Evaluación de modelos de DL en nubes de puntos LiDAR para aplicaciones de automoción en tiempo real

Pablo Liste[1], Silvia R. Alcaraz[1], Oscar G. Lorenzo[1,2], José C. Cabaleiro[1,2] y Francisco F. Rivera[1,2]

Resumen— **Este trabajo presenta una evaluación de la precisión, el rendimiento y el consumo de recursos de modelos de *Deep Learning* para nubes de puntos 3D orientados a aplicaciones de automoción en tiempo real. En concreto, se han evaluado los modelos PointNet, PointNet++, PointPillars y PointRCNN en la tarea de clasificación. Para ello, se han desarrollado dos aplicaciones: un *pipeline* con ROS2 para la identificación, clasificación y seguimiento de objetos en tiempo real; y una herramienta de evaluación que permite comparar modelos en función de un conjunto de métricas configurables. Los resultados obtenidos, utilizando el *dataset* KITTI y bajo las condiciones empleadas en este estudio, señalan a PointPillars como el modelo mejor balanceado en cuanto a rendimiento y precisión. Por su parte, PointRCNN se posiciona como el más preciso, mientras que PointNet destaca por ofrecer el menor tiempo de inferencia. Con respecto al uso de recursos, aunque PointRCNN y PointPillars son modelos significativamente más pesados, presentan un consumo de memoria por objeto inferior al de PointNet y PointNet++ durante el procesamiento. Por último, todos los modelos requirieron aceleración mediante GPU para completar la tarea de inferencia dentro de las condiciones de tiempo real establecidas en el *pipeline* desarrollado.**

Palabras clave— ***Deep Learning*, nubes de puntos 3D, LiDAR, *tracking* en tiempo real, automoción**

I. Introducción

LOS datos obtenidos mediante sensores *Laser Imaging Detection and Ranging* (LiDAR) son utilizados en un amplio abanico de aplicaciones debido a su alta precisión. En el ámbito de la automoción, en el que se centra este trabajo, se emplean para múltiples tareas como la identificación de objetos (p.e.: peatones, obstáculos, vehículos, baches, etc.), el cálculo de volúmenes o velocidades, entre otros. Con el auge de la inteligencia artificial (IA), ciertas tareas como la segmentación o la clasificación de nubes de puntos 3D han ido evolucionando desde el uso de reglas basadas en propiedades geométricas al uso de técnicas de aprendizaje automático o *Machine Learning* (ML) o, posteriormente, modelos más complejos basados en aprendizaje profundo o *Deep Learning* (DL). Estos modelos, han tenido que adaptarse empleando distintos enfoques para poder funcionar con datos desestructurados tridimensionales, como las nubes LiDAR. Desde la publicación de PointNet [1], modelo de DL pionero para trabajar con este tipo de datos en bruto, han aparecido nuevas arquitecturas que buscan mejorar tanto la precisión de las predicciones como aspectos relacionados con el rendimiento. El trabajo de Li et al. [2] proporciona una revisión de los principales modelos de DL para nubes LiDAR orientados a automoción, concretamente, a conducción autónoma. Generalmente, este tipo de aplicaciones deben ser capaces de funcionar en tiempo real, por lo que no solo es crucial la precisión de los modelos, sino también el rendimiento que presentan. En este contexto, una aplicación cumple la condición de tiempo real cuando el tiempo de procesamiento, T_{proc}, de cada *frame* de datos es menor que el intervalo entre adquisiciones consecutivas del sensor, T_{ad}, garantizando así una respuesta inmediata. El T_{ad} depende de la frecuencia de adquisición del sensor LiDAR, f_{LiDAR}, generalmente representada en hercios (Hz). Esta indica el número de escaneados completos o *frames* por unidad de tiempo que obtiene el sensor. Por tanto, la condición de tiempo real para aplicaciones que procesan datos LiDAR se puede resumir según la condición de la eq. (1).

