para el *captioning*, se utilizan métricas de evaluación estándar como *precision*, *recall*, y F1-score.

A continuación, se puede ver un ejemplo de la visualización de resultados que proporciona el entorno de pruebas para el modelo GroundingDINO-tiny.

Ilustración 36. Ejemplo de visualización de resultados para GroundingDINO-tiny

4.2.5 Uso de inteligencia artificial

Durante el desarrollo del presente proyecto, se han utilizado modelos de inteligencia artificial como herramienta complementaria en algunas fases del trabajo, tanto en el plano de desarrollo de código como en el plano documental.

En el ámbito técnico, se ha utilizado la IA para apoyar tareas de desarrollo de código, tales como la preparación del entorno y de los datos, la implementación de los modelos de visión por computadora y la obtención y análisis de los resultados obtenidos. Estas tareas han sido

facilitadas gracias al uso de agentes inteligentes que son capaces de generar código según las necesidades, como instalar dependencia y librerías, preparar el propio entorno, depurar errores y proporcionar soluciones al proyecto.

Además, la IA ha contribuido puntualmente a la estructuración de ciertos apartados del presente documento, ayudando a reorganizar los contenidos, incluir ciertas ideas interesantes y dar formato al proyecto.

4.3 Recursos requeridos

En esta sección se enumeran las herramientas tecnológicas empleadas durante las distintas fases del proyecto, tanto para el análisis teórico como para el desarrollo experimental. Entre estas herramientas se encuentra, en primer lugar, los recursos hardware, y en segundo lugar, los recursos software. Dentro de los recursos software se detallan todos los entornos de desarrollo y de programación, las librerías utilizadas, y los modelos empleados en el análisis.

Recursos Hardware

- Modelo: PC sobremesa
- Procesador: AMD Ryzen 5 3600 6-Core Processor a 3.60 GHz
- Memoria RAM: 16 GB
- Tarjeta gráfica: NVIDIA GeForce RTX 3060
- Sistema operativo: Windows 11 64 bits, procesador basado en arquitectura x64
- Periféricos: teclado y ratón de la marca Razer
- Monitor Acer
- Monitor HP

Recursos Software

- **Entornos de desarrollo y herramientas auxiliares:**
  - Python 3.10 como lenguaje principal
  - Jupyter Notebooks para prototipado y visualización interactiva.
  - Visual Studio Code como entorno de desarrollo integrado (IDE).
  - Chat-GPT y Copilot.

- **Librerías y frameworks de machine learning y visión por computador:**
  - Requests: para descargar imágenes desde URLs.
  - PyTorch: *framework* de Deep Learning, utilizado para entrenamiento e inferencia en GPU y CPU.
  - PIL: manipulación y carga de imágenes.
  - Transformers: modelos preentrenados de visión y lenguaje:
    - AutoProcessor, AutoModelForZeroShotObjectDetection
    - OmDetTurboForObjectDetection
    - Owlv2ForObjectDetection
    - LlavaNextProcessor, LlavaNextForConditionalGeneration
    - Gemma3ForConditionalGeneration

- Ultralytics: para cargar y ejecutar modelos de YOLO
- cv2 (openCV): para leer y procesar imágenes, así como para dibujar las cajas.
- Matplotlib y pyplot: visualización de resultados y *bounding boxes*.
- Numpy: para manejos de arrays y tensores.
- Pycocotool y pycocoevalcaps: para trabajar con *datasets* de COCO y evaluar modelos.
- Tqdm: para incorporar barras de progreso en bucles.
- Torchvision.ops: para operaciones de visión.
- Sklearn: para métricas como precision, recall y F1-score
- Os, glob, tempfile, gc, json, time: librerías de Python para manejo de archivos json, tiempos y recursos.

- **Modelos utilizados:**
  - YOLOv11: ultralytics.
  - GroundingDINO (tiny y base): *Hugging Face* (IDEA-Research/grounding-dino-*).
  - OmDet-Turbo: *Hugging Face* (omlab/omdet-turbo-swin-tiny-hf).
  - OWLv2: *Hugging Face* (google/owlv2-base-patch16-ensemble).
  - LLaVA-Next: *Hugging Face* (llava-hf/llava-v1.6-mistral-7b-hf).
  - Gemma-3: *Hugging Face* (google/gemma-3-4b-it).

4.4 Presupuesto

El presupuesto presentado en este apartado representa una estimación global del coste económico asociado al desarrollo de este Trabajo Fin de Máster. En él, se incluyen tanto los gastos directos, relacionados con los recursos materiales y tecnológicos utilizados, como los indirectos, más relacionados con la valoración del tiempo dedicado al desarrollo del trabajo.

Es importante destacar que además de los costes materiales, el tiempo invertido por el investigador en las diferentes actividades —incluyendo la revisión bibliográfica, el diseño, la implementación, la experimentación, la elaboración de la memoria y las reuniones de seguimiento— se considera un recurso valioso y debe ser contabilizado como parte esencial del presupuesto.

Tipo de coste	Valor	Comentarios
Horas de trabajo en el proyecto	150 horas * 13€/hora = 1950 €	Tiempo dedicado al desarrollo del proyecto de investigación, desarrollo de memoria y reuniones con el tutor.
Equipo técnico utilizado	1000 €	Valor de mercado del ordenador utilizado, junto a monitores, ratón y teclado.

Software utilizado	0 €	Todas las herramientas de software utilizados para el desarrollo de este proyecto son gratuitas.
Estudios e informes	0 €	Todos los recursos bibliográficos estudios están disponibles de forma gratuita.
Materiales empleados	0 €	No aplica

Tabla 4. Desglose de costes asociados al desarrollo del proyecto

4.5 Viabilidad

Este apartado presenta un análisis de la viabilidad económica y sostenible del proyecto, considerando tanto recursos empleados como el potencial de evolución según los resultados.

4.5.1 Viabilidad económica

El desarrollo del proyecto ha demostrado una alta viabilidad económica, al haberse llevado a cabo una inversión contenida y optimizada en cuanto a recursos. Los costes han sido principalmente utilizados debido a las horas del proyecto, y al equipo técnico utilizado. Los modelos de visión artificial han sido preentrenados y son de libre acceso, y esto ha permitido evitar costes asociados a licencias, fine-*tunning*, o infraestructuras en la nube. Estas estrategias han favorecido una relación coste-beneficio, y es especialmente importante en el contexto de la investigación aplicada, reduciendo todos los posibles costes no necesarios.

Sin embargo, es importante destacar que, para una implementación real en sistemas de conducción autónoma, es necesario invertir en hardware especializado (GPUs embebidas, sensores complementarios, etc.) y en procesos de optimización de modelos. Todas estas limitaciones pueden impactar de manera importante en los costes asociados al proyecto y por tanto a su viabilidad. Aun así, el potencial de escalabilidad y la capacidad de reducir errores humanos en entornos críticos como la conducción autónoma pueden justificar la inversión desde un punto de vista de beneficios sociales y tecnológico.

Desde el punto de vista técnico, los modelos LVM estudiados durante el proyecto muestran una evolución constante en la comunidad científica, ya que se encuentran continuamente actualizados, y esto garantiza una buena depuración de errores y mejoras continuas. Su capacidad de adaptación a distintos escenarios no vistos, gracias a los escenarios *zero-shot*, así como el aprendizaje y complementos multimodales refuerzan su sostenibilidad como una solución a largo plazo.

El enfoque modular del proyecto basado en compontes reemplazables, reutilizables y escalables permiten su integración progresiva en sistemas de percepción dentro de la conducción autónoma, facilitando su evolución y adaptación a futuras normativas cambiantes del sector.

4.5.2 Viabilidad sostenible

Mas allá de la viabilidad económica, la implementación de inteligencia artificial en la conducción autónoma, a pesar de presentar limitaciones en el apartado de los LVM, presenta un potencial impacto positivo en distintos ámbitos de sostenibilidad.

Desde el punto de vista social, la automatización del transporte puede contribuir a reducir los accidentes provocados por errores humanos, lo que mejora la seguridad vial, y genera beneficios directos a la sociedad.

En el aspecto económico, se ha podido ver que la mayoría de estos modelos están muy optimizados en cuanto a tamaño en disco, y pueden ayudar a reducir costes operativos y favorecer una movilidad eficiente.

Por último, en términos medioambientales, la conducción autónoma permite una gestión más eficiente del tráfico y del combustible consumido. Esto se traduce en una menor emisión de gases contaminantes, especialmente si se combina con vehículos eléctricos o sistemas inteligentes de movilidad.

En general, estos factores refuerzan la sostenibilidad de esta tecnología, a pesar de las evidentes limitaciones actuales que presentan en rendimiento.

4.6 Resultados del proyecto

4.6.1 Resultados de la detección de objetos

En esta sección se presentan los resultados obtenidos del proceso de evaluación de los modelos LVM aplicados sobre un conjunto de imágenes en la tarea de detección de objetos. El análisis de estas métricas se ha estructurado en torno a tres ejes fundamentales, que son: la velocidad de procesamiento, el tiempo total de inferencia y el número de objetos detectados por los modelos, a lo que se ha complementado posteriormente con métricas de precisión y recuperación.

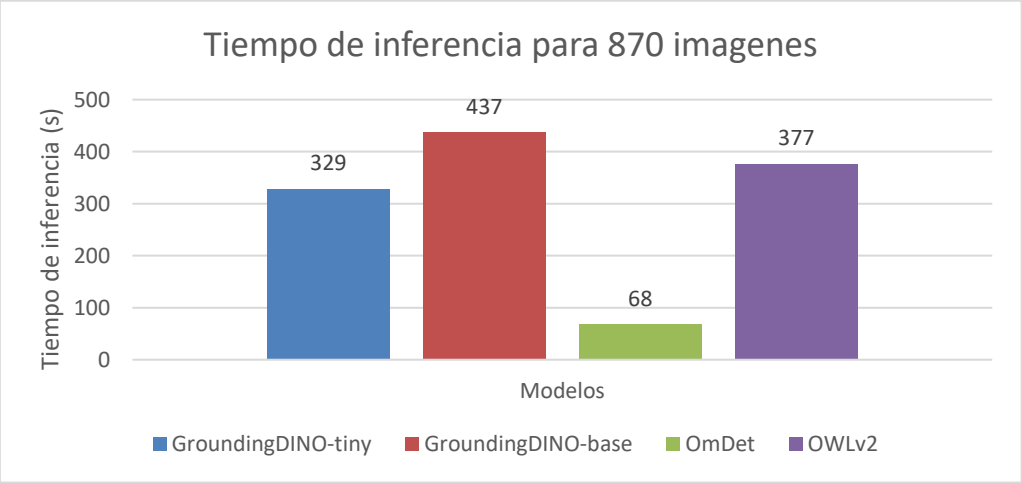


Ilustración 37. Gráfica comparativa de tiempos de inferencia

En la imagen anterior, se puede ver una gráfica que muestra los distintos tiempos de inferencia, representados en segundos, que arrojan los cuatro modelos seleccionados para procesar un conjunto de 870 imágenes en un escenario *zero-shot*.