$$T_{\text{proc}} \leq \frac{1}{f_{\text{LiDAR}}} \tag{1}$$

Este trabajo presenta una comparativa entre modelos de DL para nubes de puntos 3D, evaluando su precisión, rendimiento y consumo de recursos en la tarea de clasificación de objetos en tiempo real. La selección de modelos incluye a PointNet [1] y PointNet++ [3], dos propuestas pioneras y ampliamente utilizadas como base para la creación de nuevas arquitecturas. Además, se ha evaluado PointPillars [4], orientado a aplicaciones con mayores restricciones en términos de latencia, y PointRCNN [5], optimizado para la tarea de clasificación. En cuanto a los datos de entrada, se ha empleado el *dataset* KITTI [6], ampliamente utilizado. Los resultados permiten determinar qué modelos proporcionan un mejor equilibrio en base a las métricas estudiadas y a las características concretas de este estudio. Además, para la realización del análisis comparativo, se han llevado a cabo dos desarrollos que se encuentran disponibles en abierto. Por una parte, un *pipeline* para identificación, clasificación y seguimiento de objetos en tiempo real, el cual emplea el *framework Robot Operating System* 2 (ROS2) [7] para emular un sensor LiDAR y comunicar los distintos componentes

[1]Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), e-mails: pablo.liste@rai.usc.es, silvia.alcaraz@usc.es

[2]Departamento de Electrónica e Computación, Universidade de Santiago de Compostela (USC), e-mails: {oscar.garcia, jc.cabaleiro, ff.rivera}@usc.es

del sistema. Por otra parte, una herramienta para facilitar la comparación de modelos de DL en base a una serie de métricas configurables por el usuario.

II. Estado del arte

El procesamiento de nubes de puntos 3D mediante técnicas de aprendizaje profundo ha experimentado avances significativos en los últimos años, especialmente en aplicaciones de percepción para vehículos autónomos. Generalmente, estas aplicaciones deben operar en tiempo real para poder proporcionar respuestas inmediatas. Teniendo esto en cuenta, no sólo es fundamental prestar atención a la precisión de los modelos, si no también a aspectos relacionados con la latencia de la inferencia o el uso de recursos.

Revisando la literatura, pueden encontrarse trabajos que analizan y caracterizan el rendimiento computacional de sistemas que emplean DL con nubes de puntos 3D. En concreto, el trabajo de Uecker et al. [8] analiza la relación entre las representaciones utilizadas con nubes de puntos 3D en redes neuronales y su rendimiento. Para ello, proponen una taxonomía que permite identificar las ventajas y desventajas de cada representación en términos de su eficiencia computacional, uso de recursos, y precisión en la tarea de clasificación. La propuesta de Lin et al. [9] no sólo caracteriza los cuellos de botella de los modelos para nubes de puntos 3D en varios dispositivos, si no que también proponen un acelerador de aprendizaje profundo para nubes de puntos. Otros trabajos que también buscan mejorar la eficiencia computacional de estos sistemas incluyen el de Bai et al. [10], donde se implementa PointNet en una *Field Programmable Gate Array* (FPGA) para acelerar el modelo y lograr tiempo real, o el de Tang et al. [11], que presenta un motor de inferencia optimizado para acelerar redes neuronales que procesan nubes de puntos 3D usando convoluciones *sparse*. Otros enfoques se centran en la implementación de arquitecturas específicamente diseñadas para reducir la latencia de la inferencia preservando la precisión [12], [13], [14], [15]. Para facilitar la optimización y el diseño de estas redes, surgieron las soluciones basadas en *Neural Architecture Search* (NAS) [16], que automatizan el proceso de diseño buscando configuraciones que se adapten a unos criterios dados. Tradicionalmente, estos criterios se limitaban a aspectos relacionados con maximizar la precisión. Para abordar esta limitación, Li et al. [17] proponen un NAS que incorpora explícitamente la latencia como una restricción durante el proceso de búsqueda de arquitecturas. La importancia de preservar un rendimiento óptimo en este tipo de aplicaciones también alcanza el ámbito de la seguridad. Prueba de esto es el estudio de Liu et al. [18], donde investigan qué perturbaciones se pueden introducir para aumentar la latencia sin afectar a la calidad de la predicción en sistemas de detección basados en LiDAR y DL.

En conjunto, estos trabajos reflejan el interés existente, no sólo en mejorar la precisión de los modelos de DL para nubes de puntos 3D, sino también en caracterizar y optimizar su rendimiento computacional, dada su importancia en aplicaciones sensibles en latencia y uso de recursos.

III. Materiales y entorno de pruebas

Esta sección describe tanto el *software* como el *hardware* empleados en este trabajo.

A. Software

El desarrollo y las pruebas se llevaron a cabo en un equipo con Ubuntu 22.04 LTS, empleando Python 3.10 junto con otras librerías y *frameworks*, entre ellas: PyTorch 2.0.1, CUDA Toolkit 12.8, Scikit-learn 1.6.1, NumPy 1.26.4 y Pandas 2.2.3. También se utilizó la *toolbox* de OpenPCDet [19] por su soporte de modelos de DL preentrenados para nubes de puntos 3D. Por otra parte, el *pipeline* de procesamiento de datos LiDAR en tiempo real se implementó con el *framework* ROS2, versión *Humble*, usando Open3D [20] para el manejo de los datos. Por último, el *dataset* KITTI fue escogido como base para el entrenamiento y la validación de los modelos.

B. Hardware

Todos los experimentos se ejecutaron en una única estación de trabajo equipada con un procesador Intel Core i7-6700K, 32 GB de RAM DDR4 3200 MHz y una GPU NVIDIA GeForce GTX 1070 de 8 GB.

IV. Modelos DL para nubes de puntos

A. Modelos analizados

Para este estudio comparativo, se han seleccionado cuatro modelos representativos en el procesamiento de nubes de puntos 3D mediante *Deep Neural Networks* (DNNs), cada uno con características arquitectónicas distintivas y diferentes enfoques para manejar datos LiDAR. Esta selección proporciona un espectro de las arquitecturas utilizadas en este campo, permitiendo analizar distintos compromisos entre precisión y rendimiento computacional. La tabla I resume las principales características de los modelos seleccionados.

PointNet [1] es un enfoque pionero que procesa nubes de puntos 3D directamente sin conversión a vóxeles o proyecciones 2D. Su arquitectura emplea redes totalmente conectadas para cada punto y una subred de transformación espacial para alineación geométrica. Su innovación principal es el uso de funciones simétricas que garantizan invariancia al orden de los puntos, solucionando así la naturaleza no estructurada de estos datos.

PointNet++ [3] mejora su predecesor incorporando un enfoque jerárquico que captura características locales a múltiples escalas. Aplica recursivamente redes tipo PointNet a subconjuntos definidos mediante técnicas como *farthest point sampling* y *ball query*, permitiendo analizar tanto información local como global para una mejor comprensión de las relaciones espaciales entre puntos.

PointPillars [4] está optimizado para aplicaciones en tiempo real, transformando la nube de puntos

en "*pillars*" (columnas verticales en rejilla 2D) para aprovechar operaciones de convolución 2D eficientes. Usa una red inspirada en PointNet para extraer características de cada pilar, que luego reorganiza en vista de pájaro (*Bird's Eye View*) para procesamiento mediante redes convolucionales convencionales.

PointRCNN [5] implementa detección en dos etapas: primero predice puntos de primer plano y genera propuestas iniciales de cuadros delimitadores 3D directamente desde la nube de puntos; después refina estas propuestas transformando los puntos a coordenadas canónicas centradas en cada caja, permitiendo estimaciones más precisas de posición, dimensiones y orientación de objetos.

B. Entrenamiento de modelos

Dado que no se encontraron versiones preentrenadas de PointNet y PointNet++ compatibles con el *dataset* KITTI, fue necesario entrenarlos desde cero. Esto implicó una preparación adicional del *dataset*, así como ajustes específicos sobre los modelos para mitigar el sobreajuste y garantizar una evaluación rigurosa de su rendimiento. Por la contra, PointPillars y PointRCNN cuentan con implementaciones preentrenadas disponibles públicamente para KITTI. En este trabajo se optó por utilizar las versiones provistas por el *framework* OpenPCDet, lo que permitió su uso directo sin necesidad de entrenamiento desde cero.