En este caso, y como era de esperar, OmDet-Turbo destaca como el modelo más rápido, ya que solo requiere 68 segundos para procesar las 870 imágenes, y esto sugiere que es un modelo altamente optimizado para procesar imagenes. Por otro lado, los modelos GroundingDINO-tiny y OWLv2 presentan tiempos intermedios, 329s y 377s, respectivamente, ofreciendo un balance entre velocidad de inferencia y capacidad de representación. Por último, GroundingDINO-base, aunque es el más robusto porque dispone de muchos más parámetros que el resto, es el más lento con 437 segundos, reflejando una mayor complejidad computacional del modelo.

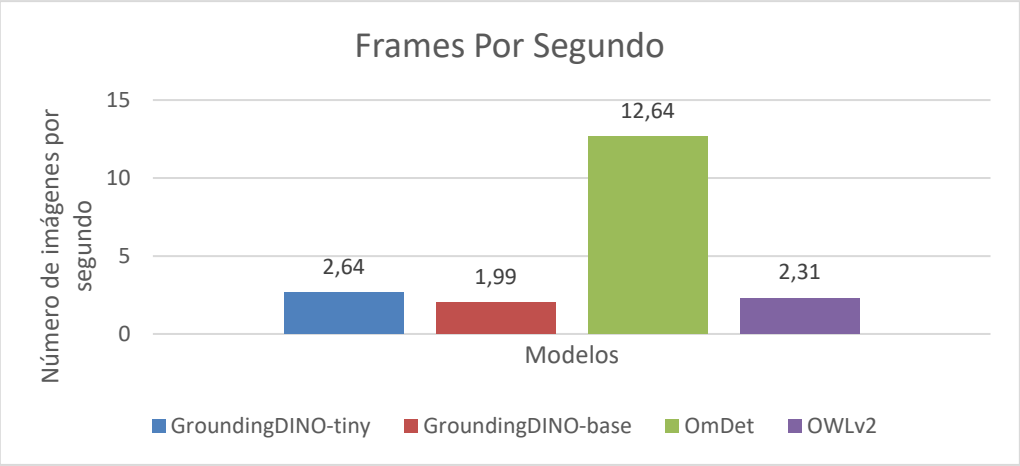


Ilustración 38. Grafica comparativa de imágenes procesadas por segundo

En la imagen anterior se muestra una gráfica con el rendimiento de los cuatro modelos definidos anteriormente en términos de velocidad de procesamiento, expresada como el número de imágenes o *frames* procesadas por segundo. Está métrica está directamente relacionada con el tiempo de inferencia, aplicado individualmente a cada una de las 870 imágenes.

En sintonía con los resultados anteriores, OmDet-Turbo sobresale con 12.64 *frames* procesados por segundo, lo que lo posiciona como el modelo más eficiente para tareas que requiere procesamiento en tiempo real. A estos resultados lo siguen los modelos GroundingDINO-tiny y OWLv2 con 2,64 y 2,31 FPS respectivamente. Por último, se encuentra GroundingDINO-base, que a pesar de su mayor capacidad de representación, no es capaz de llegar a procesar 2 FPS, con un total de 1,99 imágenes procesadas por segundo, de nuevo indicando que se trata del modelo más pesado en tareas computacionales.

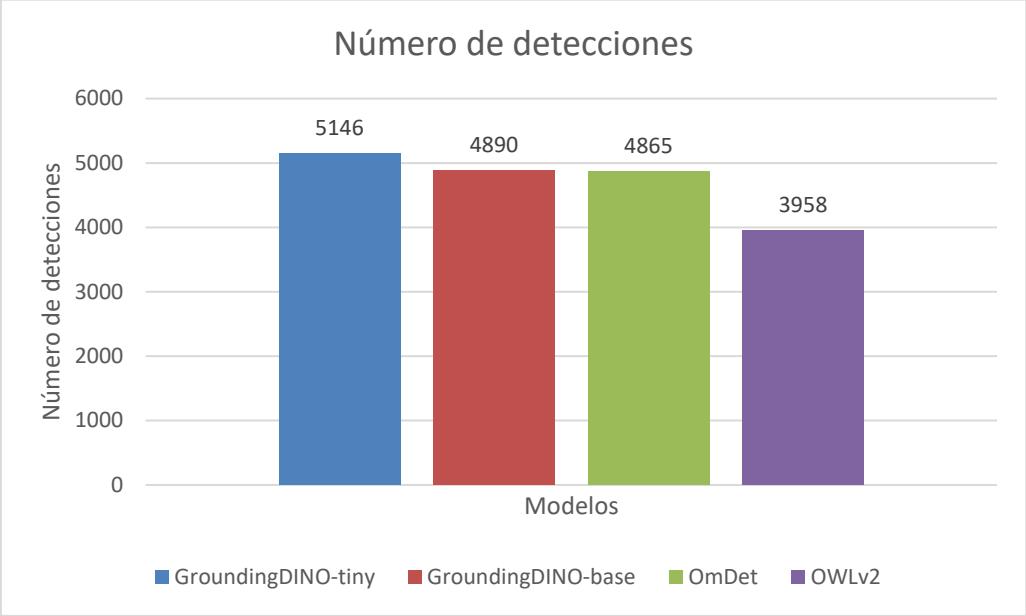


Ilustración 39. Gráfica comparativa del número de detecciones

En la imagen anterior, se muestra una gráfica con el número de objetos detectados por los cuatro modelos LVM. Es una métrica interesante para analizar, ya que permite ver la capacidad de modelos para detectar más o menos objetos en una imagen.

En este caso, el modelo GroundingDINO-tiny es el modelo que más detecciones realiza, con un total de 5146 objetos detectados en las 870 imágenes. Esto sugiere que es un modelo muy sensible a la hora de identificar objetos en este conjunto de imágenes concreto. Le sigue GroundingDINO-base, con 4890 objetos detectados, ligeramente por encima de OmDet-Turbo, que alcanza 4865 detecciones. El modelo OWLv2 es el modelo con menor número de detecciones, pues solo ha detectado 3958 vehículos, lo que puede indicar una menor cobertura o mayor selectividad en sus predicciones.

A partir del análisis del conjunto de estas tres métricas se pueden extraer ciertas conclusiones:

- OmDet-Turbo se posiciona como el modelo más eficiente en términos de rendimiento computacional. Su bajo tiempo de inferencia y el alto número de imágenes procesadas por segundo lo hacen más adecuado para aplicaciones en tiempo real. Aunque no lidera la métrica de número de objetos detectados, ofrece resultados bastante buenos y un buen equilibrio entre rapidez y rendimiento de detección.
- GroundingDINO-tiny destaca por ser el modelo que más detecciones ha realizado, lo que sugiere una alta sensibilidad en la identificación de vehículos. Sin embargo, su rendimiento computacional es limitado y tiene unos tiempos y velocidades de inferencia bastante promedios. Todo esto lo convierte en una opción interesante cuando se prioriza la capacidad de detección frente a la velocidad.
- GroundingDINO-base es el modelo más robusto de los cuatro probados, ya que es el que tiene mayor número de parámetros. Sin embargo, presenta el peor rendimiento

computacional. Su número de detecciones es elevado, lo que indica que su complejidad se traduce en la buena capacidad de representación, aunque a costa de la eficiencia.

- OWLv2 muestra un rendimiento intermedio entre velocidad, pero es el modelo que presenta menor capacidad de detecciones, lo que puede indicar una mayor selectividad en sus predicciones. Esto puede ser útil en contextos donde se requiere una mayor precisión y menor ruido en los resultados.

Una vez analizados los tiempos de inferencia, la velocidad de procesamiento y el número total de detecciones realizadas por cada uno de los modelos en el escenario *zero-shot*, es fundamental complementar este estudio con métricas que evalúan y estudian las calidades de las detecciones. Para ello, se presentan en la siguiente tabla los valores de *Average Precision* y *Average Recall* bajo distintos umbrales de intersección sobre unión (IoU) y sobre distintos tamaños de objeto.

Métrica	Grounding DINO-tiny	Grounding DINO-base	OmDet-Turbo	OWLv2	Observaciones
AP @[IoU=0.50:0.95] (All)	0.439	0.514	0.424	0.394	GroundingDINO-base lidera la precisión en general, puesto que muestra una mejor capacidad de detección en rangos de IoU variados, siendo OWLv2 el que peor resultados muestra.
AP @[IoU=0.50] (All)	0.595	0.695	0.603	0.623	Todos los modelos mejoran cuando se relaja el umbral de intersección. GroundingDINO-base sigue siendo el más preciso, y OWLv2 mejora mucho, superando a OmDet-Turbo.
AP @[IoU=0.75] (All)	0.480	0.558	0.447	0.414	GroundingDINO-base mantiene ventaja, incluso con un umbral estricto, y OWLv2 cae drásticamente. El resto cae, pero se mantiene estable.
AP Small	0.206	0.283	0.200	0.257	GroundingDINO-base destaca detectando objetos pequeños, seguido por OWLv2. En este caso OmDet-Turbo y GroundingDINO-tiny presentan dificultades para hacerlo.

AP Medium	0.452	0.525	0.418	0.400	GroundingDINO-base es otra vez superior en esta métrica. OWLv2 y OmDet-Turbo muestran un rendimiento similar, pero sigue siendo inferior, y GroundingDINO-tiny se encuentra por encima de estos dos.
AP Large	0.642	0.743	0.669	0.578	Todos los modelos rinden mejor detectando objetos grandes, siendo GroundingDINO-base el que mejores resultados muestra, siendo bastante buenos. A este modelo le sigue OmDet-Turbo.
AR MaxDets= 100	0.530	0.613	0.518	0.502	GroundingDINO-base logra una mejor recuperación de objetos. OWLv2 y OmDet-Turbo están debajo del promedio.
AR Small	0.300	0.383	0.286	0.348	GroundingDINO-base es el que mejor capacidad de recuperación de objetos pequeños muestra. En este caso, OWLv2 muestra mejor capacidad que OmDet-Turbo.
AR Medium	0.568	0.642	0.533	0.520	GroundingDINO-base también lidera la recuperación de objetos medianos, seguida del GroundingDINO-tiny, mientras que OmDet-Turbo y OWLv2 tienen un rendimiento similar.
AR Large	0.718	0.832	0.765	0.687	GroundingDINO-base tiene un notable rendimiento en la recuperación de objetos grandes, y todos los modelos prestan rendimientos buenos.

Tabla 5. Métricas de precision y recall en escenario zero-shot

En tabla anterior, se puede visualizar y analizar las distintas métricas de *precision* y *recall* en los cuatro modelos seleccionados.

Para poder analizar mejor estos resultados, se presentan dos gráficas con las principales métricas AP y AR analizadas para los modelos.

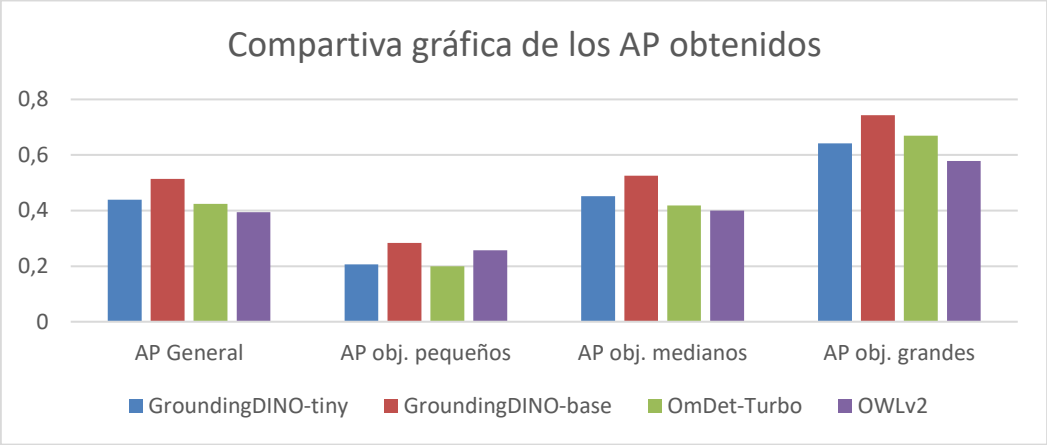


Ilustración 40. Comparativa gráfica de los modelos LVM para AP

En esta gráfica se puede ver como el modelo representado por la barra roja, GroundingDINO-base, es el que mejor precisión presenta respecto a todos los modelos. A continuación, se puede visualizar una gráfica comparando los mismos modelos para las principales métricas de *recall.*

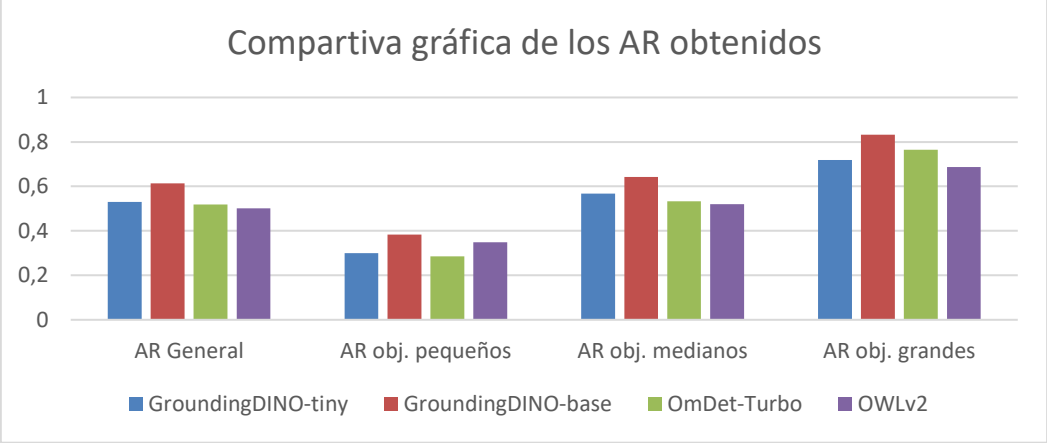


Ilustración 41. Comparativa gráfica de los modelos LVM para AR

De igual manera que en la gráfica anterior, el modelo representado por la barra roja, GroudingDINO-base, es el que mayor capacidad presenta para recuperar objetos en cualquier tamaño. A partir de estos resultados, se exponen unas breves conclusiones acerca de los resultados obtenidos.

- GroundingDINO-base se consolida como el modelo más preciso y completo en detectar objetos en escenarios *zero-shot*. Lidera en todas las métricas de *Average Precision* y *Average Recall*, tanto en rangos IoU distintos como en la detección de vehículos de distintos tamaños (vehículos que se ven más cerca se consideran objetos más pequeños en la tarea de la detección). Su rendimiento destaca especialmente en objetos grandes, donde alcanza valores muy prometedores de *recall*.

- OWLv2, aunque muestra el menor rendimiento en precisión general, mejora significativamente al relajar el umbral de intersección, y ofrece buenos resultados en detección y recuperación de objetos pequeños. Esto sugiere que este modelo puede ser útil en escenarios donde se prioriza la sensibilidad sobre la precisión estricta.
- OmDet-Turbo, a pesar de ser el más eficiente en términos de velocidad, presenta unos resultados discretos en precisión y recuperación de objetos, especialmente en los objetos más pequeños. Sin embargo, presenta un rendimiento aceptable en detección de objetos grandes, y esto lo hace competitivo en tareas menos exigentes.
- GroundingDINO-tiny ofrece un rendimiento intermedio, puesto que destaca en recuperación de objetos medianos y se mantiene por encima de OmDet-Turbo y OWLv2 en algunas de las métricas analizadas. Presenta un buen balance entre detección de objetos y velocidad de inferencia, y esto lo convierte en una opción muy versátil para entornos con recursos limitados.

4.6.2 Resultados de la descripción semántica de imágenes

En esta sección se presentan los resultados obtenidos por los modelos LVM encargados de describir semánticamente las imágenes, concretamente en la identificación de los 5 vehículos indicados anteriormente en las imágenes de validación de COCO.

Para ello, se han empleado los dos modelos ya descritos, Gemma-3-4b-it y LLaVA v1.6. Para cada imagen, se generó una *caption* con instrucciones precisas, y las predicciones han sido evaluadas frente a las anotaciones de COCO utilizando métricas de multi etiqueta como *precision*, *recall*, y F1-score.

De esta manera, se puede medir la capacidad de los modelos para reconocer correctamente los vehículos y medir la fidelidad semántica, a pesar de no devolver *bounding boxes* como en el caso de los modelos encargados de detectar objetos.

A continuación, se muestran los resultados obtenidos por dichos modelos en diferentes métricas, entre ellas las métricas de calidad, número de detecciones, FPS y tiempos de inferencia, que son también importantes a la hora de evaluarlos.

Modelo	Precision (%)	Recall (%)	F1-score (%)	Detecciones	FPS	Tiempo
Gemma-3	76.76	87.87	79.37	1743	0.112	2:09:35
LLaVA v1.6	89.16	89.91	86.82	4505	0.0105	22:59:40

Tabla 6. Resultados de modelos semánticos

Analizando la tabla anterior, para el caso de LLaVA v1.6, se obtuvo una precisión del 89.16%, lo que indica que, de todos los vehículos que el modelo predijo, 89.16% corresponde a los que realmente estaban presentes. Además, se obtuvo un *recall* de 89.91%, que indica que, de todos los vehículos presente, se logró identificar correctamente casi el 90%, y un F1-score del 86.82%,

que refleja un balance positivo y solido entre exactitud y exhaustividad. Se han detectado, de las 870 imágenes, 4505 vehículos, en un tiempo total de 22 horas, 59 minutos y 40 segundos, y esto implica un promedio de 0.0105 imágenes procesadas por segundo.

Por otro lado, Gemma obtuvo una precisión de 76.76%, lo que indica que de todos los vehículos que predijo, aproximadamente tres cuartas partes correspondía a los que realmente estaban presentes. El *recall*, de 87.87% indica que el modelo logró identificar correctamente casi 9 de 10 vehículos presentes. El F1-score refleja que hay un equilibrio razonable entre exactitud y exhaustividad. Se detectaron 1743 vehículos en las 870 imágenes, en un tiempo total de 2 horas, 9 minutos y 35 segundos, lo que indica un promedio de 0.112 imágenes procesadas por segundo.

A continuación, se pueden ver las comparativas de los distintos resultados obtenidos en diferentes gráficas, para hacer una idea visual de los rendimientos presentados por ambos modelos.

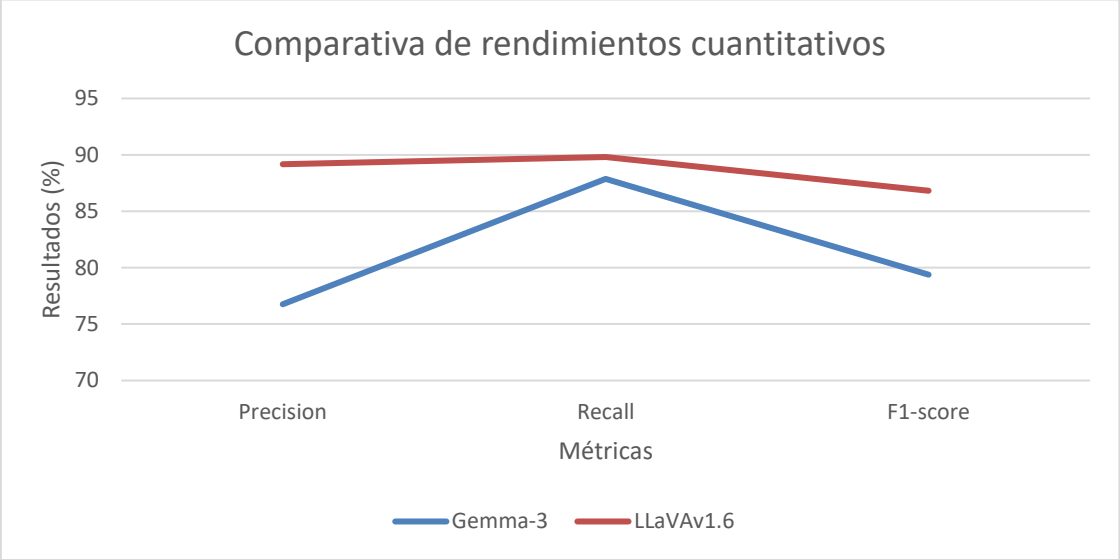


Ilustración 42. Gráfica comparativa de las métricas de precision, recall y F1-score

Como se puede ver en la gráfica anterior, el modelo LLaVA está por encima en todas las métricas de rendimiento—*precision* , *recall* y F1-score—comparado con el modelo Gemma-3. Se puede observar que la capacidad de recuperar objetos por parte de los dos modelos es muy parecida, con valores próximos. Sin embargo, la diferencia entre la capacidad de acertar con el objeto detectado y el balance entre *precision* y *recall* es mayor para el modelo LLaVA, con aproximadamente 10 % más en cada métrica.