El *dataset* KITTI incluye nubes de puntos generadas por sensores LiDAR, junto con anotaciones 3D de los objetos presentes en la escena. Para el entrenamiento de los modelos comentados, se desarrolló el código necesario para transformar cada escena del *dataset* en muestras individuales de objetos. Este proceso extrae información clave como la nube de puntos asociada a cada objeto, su clase, dimensiones y posición en el espacio. Además, se aplicaron las matrices de calibración para alinear las coordenadas, se generaron las cajas delimitadoras 3D, se recortaron las nubes de puntos de cada objeto y se filtraron aquellas con una cantidad de puntos inferior a un umbral dado. Cada muestra fue almacenada como un diccionario estructurado que facilita su uso posterior en el entrenamiento.

Una vez generadas las muestras individuales, estas se dividieron en conjuntos de entrenamiento, validación y prueba, aplicando separación estratificada sobre el conjunto de entrenamiento para preservar la proporción de clases. Para acelerar el acceso a los datos, se implementó un sistema de almacenamiento en formato `pickle` [21], lo que permitió reducir significativamente los tiempos de carga durante las ejecuciones sucesivas.

Durante el proceso de entrenamiento se incorporaron técnicas de *data augmentation* con el objetivo de mejorar la generalización de los modelos. Entre estas técnicas se incluyeron rotaciones aleatorias, escalado y *jittering*. Asimismo, las nubes de puntos fueron normalizadas y escaladas, mejorando así el entrenamiento.

Dado el desequilibrio de clases, se aplicó una ponderación en la función de pérdida, ajustando la penalización según la frecuencia relativa de cada clase.

El ciclo de entrenamiento comprende varios pasos: configuración del dispositivo de cómputo (CPU o GPU), carga de datos con muestreo balanceado, inicialización del modelo (PointNet o PointNet++), y configuración del optimizador Adam [22]. Se empleó la función de pérdida `CrossEntropyLoss` con pesos para las clases. En cada *epoch*, el modelo realizó el proceso de inferencia, se calculó la pérdida, se realizó retropropagación y se actualizaron los parámetros. Al final de cada *epoch*, se ajustó la tasa de aprendizaje, se evaluó el rendimiento en el conjunto de validación y se guardó un *checkpoint* del modelo si se obtenía una mejora en el F1 macro.

C. Diseño de los experimentos

En el contexto de la clasificación de objetos en nubes de puntos LiDAR para aplicaciones de automoción en tiempo real, la evaluación sistemática del rendimiento de los modelos requiere métricas específicas que proporcionen una caracterización completa tanto de la capacidad de inferencia como de la eficiencia computacional.

Las métricas adoptadas en este estudio para valorar el desempeño de los modelos incluyeron:

- Exactitud (*Accuracy*).
- Precisión por clase, macro y ponderada.
- Exhaustividad (*Recall*) por clase, macro y ponderada.
- Puntuación F1 por clase, macro y ponderada.
- Matriz de confusión.

En cuanto a la evaluación de la eficiencia, se han considerado las siguientes métricas:

- Tiempo de inferencia por escena y por objeto.
- Desviación estándar del tiempo de inferencia.
- Uso de memoria por escena y por objeto.
- Desviación estándar del uso de memoria.
- Huella de memoria del modelo.

La selección de las métricas de evaluación empleadas ha sido guiada por una serie de criterios que aseguran una correcta valoración del rendimiento de los modelos. Estos criterios se detallan a continuación:

- **Evaluación multidimensional**: La combinación de métricas como *accuracy*, *precision*, *recall* o F1 proporcionan una caracterización completa del rendimiento y comportamiento del modelo.
- **Análisis por clase**: Identifica debilidades específicas en clases críticas.
- **Rendimiento computacional**: Determina la viabilidad de su uso en sistemas con bajos recursos o en aplicaciones que funcionan en tiempo real.
- **Reproducibilidad y comparabilidad**: Facilita las comparaciones con otros modelos y estudios.

Tabla I: Comparativa de características principales de los modelos seleccionados.