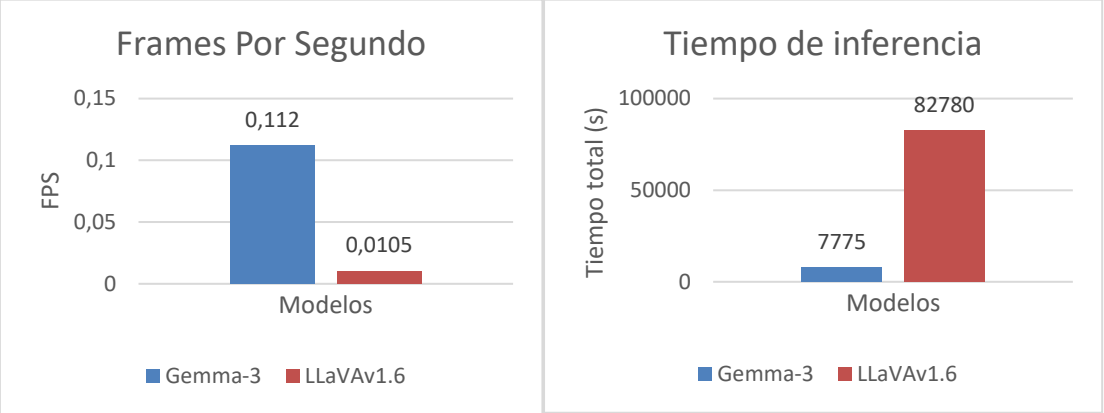


Ilustración 43. Grafica comparativa de los FPS y tiempos de inferencia

En las gráficas anteriores se pueden comparar los tiempos de inferencia necesarios para procesar las 870 imágenes de vehículos, y los *frames* procesados por segundo, que se obtienen dividiendo el tiempo de inferencia total entre el número de imágenes procesadas totales. Se puede ver que el tiempo que requiere Gemma para procesar todas las imágenes es muchísimo menor que el tiempo que requiere LLaVA, unas 10 veces menor, y por tanto los FPS son mucho mayores, también unas 10 veces mayor.

Cabe destacar que, de igual manera que en la detección de objetos, los tiempos de inferencia siguen siendo muy altos, lo que lleva importantes limitaciones al no acercarse a los 30 FPS requeridos por la visión humana.

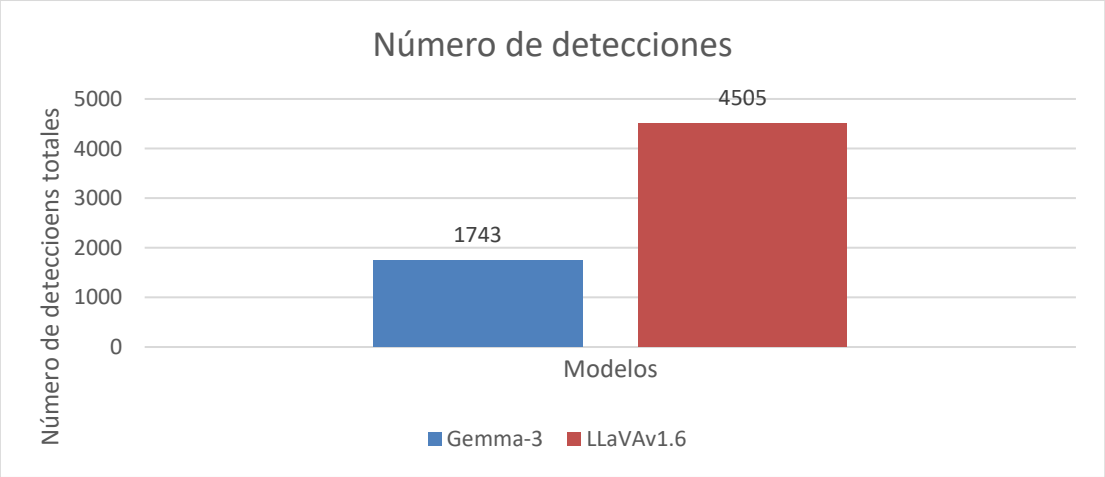


Ilustración 44. Gráfica comparativa del número de detecciones

Por último, en esta gráfica se compara el número de vehículos que los modelos han sido capaces de detectar en las 870 imágenes. Es interesante analizar que el modelo LLaVA, a pesar de ser más lento, detecta muchos más vehículos, casi 3 veces más que Gemma, situándose en línea con las métricas de precisión evaluadas.

Estas métricas son importantes, porque según la necesidad de la tarea, se puede optar por un modelo u otro. En el caso de LLaVA, es evidente que se priorizaría en tareas en las que se

requiere exactitud en las detecciones, pero no se tuviesen limitaciones de tiempo. Por otro lado, Gemma se utilizaría en situaciones donde se requieren decisiones más rápidas, pero con menos exactitud de precisión, aunque en ninguno de los casos en tiempo real.

4.6.3 Repositorio del proyecto

Durante el desarrollo del presente proyecto, se ha ido documentando y alojando todo el trabajo en un repositorio público de GitHub [35], que recopila todo el código, configuraciones, pruebas y scripts utilizados en el proyecto. El objetivo de esto es que cualquier persona interesada en los detalles de este proyecto pueda acceder libremente, e incluso ampliar los experimentos realizados.

El repositorio se puede encontrar en el siguiente enlace:

https://github.com/EnriqueAArrabal/lvm-autonomous-driving

En este repositorio se incluyen todos los modelos utilizados en este proyecto, tanto para detección de objetos como para descripción semántica de imágenes y sirve como complemento para el contenido del TFM, actuando como base para futuras mejoras o líneas de investigación en el ámbito de la conducción autónoma.

Capítulo 5. DISCUSIÓN

En este capítulo se presenta una reflexión crítica sobre el desarrollo del trabajo, así como de los resultados obtenidos y las decisiones metodológicas adoptadas. Además, se abordan las principales limitaciones encontradas, junto a una explicación de los ajustes realizados respecto a los objetivos iniciales.

En primer lugar, la metodología inicialmente planteada ha resultado muy útil y adecuada para los objetivos del proyecto. La combinación del análisis teórico y exhaustivo de los LVMs, seguida de una implementación experimental y comparativa con modelos de visión artificial más conservadores ha permitido obtener una visión integral de la situación actual de la inteligencia artificial en este ámbito. Además, la implementación de distintos modelos LVM en función de sus tareas desarrolladas, y su comparación a través de métricas cuantitativas ha permitido obtener conclusiones sólidas acerca de la viabilidad de la implementación de estos modelos en situaciones de la conducción autónoma. Esta estructuración por fases ha facilitado una progresión lógica y coherente, y permite contrastar estos modelos seleccionados con alternativas tradicionales y actuales, ofreciendo siempre unas conclusiones fundamentadas.

A lo largo del proyecto, se han realizado ciertos ajustes metodológicos en función de las necesidades:

- **Ampliación del conjunto de métricas:** Inicialmente se contemplaron métricas muy básicas como FPS, precisión, y número de detecciones. Sin embargo, a medida que avanzaba el proyecto, se vio la necesidad de incorporar métricas más avanzadas como AR y AP para evaluar también la calidad de las detecciones. Para el caso de la descripción semántica de imágenes, también se incorporaron nuevas métricas, y se eliminaron métricas más complejas, como métricas semánticas.
- **Reformulación del alcance del proyecto:** Inicialmente, el proyecto iba a contemplar también conjuntos de datos con situaciones adversas de manera que se pudiese evaluar la calidad de estos modelos en situaciones más complicadas. También se valoró estudiar la implementación del entorno simulado CARLA para probar estos modelos con videos, en lugar de solo imagen. Sin embargo, debido al alcance del proyecto, se ha considerado que el análisis realizado es más que suficiente para obtener unas conclusiones sólidas y fiables acerca de la viabilidad de la implementación. Además, las tareas sobre las que se iban a probar los modelos iban a ser únicamente de detección de objetos. Sin embargo, para ofrecer un abanico más amplio de resultados y conclusiones, se decidió incorporar modelos encargados de describir semánticamente las imágenes.

En cuanto a los resultados obtenidos, permiten realizar una valoración integral del rendimiento de los modelos LVM en tareas de percepción visual, concretamente en el contexto de la conducción autónoma. A través de las métricas analizadas principalmente—tiempo de inferencia, FPS, número de detecciones, AP, AR, *precision*, *recall* y F1-score —se ha podido identificar el potencial y las limitaciones de cada uno de los modelos. Los resultados reflejan que los LVM tienen un alto potencial para ser aplicados en sistemas de percepción de vehículos autónomos en cuanto a temas de calidad. Sin embargo, la baja velocidad de inferencia, así como

la alta complejidad computacional y la sensibilidad a detectar objetos pequeños evidencian que existen aún barreras técnicas que deben ser abordadas para poder implementar estos modelos. En sistemas de conducción autónoma, se considera que el procesamiento debe alcanzar unos 30 FPS como mínimo para igualar las capacidades de la percepción humana y reaccionar con la rapidez necesaria ante posibles eventos del entorno. Esto implica que los modelos deben ser capaces de captar, procesar y realizar una acción sobre cada una de las imágenes en menos de 33 milisegundos, algo que los modelos evaluados no han conseguido lograr. Concretamente, el modelo OmDet-Turbo ha sido el más rápido del grupo, con 12.64 FPS, pero ni siquiera cumple los requisitos de latencia. Por otro lado, el modelo GroundingDINO-base ha sido el modelo más preciso y con mejores métricas cuantitativas de AP y AR para la detección de objetos. En el caso de la descripción semántica de imágenes, el modelo LLaVA ha sido el que mejor ha descrito la imagen a través de un *prompt*. Ambos modelos han resultado ser extremadamente lentos en comparación con otros modelos utilizados, y esto sugiere que a mayor precisión proporcionada, menor será la velocidad de procesamiento, y por tanto mayor será el tiempo que requieran estos modelos para inferir una respuesta.

Por último, también es importante recoger las principales limitaciones encontradas durante el desarrollo del proyecto, ya que los resultados han arrojado ciertas deficiencias respecto a la viabilidad de aplicar estos modelos. Las limitaciones encontradas han sido las siguientes:

1) Rendimiento computacional

Algunos modelos, especialmente GroundingDINO-base para la detección de objetos, proporcionan unos tiempos de inferencia elevados para un conjunto de 870 imágenes, lo que conlleva a una velocidad de procesamiento muy baja, y esto limita su uso en escenarios de tiempo real como la conducción autónoma. El modelo LLaVA, utilizado para descripción semántica de imágenes, ha tardado un total de 23 horas para procesar 870 imágenes, lo que conlleva a una capacidad de procesado de 0.0105 FPS, algo absolutamente inviable para implementar en tiempo real. Para que se considere tiempo real, los sistemas deben procesar a 30 FPS, y el modelo que más se ha acercado llegaba a 12.64 FPS, lo que indica que aún están muy lejos de poderse utilizar en conducción autónoma.

Además, el coste computacional asociado a modelos de gran tamaño puede imponer restricciones en sistemas embebidos, como en el caso de los vehículos. Los sistemas embebidos en los vehículos disponen de recursos limitados, y se ha visto que existen modelos que pueden llegar a proporcionar problemas de espacio o memoria.

2) Sensibilidad frente a objetos pequeños

En el ámbito de la detección de objetos, las métricas AR y AP muestran que los modelos analizados tienen dificultades para detectar y recuperar objetos de tamaño reducido, y esto es crítico en contextos donde hay peatones, señales de tráfico o elementos móviles pequeños que deben ser tenidos en cuenta. Es importante destacar que vehículos grandes pero situados lejos del vehículo encargado de realizar la detección, serán considerados por el modelo como pequeños, ya que están alejados.

3) Variabilidad en la precisión

Aunque modelos como GroundingDINO-base en la detección, y LLaVA en la descripción semántica, lideran la precisión global, otros modelos como OWLv2 presentan inconsistencias en las métricas dependiendo del umbral de intersección o del tamaño del objeto, y esto dificulta su implementación estable y fiable en sistemas de percepción.

4) Escenario limitado de evaluación

El conjunto de imágenes utilizado se ha centrado única y exclusivamente en imágenes que contenían al menos un vehículo de carretera, gracias al filtrado utilizado durante la inferencia. Esto restringe la generalización de los resultados a otros elementos encontrados en la carretera, como peatones, semáforos o incluso animales que entran a la vía.

Además, el proyecto se ha llevado a cabo en un entorno con recursos limitados y no totalmente actualizados, lo que ha condicionado el alcance del análisis. Cabe destacar que, con un entorno de trabajo más actualizado y recursos computacionales mejores, los resultados podrían haberse optimizado ligeramente.

5) Ausencia de fine-tunning

Los modelos se han evaluado en tareas de detección de objetos en modo *zero-shot,* y en tareas de descripción textual del contenido de las imágenes. No se ha realizado un ajuste específico de los modelos o sus parámetros al dominio de la conducción, y esto limita su rendimiento si se comparan frente a modelos específicamente entrenados para percibir o detectar visualmente objetos en la carretera.

Capítulo 6. CONCLUSIONES

6.1 Conclusiones del trabajo

El objetivo principal de este Trabajo Fin de Máster ha sido evaluar la viabilidad de implementar modelos visuales de gran escala en tareas de percepción visual del entorno, concretamente en la conducción autónoma. Para ello, se ha realizado un estudio sobre las principales tecnologías que son utilizadas en visión artificial, así como un estudio de modelos basados en arquitecturas más tradicionales, como YOLO. Posteriormente, se ha seguido una metodología exploratoria y comparativa, donde se ha comparado el rendimiento de modelos LVM con modelos más tradicionales basados en CNN. Por último, se ha analizado el rendimiento de distintos modelos preentrenados en dos tareas concretas, detección de objetos y descripción semántica de imágenes.

Los resultados obtenidos durante el proyecto permiten concluir que los LVMs presentan un alto potencial en cuanto a calidad de detección, especialmente en métricas de precisión y recuperación. Se ha observado que modelos como DINO-base demuestran una capacidad notable para identificar objetos en distintos rangos de intersección y tamaño, y esto refuerza su aplicabilidad en entornos complejos. Además, el modelo LLaVA ha conseguido unos resultados extraordinarios en la descripción textual, pues ha acertado 9 de cada 10 veces en encontrar el vehículo que salía en la imagen.

Sin embargo, se han identificado importantes limitaciones que condicionan su implementación directa y actual en sistemas de conducción autónoma. Entre las principales limitaciones destacan la baja velocidad de inferencia, directamente ligada con los pocos *frames* procesados por segundo, la alta complejidad computacional de los modelos, y la sensibilidad frente a objetos pequeños. Un ejemplo de estas limitaciones lo ha presentado el modelo DINO-base mencionado anteriormente, puesto que, a pesar de su alta precisión, ha presentado unas velocidades de inferencia muy limitadas, al contrario que el modelo OmDet-Turbo, que ha sido el más rápido. Algo interesante ha ocurrido con el modelo LLaVA, que ha arrojado un tiempo de inferencia de 23 horas para procesar 870 imágenes, algo que limita mucho la implementación de estos modelos actualmente.

Estos aspectos deben ser abordados mediante distintas técnicas, como optimización, integración multimodal y en entornos simulados, *fine-tunning* y evaluación en condiciones más exigentes, entre otras muchas técnicas.

En general, el proyecto ha cumplido con los objetivos previstos, ofreciendo una visión clara e informada acerca del estado actual de los LVMs en el ámbito de la conducción autónoma, y permite sentar las bases para futuras líneas de investigación y desarrollo, partiendo de las bases de las fortalezas y debilidades encontradas en estos sistemas.

6.2 Conclusiones personales

Personalmente, el desarrollo de este Trabajo Fin de Máster ha sido una experiencia muy enriquecedora, tanto a nivel técnico como nivel formativo, y ha resultado altamente gratificante a nivel individual.

Este trabajo ha permitido profundizar en los modelos de visión artificial más utilizados actualmente para tareas de detección y descripción de imágenes, y esto ha facilitado comprender las fortalezas y debilidades de un sector clave de la inteligencia artificial. En particular, trabajar en el contexto de la conducción autónoma ha sido especialmente motivador.

Este tema tiene una gran relevancia social y tecnológica, y es uno de los principales campos de estudio que investigan empresas populares como pueden ser Tesla y Google. Este tema ha permitido conectar con la investigación de estos sistemas en entornos reales, y se ha podido analizar el impacto directo que pueden tener en la seguridad vial e incluso en la movilidad del futuro.

Durante el desarrollo del trabajo se ha aprendido a gestionar un proyecto complejo, con distintas necesidades metodológicas, y a adaptarse a los cambios que han surgido durante el avance del camino. También ha resultado muy importante documentar estructuradamente todos los pasos realizados, así como la importante estructuración del código para un correcto funcionamiento del entorno.

En definitiva, este proyecto ha resultado ser una oportunidad para crecer profesionalmente, permitiendo explorar el potencial de la inteligencia artificial en un campo emergente y contribuir modestamente a la implementación de estos modelos en sistemas de conducción autónoma.

Capítulo 7. FUTURAS LÍNEAS DE TRABAJO

A partir de los resultados obtenidos y de las limitaciones encontradas en los modelos, se han identificado diversas líneas de trabajo que permiten profundizar en el estudio de este Trabajo Fin de Máster y así aportar valor al proyecto en futuras fases. Por tanto, estas propuestas buscan abrir nuevas oportunidades de investigación y aplicación.

1) Optimización de modelos

Se pueden optimizar modelos a través de distintas técnicas encargadas de reducir el tamaño de los modelos de manera significativa, a costa de sacrificar la precisión. Estas técnicas pueden ser *prunning*, *quantization* o *knowledge distillation*.

Además, se pueden utilizar variantes más ligeras de los modelos actuales, como ha sido el caso de utilizar GroundingDINO-tiny o usar arquitecturas diseñadas exclusivamente para entornos embebidos.

2) Fine-tunning para conjuntos de datos específicos

Se pueden realizar ajustes finos o entrenamientos específicos de los parámetros de los modelos sobre los *datasets* específicos de la conducción autónoma, como pueden ser BDD100k o Waymo Open Dataset, con el objetivo de mejorar la precisión en entornos reales, a pesar de haber obtenido en los resultados unas precisiones decentes.

3) Integración multimodal

Se pueden combinar los modelos visuales de gran escala con datos obtenidos a través de otros sensores complementarios situados en distintas partes del vehículo, tales como LiDAR, radar o GPS. De esta forma, se añada robustez a la detección, y sobre todo se reduce la ambigüedad de las detecciones en entornos complejos.

4) Integración con entornos simulados

Como se ha mencionado en el apartado de discusiones, puede ser viable implementar modelos en entornos de conducción simulados, como el simulador CARLA, un entorno que simula la conducción de una manera muy real, y que permitiría realizar pruebas dinámicas sobre imágenes y videos en escenarios interactivos.

Esta integración puede ser útil para la evaluación temporal de los modelos y su respuesta ante situaciones cambiantes en tiempo real, como puede ser la aparición de la lluvia.

5) Evaluación en condiciones adversas

Se puede valorar incluir imágenes de vehículos en condiciones de baja iluminación, lluvia, niebla o ángulos extremos, como se ha mencionado en las conclusiones. De esta manera, se amplía el análisis y se evalúa la resiliencia de estos modelos en situaciones mucho más reales que ocurren durante la conducción.

6) Desarrollo de métricas específicas

Además de las métricas analizadas para los modelos, se pueden incorporar diversas métricas que estén orientadas a la seguridad vial, como puede ser el tiempo de reacción ante la detección de obstáculos, o la tasa de falsos negativos en zonas críticas (si detecta un obstáculo como no obstáculo en tiempo real puede conllevar a situaciones graves como atropellos) o incluso la precisión en la detección de señales de tráfico.

Además, se podrían incluir métricas semánticas que analicen lo que se acercan las predicciones obtenidas por los modelos semánticos a la respuesta esperada, en el caso de obtener anotaciones mucha más precisas que simplemente el etiquetado del vehículo.

Capítulo 8. REFERENCIAS

[1] «Pasado, presente y futuro de la conducción autónoma», Historia National Geographic. Accedido: 23 de septiembre de 2025. [En línea]. Disponible en: https://historia.nationalgeographic.com.es/promociones/pasado-presente-futuro-conduccion-autonoma_12014

[2] A. L. T. Rodríguez, «Una aproximación general al desarrollo de los coches autónomos», *CTS: Revista iberoamericana de ciencia, tecnología y sociedad*, vol. 16, n.º 47, pp. 153-175, 2021.

[3] A. Balbás Calvo, M. Domínguez, y M. D. M. Espinosa, «El volante en la conducción autónoma», *RIBIM*, vol. 23, n.º 2, pp. 89-101, oct. 2019, doi: 10.5944/ribim.23.2.42245.

[4] A. Díez Ramírez, «Conducción autónoma: Estudio del estado del arte, impacto sobre la movilidad y desarrollo de simulador de tráfico». Accedido: 23 de septiembre de 2025. [En línea]. Disponible en: https://oa.upm.es/53520/

[5] R. T. H. Collis, «Lidar», *Appl. Opt., AO*, vol. 9, n.º 8, pp. 1782-1788, ago. 1970, doi: 10.1364/AO.9.001782.

[6] J. A. Valero Matas y A. de la Barrera, «The autonomous car: a better future?», *Sociology and Technoscience*, n.º 1, pp. 136-158, 2020.

[7] editor, «El universo IoT y el sistema del vehículo conectado y autónomo • bit», bit. Accedido: 4 de octubre de 2025. [En línea]. Disponible en: https://bit.coit.es/el-universo-iot-y-el-sistema-del-vehiculo-conectado-y-autonomo/

[8] «Cómo instalar una cámara trasera en el coche», carwow.es. Accedido: 23 de septiembre de 2025. [En línea]. Disponible en: https://www.carwow.es/blog/como-instalar-camara-trasera-coche

[9] *Visión artificial*. Accedido: 23 de septiembre de 2025. [En línea]. Disponible en: https://books.google.com/books/about/Visi%C3%B3n_artificial.html?hl=es&id=FE1OEAAAQBAJ

[10] X. Wang, A. Wang, J. Yi, Y. Song, y A. Chehri, «Small Object Detection Based on Deep Learning for Remote Sensing: A Comprehensive Review», *Remote Sensing*, vol. 15, n.º 13, p. 3265, ene. 2023, doi: 10.3390/rs15133265.