Modelo	Representación	Ventajas principales	Limitaciones
PointNet	Puntos en bruto	Simplicidad y rapidez	No modela relaciones locales
PointNet++	Puntos en bruto	Captura relaciones locales y globales	Coste computacional
PointPillars	Pilares	Velocidad de inferencia	Pérdida de resolución
PointRCNN	Puntos en bruto	Precisión	Complejidad y latencia

D. Método para evaluar los modelos

El sistema de comparación de modelos implementado[1] constituye un marco completo para la evaluación sistemática de modelos de DL diseñados para el procesamiento de nubes de puntos 3D. Su implementación sigue un paradigma modular que facilita la incorporación de nuevos modelos y métricas, dividiéndose en varias fases clave.

El proceso comienza con la carga dinámica de modelos preentrenados a partir de archivos de *checkpoint*, detectando automáticamente la arquitectura mediante el análisis del nombre del archivo y adaptándose a los requisitos específicos de cada modelo. Durante esta etapa se monitoriza la huella de memoria con el módulo `tracemalloc`, y se configura el *dataset* KITTI utilizando las capacidades de OpenPCDet para una carga eficiente de nubes de puntos y sus anotaciones.

A fin de garantizar la consistencia entre diferentes arquitecturas, se aplica un *pipeline* de preprocesamiento, estructurado en varias etapas, que comienza con una segmentación por objeto, lo que permite delimitar los puntos correspondientes a cada objeto de la escena. Posteriormente, cada agrupación se normaliza mediante el centrado espacial por sustracción del centroide, se escala inscribiéndola en una esfera unitaria y se regula la densidad para mantener exactamente 1024 puntos por objeto, adaptando finalmente el formato tensorial requerido por cada arquitectura y transfiriéndolo al dispositivo de cómputo designado.

Una vez preparados los datos, se inicia el proceso de inferencia, durante el cual se procesa cada objeto identificado en la escena. Para cada instancia, se registran los tiempos de inicio y finalización con el fin de calcular la latencia y se estima el consumo de memoria. Simultáneamente, se extraen la clase predicha y la etiqueta correspondiente al objeto (*ground truth*) a partir de las anotaciones, almacenándose ambas para su evaluación posterior. Finalizado el procesamiento de todos los objetos de la escena, las predicciones y etiquetas obtenidas se introducen en estructuras de datos optimizadas que consolidan los resultados a nivel de escena completa. La evaluación del rendimiento se realiza posteriormente mediante funciones especializadas que calculan métricas tanto a nivel global como por clase, utilizando promedios ponderados y no ponderados, comentadas anteriormente.

El sistema incorpora mecanismos de protección para garantizar la estabilidad durante las evaluaciones, como bloques `try-except` que capturan excepciones, gestión de memoria mediante eliminación de variables temporales, activación del recolector de basura y liberación de memoria GPU. De igual forma, se asegura la finalización correcta de procesos de monitorización incluso en presencia de errores, e incluye verificaciones para condiciones especiales como objetos con pocos puntos o resultados no válidos.

Para el análisis de resultados, el sistema genera informes estructurados con todas las métricas y los exporta a formato CSV, manejando la creación de directorios y encabezados, y complementa esta salida con recursos gráficos de matrices de confusión.

La coordinación del proceso está a cargo del método `run_benchmark`, que garantiza pruebas bajo condiciones controladas, verifica los directorios necesarios, itera sobre los modelos, asegura parámetros homogéneos, consolida resultados y genera los archivos de salida correspondientes.

Todo el proceso sigue una secuencia definida iniciada por la clase `ModelComparer`, donde se establecen parámetros como el dispositivo, la ruta a los datos y los directorios de salida. También se cargan y preparan los modelos entrenados manualmente, como PointNet y PointNet++, y los modelos preentrenados de OpenPCDet, como PointPillars y PointRCNN, configurando el acceso a las escenas mediante la API adecuada.