[11] J. Schmidhuber, «Deep learning in neural networks: An overview», *Neural Networks*, vol. 61, pp. 85-117, ene. 2015, doi: 10.1016/j.neunet.2014.09.003.

[12] Universidad Europea, «Aprendizaje automático: Computer Vision». [En línea]. Disponible en: https://campus.europaeducationgroup.es/courses/92103/files/18216991?module_item_id=1763024

[13] S. Ren, K. He, R. Girshick, y J. Sun, «Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks», 6 de enero de 2016, *arXiv*: arXiv:1506.01497. doi: 10.48550/arXiv.1506.01497.

[14] J. Redmon, S. Divvala, R. Girshick, y A. Farhadi, «You Only Look Once: Unified, Real-Time Object Detection», 9 de mayo de 2016, *arXiv*: arXiv:1506.02640. doi: 10.48550/arXiv.1506.02640.

[15] «Vision Language Models Explained». Accedido: 23 de septiembre de 2025. [En línea]. Disponible en: https://huggingface.co/blog/vlms

[16] C. Cui *et al.*, «A Survey on Multimodal Large Language Models for Autonomous Driving», 21 de noviembre de 2023, *arXiv*: arXiv:2311.12320. doi: 10.48550/arXiv.2311.12320.

[17] R. Padilla, W. L. Passos, T. L. B. Dias, S. L. Netto, y E. A. B. da Silva, «A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit», *Electronics*, vol. 10, n.º 3, p. 279, ene. 2021, doi: 10.3390/electronics10030279.

[18] «La importancia del contexto. Más allá de los datos | Viaje al centro del dato». Accedido: 23 de septiembre de 2025. [En línea]. Disponible en: https://datacy.es/blog/la-importancia-del-contexto-mas-alla-de-los-datos/

[19] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, y M. Shah, «Transformers in Vision: A Survey», *ACM Comput. Surv.*, vol. 54, n.º 10S, p. 200, ene. 2022, doi: 10.1145/3505244.

[20] «Road Vehicle Images Dataset». Accedido: 4 de octubre de 2025. [En línea]. Disponible en: https://www.kaggle.com/datasets/ashfakyeafi/road-vehicle-images-dataset

[21] «COCO - Common Objects in Context». Accedido: 4 de octubre de 2025. [En línea]. Disponible en: https://cocodataset.org/#home

[22] J. Son y H. Jung, «Teacher–Student Model Using Grounding DINO and You Only Look Once for Multi-Sensor-Based Object Detection», *Applied Sciences*, vol. 14, n.º 6, p. 2232, ene. 2024, doi: 10.3390/app14062232.

[23] «Grounding DINO». Accedido: 4 de octubre de 2025. [En línea]. Disponible en: https://huggingface.co/docs/transformers/v4.50.0/model_doc/grounding-dino

[24] «Ultralytics | Ultralytics YOLO11 | Kaggle». Accedido: 4 de octubre de 2025. [En línea]. Disponible en: https://www.kaggle.com/models/ultralytics/yolo11

[25] K. Naminas, «Mean Average Precision: Is It the Same as Average Precision?» Accedido: 4 de octubre de 2025. [En línea]. Disponible en: https://labelyourdata.com/articles/mean-average-precision-map

[26] «(PDF) Investigating Robustness of Open-Vocabulary Foundation Object Detectors under Distribution Shifts», ResearchGate. Accedido: 4 de octubre de 2025. [En línea]. Disponible en: https://www.researchgate.net/publication/380894978_Investigating_Robustness_of_Open-Vocabulary_Foundation_Object_Detectors_under_Distribution_Shifts

[27] «Hugging Face – The AI community building the future.» Accedido: 4 de octubre de 2025. [En línea]. Disponible en: https://huggingface.co/

[28] «Image-Text-to-Text Models – Hugging Face». Accedido: 4 de octubre de 2025. [En línea]. Disponible en: https://huggingface.co/models

[29] «LLaVA». Accedido: 4 de octubre de 2025. [En línea]. Disponible en: https://llava-vl.github.io/

[30] «google/gemma-3-4b-it · Hugging Face». Accedido: 5 de octubre de 2025. [En línea]. Disponible en: https://huggingface.co/google/gemma-3-4b-it

[31] «IDEA-Research/grounding-dino-base · Hugging Face». Accedido: 5 de octubre de 2025. [En línea]. Disponible en: https://huggingface.co/IDEA-Research/grounding-dino-base

[32] «IDEA-Research/grounding-dino-tiny · Hugging Face». Accedido: 5 de octubre de 2025. [En línea]. Disponible en: https://huggingface.co/IDEA-Research/grounding-dino-tiny

[33] «omlab/omdet-turbo-swin-tiny-hf · Hugging Face». Accedido: 5 de octubre de 2025. [En línea]. Disponible en: https://huggingface.co/omlab/omdet-turbo-swin-tiny-hf

[34] «google/owlv2-base-patch16-ensemble · Hugging Face». Accedido: 5 de octubre de 2025. [En línea]. Disponible en: https://huggingface.co/google/owlv2-base-patch16-ensemble

[35] EnriqueAArrabal, *EnriqueAArrabal/lvm-autonomous-driving*. (13 de octubre de 2025). Jupyter Notebook. Accedido: 13 de octubre de 2025. [En línea]. Disponible en: https://github.com/EnriqueAArrabal/lvm-autonomous-driving

Capítulo 9. ANEXOS

draw_bounds.py > draw_yolo_boxes

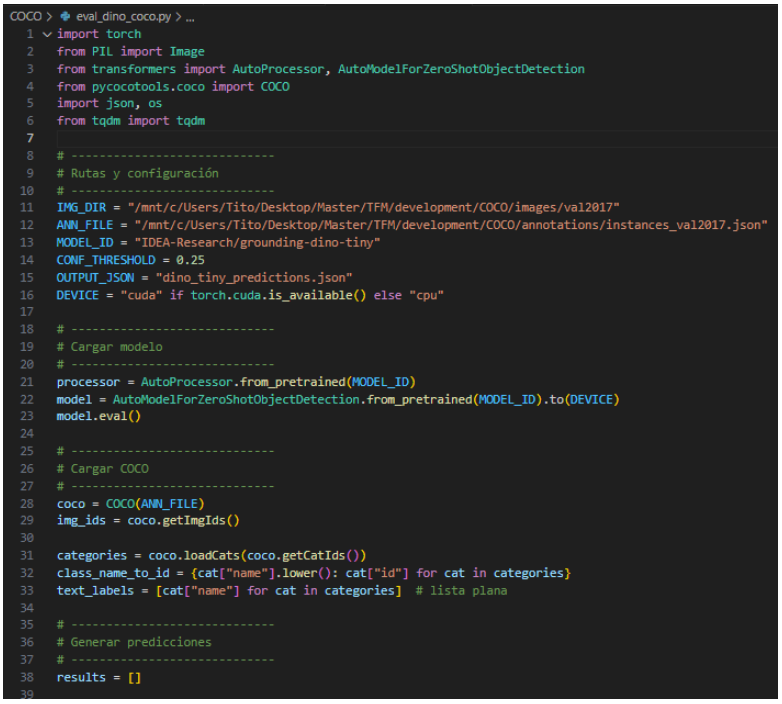
```python
import cv2
import matplotlib.pyplot as plt

def draw_yolo_boxes(image_path, label_path):
    # Cargar imagen
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    height, width, _ = image.shape

    # Leer cajas desde el txt
    with open(label_path, 'r') as f:
        for line in f:
            parts = line.strip().split()
            class_id = int(parts[0])
            x_center = float(parts[1]) * width
            y_center = float(parts[2]) * height
            box_width = float(parts[3]) * width
            box_height = float(parts[4]) * height

            # Convertir a formato (x1, y1, x2, y2)
            x1 = int(x_center - box_width / 2)
            y1 = int(y_center - box_height / 2)
            x2 = int(x_center + box_width / 2)
            y2 = int(y_center + box_height / 2)

            # Dibujar rectángulo
            cv2.rectangle(image, (x1, y1), (x2, y2), color=(0, 255, 0), thickness=2)
            cv2.putText(image, f'Class {class_id}', (x1, y1 - 5), cv2.FONT_HERSHEY_SIMPLEX,
                        0.5, (255, 0, 0), 1, cv2.LINE_AA)

    # Mostrar la imagen con matplotlib
    plt.figure(figsize=(8, 8))
    plt.imshow(image)
    plt.axis('off')
    plt.title("Bounding Boxes")
    plt.show()

# Ejemplo de uso
draw_yolo_boxes(
    "./datasets/trafic_data/train/images/41_jpg.rf.cccfea1ad9ef5ff8ad3b5468c36b2709.jpg",
    "./datasets/trafic_data/train/labels/41_jpg.rf.cccfea1ad9ef5ff8ad3b5468c36b2709.txt"
)
```

Ilustración 45. Script para dibujar bounding boxes de ejemplo

COCO > eval_yolo_coco.py > ...

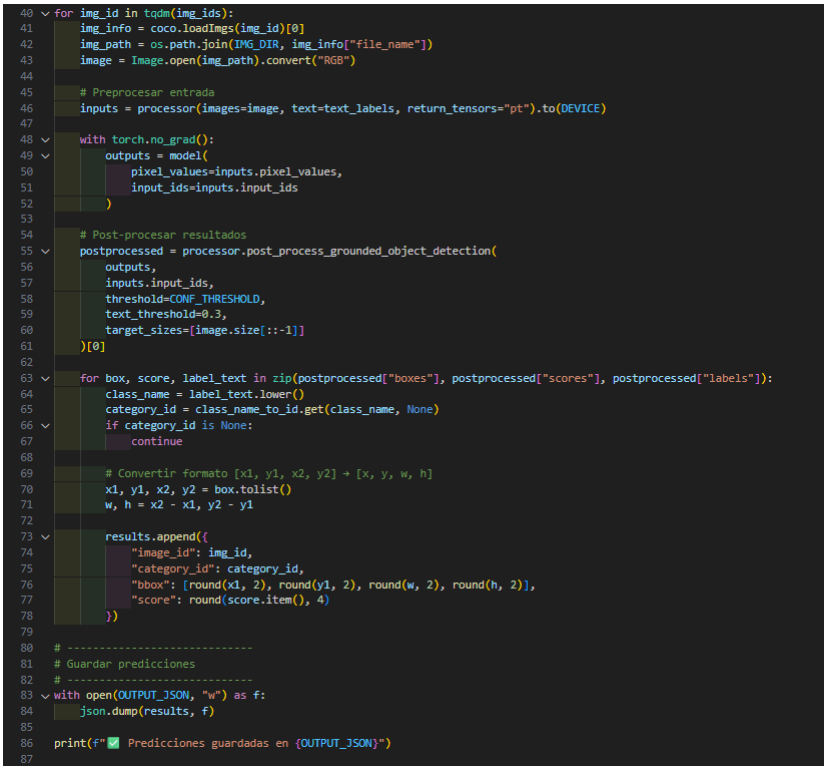
```python
from ultralytics import YOLO
from pycocotools.coco import COCO
import json, os
from PIL import Image
from tqdm import tqdm

# Rutas
IMG_DIR = "C:/Users/Tito/Desktop/Master/TFM/development/COCO/images/val2017"
ANN_FILE = "C:/Users/Tito/Desktop/Master/TFM/development/COCO/annotations/instances_val2017.json"
MODEL_PATH = "yolo11n.pt"  # Cambia según el modelo que uses

# Cargar modelo
model = YOLO(MODEL_PATH)

# Cargar COCO
coco = COCO(ANN_FILE)
img_ids = coco.getImgIds()

# Diccionario de clases COCO
categories = coco.loadCats(coco.getCatIds())
class_name_to_id = {cat["name"]: cat["id"] for cat in categories}

results = []

for img_id in tqdm(img_ids):
    img_info = coco.loadImgs(img_id)[0]
    img_path = os.path.join(IMG_DIR, img_info["file_name"])
    image = Image.open(img_path).convert("RGB")

    # Inferencia
    pred = model.predict(image, imgsz=640, conf=0.25)[0]

    for box in pred.boxes:
        x1, y1, x2, y2 = box.xyxy[0].tolist()
        w, h = x2 - x1, y2 - y1
        score = box.conf[0].item()
        class_id = int(box.cls[0].item())
        class_name = model.names[class_id]
        category_id = class_name_to_id.get(class_name, None)
        if category_id is None:
            continue

        results.append({
            "image_id": img_id,
            "category_id": category_id,
            "bbox": [x1, y1, w, h],
            "score": score
        })

# Guardar predicciones
with open("yolo11_predictions.json", "w") as f:
    json.dump(results, f)
```

Ilustración 37. Script para obtener predicciones con YOLOv11



```python
import torch
from PIL import Image
from transformers import AutoProcessor, AutoModelForZeroShotObjectDetection
from pycocotools.coco import COCO
import json, os
from tqdm import tqdm

# ----------------------------
# Rutas y configuración
# ----------------------------
IMG_DIR = "/mnt/c/Users/Tito/Desktop/Master/TFM/development/COCO/images/val2017"
ANN_FILE = "/mnt/c/Users/Tito/Desktop/Master/TFM/development/COCO/annotations/instances_val2017.json"
MODEL_ID = "IDEA-Research/grounding-dino-tiny"
CONF_THRESHOLD = 0.25
OUTPUT_JSON = "dino_tiny_predictions.json"
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"

# ----------------------------
# Cargar modelo
# ----------------------------
processor = AutoProcessor.from_pretrained(MODEL_ID)
model = AutoModelForZeroShotObjectDetection.from_pretrained(MODEL_ID).to(DEVICE)
model.eval()

# ----------------------------
# Cargar COCO
# ----------------------------
coco = COCO(ANN_FILE)
img_ids = coco.getImgIds()

categories = coco.loadCats(coco.getCatIds())
class_name_to_id = {cat["name"].lower(): cat["id"] for cat in categories}
text_labels = [cat["name"] for cat in categories]  # lista plana

# ----------------------------
# Generar predicciones
# ----------------------------
results = []
```

Ilustración 46. Script para obtener predicciones cono DINO - parte 1



```python
for img_id in tqdm(img_ids):
    img_info = coco.loadImgs(img_id)[0]
    img_path = os.path.join(IMG_DIR, img_info["file_name"])
    image = Image.open(img_path).convert("RGB")

    # Preprocesar entrada
    inputs = processor(images=image, text=text_labels, return_tensors="pt").to(DEVICE)

    with torch.no_grad():
        outputs = model(
            pixel_values=inputs.pixel_values,
            input_ids=inputs.input_ids
        )

    # Post-procesar resultados
    postprocessed = processor.post_process_grounded_object_detection(
        outputs,
        inputs.input_ids,
        threshold=CONF_THRESHOLD,
        text_threshold=0.3,
        target_sizes=[image.size[::-1]]
    )[0]

    for box, score, label_text in zip(postprocessed["boxes"], postprocessed["scores"], postprocessed["labels"]):
        class_name = label_text.lower()
        category_id = class_name_to_id.get(class_name, None)
        if category_id is None:
            continue

        # Convertir formato [x1, y1, x2, y2] → [x, y, w, h]
        x1, y1, x2, y2 = box.tolist()
        w, h = x2 - x1, y2 - y1

        results.append({
            "image_id": img_id,
            "category_id": category_id,
            "bbox": [round(x1, 2), round(y1, 2), round(w, 2), round(h, 2)],
            "score": round(score.item(), 4)
        })

# ----------------------------
# Guardar predicciones
# ----------------------------
with open(OUTPUT_JSON, "w") as f:
    json.dump(results, f)

print(f"✅ Predicciones guardadas en {OUTPUT_JSON}")
```

Ilustración 47. Script para obtener predicciones con DINO - parte 2

```python
COCO > evaluate.py > ...
from pycocotools.coco import COCO
from pycocotools.cocoeval import COCOeval

# Cargar anotaciones reales
coco_gt = COCO("./annotations/instances_val2017.json")

# Cargar predicciones del modelo
coco_dt = coco_gt.loadRes("./yolo11_predictions.json")

# Evaluar
coco_eval = COCOeval(coco_gt, coco_dt, "bbox")
coco_eval.evaluate()
coco_eval.accumulate()
coco_eval.summarize()

```

Ilustración 48. Script de evaluación de métricas de COCO con YOLO

```python
inference > dino_inference.py > ...
import requests
import torch
from PIL import Image
from transformers import AutoProcessor, AutoModelForZeroShotObjectDetection
import time

model_id = "IDEA-Research/grounding-dino-tiny"
device = "cuda" if torch.cuda.is_available() else "cpu"

processor = AutoProcessor.from_pretrained(model_id)
model = AutoModelForZeroShotObjectDetection.from_pretrained(model_id).to(device)

# Imagen local
image_path = "./datasets/trafic_data/train/images/41_jpg.rf.cccfea1ad9ef5ff8ad3b5468c36b2709.jpg"
image = Image.open(image_path)

# Etiquetas (puedes usar tus clases de tráfico)
text_labels = [["ambulance", "army vehicle", "auto rickshaw", "bicycle", "bus",
                "car", "garbage van", "human hauler", "minibus", "minivan",
                "motorbike", "pickup truck", "police car", "rickshaw", "scooter",
                "SUV", "taxi", "three wheeler CNG", "truck", "van", "wheelbarrow"]]

num_runs = 100
total_time = 0.0

for _ in range(num_runs):
    inputs = processor(images=image, text=text_labels, return_tensors="pt").to(device)
    start_time = time.time()
    with torch.no_grad():
        outputs = model(**inputs)
    total_time += time.time() - start_time

avg_time = total_time / num_runs
fps = 1.0 / avg_time

print(f"Tiempo medio por imagen: {avg_time:.3f} s")
print(f"FPS aproximados: {fps:.2f}")
```

Ilustración 49. Script para calcular tiempos de inferencia en DINO

inference > yolo11_inference.py > ...

```python
import time
from ultralytics import YOLO

if __name__ == "__main__":
    # Cargar el modelo
    model = YOLO("yolo11n.pt")

    # Imagen de prueba
    img = "./datasets/trafic_data/train/images/41_jpg.rf.cccfea1ad9ef5ff8ad3b5468c36b2709.jpg"

    # Calentar la GPU (primera pasada no cuenta)
    model(img)

    # Medir tiempo de inferencia
    N = 100  # número de repeticiones
    start = time.time()
    for _ in range(N):
        results = model(img)
    end = time.time()

    avg_time = (end - start) / N
    fps = 1 / avg_time

    print(f"Tiempo medio por imagen: {avg_time:.4f} s ({avg_time*1000:.2f} ms)")
    print(f"FPS aproximados: {fps:.2f}")

```

Ilustración 50. Script para calcular tiempos de inferencia en YOLOv11

```python
# ==============================
# Función para medir memoria en GPU
# ==============================
def medir_memoria_inferencia(modelo, inputs, dispositivo):
    torch.cuda.empty_cache()
    torch.cuda.reset_peak_memory_stats(dispositivo)

    modelo.to(dispositivo)
    inputs = {k: v.to(dispositivo) for k, v in inputs.items()}

    with torch.no_grad():
        _ = modelo(**inputs)

    mem_pico = torch.cuda.max_memory_allocated(dispositivo) / (1024 ** 2)
    return mem_pico

# ==============================
# YOLOv11
# ==============================
print("=== YOLOv11 ===")
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Cargar modelo YOLOv11
yolo_model_path = "yolo11n.pt"
yolo_model = YOLO(yolo_model_path)

# Crear imagen de prueba (tensor aleatorio)
yolo_input = torch.randn(1, 3, 640, 448)

# Medir memoria
yolo_memoria = medir_memoria_inferencia(yolo_model.model, {"x": yolo_input}, device)
print(f"Memoria pico usada en GPU: {yolo_memoria:.2f} MB")

# ==============================
# GroundingDINO-Tiny
# ==============================
print("\n=== GroundingDINO-Tiny ===")

# Cargar modelo y procesador
modelo_id = "IDEA-Research/grounding-dino-tiny"
processor = AutoProcessor.from_pretrained(modelo_id)
dino_model = AutoModelForZeroShotObjectDetection.from_pretrained(modelo_id)

# Usar imagen real para inferencia
url = "https://huggingface.co/datasets/Narsil/image_dummy/raw/main/lena.png"
image = Image.open(requests.get(url, stream=True).raw).convert("RGB")
text_prompt = ["a person", "a face"]

# Preparar inputs correctamente
inputs = processor(images=image, text=text_prompt, return_tensors="pt")

# Medir memoria
dino_memoria = medir_memoria_inferencia(dino_model, inputs, device)
print(f"Memoria pico usada en GPU: {dino_memoria:.2f} MB")

```

Ilustración 51. Script para calcular memoria utilizada en DINO y YOLOv11

```python
# ==============================
# YOLOv11
# ==============================
print("=== YOLOv11 ===")

# Ruta al archivo del modelo YOLOv11
yolo_model_path = "yolo11n.pt"

# Tamaño en disco
yolo_size_bytes = os.path.getsize(yolo_model_path)
yolo_size_mb = yolo_size_bytes / (1024 * 1024)

# Cargar modelo y contar parámetros
yolo_model = YOLO(yolo_model_path)
yolo_num_params = sum(p.numel() for p in yolo_model.model.parameters())

print(f"Tamaño en disco: {yolo_size_mb:.2f} MB")
print(f"Número de parámetros: {yolo_num_params:,}")

# ==============================
# GroundingDINO-Tiny
# ==============================
print("\n=== GroundingDINO-Tiny ===")

# ID del modelo en Hugging Face
grounding_model_id = "IDEA-Research/grounding-dino-tiny"

# Descargar y cargar modelo
grounding_model = AutoModelForZeroShotObjectDetection.from_pretrained(grounding_model_id)

# Número de parámetros
gd_num_params = sum(p.numel() for p in grounding_model.parameters())
print(f"Número de parámetros: {gd_num_params:,}")

# Guardar el modelo en una carpeta temporal para calcular el tamaño total
with tempfile.TemporaryDirectory() as tmpdirname:
    grounding_model.save_pretrained(tmpdirname)

    total_size_bytes = 0
    for root, dirs, files in os.walk(tmpdirname):
        for file in files:
            filepath = os.path.join(root, file)
            total_size_bytes += os.path.getsize(filepath)

    gd_size_mb = total_size_bytes / (1024 * 1024)
    print(f"Tamaño en disco: {gd_size_mb:.2f} MB")
```

Ilustración 52. Script para analizar almacenamiento y parámetros de YOLO y DINO

LLaVa_model.py > ...

```python
from transformers import LlavaNextProcessor, LlavaNextForConditionalGeneration
import torch
from PIL import Image
import requests

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Cargar el modelo y procesador
processor = LlavaNextProcessor.from_pretrained("llava-hf/llava-v1.6-mistral-7b-hf")
model = LlavaNextForConditionalGeneration.from_pretrained(
    "llava-hf/llava-v1.6-mistral-7b-hf",
    torch_dtype=torch.float16,
    low_cpu_mem_usage=True
)
model.to(device)

# Cargar la imagen
image = Image.open("./datasets/trafic_data/train/images/41_jpg.rf.cccfea1ad9ef5ff8ad3b5468c36b2709.jpg")

# Construir conversación en formato correcto
conversation = [
    {
        "role": "user",
        "content": [
            {"type": "text", "text": "What is shown in this image?"},
            {"type": "image"}
        ]
    }
]

# Crear prompt con plantilla
prompt = processor.apply_chat_template(conversation, add_generation_prompt=True)

# Preparar inputs para el modelo
inputs = processor(text=prompt, images=image, return_tensors="pt").to(device)

# Generar salida
output = model.generate(**inputs, max_new_tokens=100)

# Mostrar respuesta
print(processor.decode(output[0], skip_special_tokens=True))

```

Ilustración 53. Script para generación de texto por el modelo LLaVA

```python
# ============================
# OmDet-Turbo en COCO val2017 (vehículos)
# ============================

import os
import torch
import json
from tqdm import tqdm
from PIL import Image, ImageDraw
from pycocotools.coco import COCO
from pycocotools.cocoeval import COCOeval
from transformers import AutoProcessor, OmDetTurboForObjectDetection

device = "cuda" if torch.cuda.is_available() else "cpu"

# ----- 1️⃣ COCO dataset -----
dataDir = "./COCO"
dataType = "val2017"
annFile = os.path.join(dataDir, "annotations", f"instances_{dataType}.json")
coco_gt = COCO(annFile)

vehicle_classes = ['car', 'bus', 'truck', 'bicycle', 'motorcycle']
cat_ids = coco_gt.getCatIds(catNms=vehicle_classes)

# Filtramos imágenes con vehículos
img_ids_set = set()
for cat_id in cat_ids:
    ids = coco_gt.getImgIds(catIds=[cat_id])
    img_ids_set.update(ids)

img_ids = list(img_ids_set)
images = coco_gt.loadImgs(img_ids)

print(f"{len(images)} imágenes con al menos un vehículo encontradas.")

# ----- 2️⃣ Cargar OmDet-Turbo -----
model_id = "omlab/omdet-turbo-swin-tiny-hf"
processor = AutoProcessor.from_pretrained(model_id)
model = OmDetTurboForObjectDetection.from_pretrained(model_id).to(device)
model.eval()

# ----- 3️⃣ Inferencia y conversión a formato COCO -----
coco_results = []

# Puedes limitar para pruebas rápidas
# for img in tqdm(images[:50], desc="Procesando imágenes"):
for img in tqdm(images, desc="Procesando imágenes"):
    img_path = os.path.join(dataDir, "images", dataType, img['file_name'])
    image = Image.open(img_path).convert("RGB")
    width, height = image.size

    # 👈 Aquí pasamos imagen + clases
    inputs = processor(images=image, text=vehicle_classes,
                       return_tensors="pt").to(device)
```

Ilustración 54. Script para inferencia y evaluación de OmDet-Turbo - parte 1

```
    with torch.no_grad():
        outputs = model(**inputs)

    # 👈 Postproceso correcto para OmDet
    results = processor.post_process_grounded_object_detection(
        outputs,
        vehicle_classes,   # 👈 directamente la lista de clases
        threshold=0.25,
        target_sizes=[(height, width)]
    )[0]

    # Dibujar detecciones y guardar en formato COCO
    draw = ImageDraw.Draw(image)
    for box, score, label_id in zip(results["boxes"], results["scores"], results["labels"]):
        # Convierte tensor -> int
        label_id = int(label_id.item()) if hasattr(label_id, "item") else int(label_id)
        # Mapea al nombre de clase
        label_clean = vehicle_classes[label_id]

        if label_clean in vehicle_classes:
            x1, y1, x2, y2 = box.tolist()
            coco_results.append({
                "image_id": img['id'],
                "category_id": coco_gt.getCatIds(catNms=[label_clean])[0],
                "bbox": [x1, y1, x2 - x1, y2 - y1],
                "score": float(score)
            })
            # Opcional: dibujar
            draw.rectangle([x1, y1, x2, y2], outline="red", width=2)
            draw.text((x1, y1 - 10), f"{label_clean}: {score:.2f}", fill="red")


# ----- 4️⃣ Guardar resultados -----
with open("omdet_turbo_coco_results.json", "w") as f:
    json.dump(coco_results, f)

print(f"Resultados guardados: {len(coco_results)} detecciones")

# ----- 5️⃣ Evaluación COCO -----
if len(coco_results) > 0:
    coco_dt = coco_gt.loadRes("omdet_turbo_coco_results.json")
    coco_eval = COCOeval(coco_gt, coco_dt, iouType='bbox')
    coco_eval.params.catIds = cat_ids  # Solo vehículos
    coco_eval.evaluate()
    coco_eval.accumulate()
    coco_eval.summarize()
else:
    print("No se detectaron objetos, ajusta thresholds.")

# Mostrar en notebook
plt.figure(figsize=(10, 8))
plt.imshow(image)
plt.axis("off")
plt.show()
```

Ilustración 55. Script para inferencia y evaluación de OmDet-Turbo - parte 2

```
prompt_template = (
    "List all vehicles visible in the image using only these options: "
    "car, bus, truck, motorcycle, bicycle. Respond with a comma-separated list only."
)
```

Ilustración 56. Estructura del prompt utilizado en Gemma-3 y LLaVAv1.6

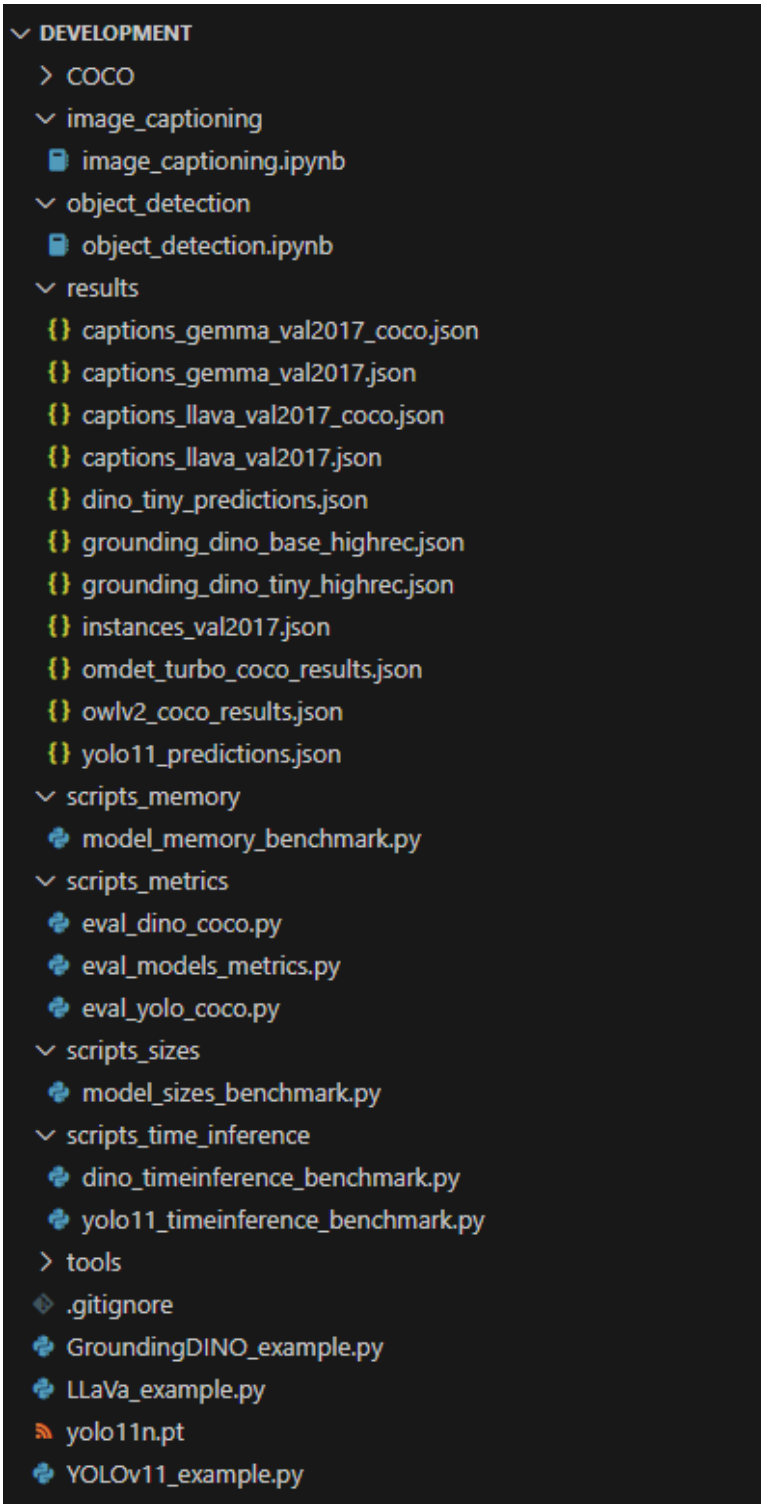


Ilustración 57. Estructura del código empleado para el desarrollo

[PÁGINA INTENCIONADAMENTE EN BLANCO]