Cada modelo es evaluado en 10 ejecuciones independientes para mitigar efectos externos y detectar inestabilidades, procesando todas las escenas del conjunto de validación, aplicando el preprocesamiento, realizando inferencias y registrando resultados. Finalmente, se realiza un análisis comparativo que incluye promedios, desviaciones estándar y métricas agregadas, permitiendo cuantificar los compromisos entre precisión y eficiencia de cada arquitectura y facilitando la selección del modelo más adecuado según se priorice la precisión, la latencia o un equilibrio entre ambas.

V. Pipeline de procesamiento en tiempo real

Además del análisis comparativo entre modelos basado en las métricas previamente presentadas, se ha desarrollado un *pipeline* de procesamiento enfocado al estudio del proceso de clasificación en un entorno de ejecución en tiempo real. Este *pipeline* permite evaluar el rendimiento de los modelos en condiciones representativas de escenarios reales, lo que facilita un análisis más preciso de su comportamiento, capacidad de respuesta y robustez frente a flujos de datos continuos y dinámicos.

A. Introducción a ROS2

El funcionamiento del *framework* ROS2 se basa en una arquitectura distribuida y orientada a men-

[1] `https://github.com/ppabli/cmet`

sajes. Sus componentes fundamentales, denominados nodos, son procesos independientes encargados de tareas específicas. La comunicación entre nodos se realiza mediante dos modelos principales: publicación/-suscripción, donde los *publish nodes* envían mensajes a canales denominados *topics* y los *suscribe nodes* los reciben; y cliente/servidor, que incluye servicios para interacciones síncronas y acciones para operaciones complejas o de larga duración que permiten enviar objetivos, recibir retroalimentación y cancelar tareas. Una característica distintiva de ROS2 es el uso del estándar *Data Distribution Service* (DDS) como *middleware*, proporcionando capacidades avanzadas como descubrimiento automático de nodos, configuración de calidad de servicio (QoS) y comunicación eficiente en redes distribuidas.

B. Arquitectura y componentes

El sistema desarrollado, y disponible públicamente en Github[2], presenta un *pipeline* parametrizado de procesamiento para datos provenientes de sensores LiDAR. Este sistema está implementado como un nodo en el *framework* ROS2, diseñado para procesar nubes de puntos tridimensionales en tiempo real. El objetivo principal del *pipeline* es detectar, clasificar y realizar seguimiento de objetos en un entorno dinámico, utilizando técnicas de procesamiento de nubes de puntos en combinación con aprendizaje automático.

El *pipeline* se estructura en varios componentes fundamentales que trabajan para transformar los datos crudos del sensor en información procesable sobre los objetos presentes en el entorno. La arquitectura, de enfoque modular, permite una clara separación de responsabilidades entre los distintos componentes del sistema. En concreto, el flujo del sistema se representa en la figura 1.

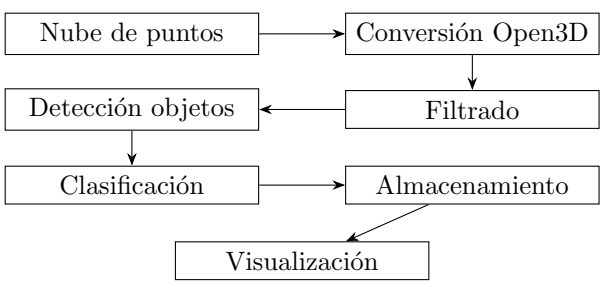

Fig. 1: Esquema del *pipeline* desarrollado.

El flujo de datos se inicia con la recepción de mensajes procedentes del sensor LiDAR, los cuales contienen coordenadas tridimensionales que describen las superficies de los objetos detectados. Estos datos son posteriormente procesados mediante una serie de filtros, tras lo cual se lleva a cabo la identificación de objetos individuales. A continuación, cada objeto es clasificado utilizando un modelo DNN, y finalmente se realiza el proceso de seguimiento a lo largo del tiempo a través de múltiples *frames*.

El componente central del sistema es la clase `LidarProcessor`, que implementa un nodo de ROS2 encargado de orquestar todo el *pipeline* de procesamiento. Este nodo se subscribe al *topic* que transmite los datos del sensor LiDAR y expone diferentes *topics* de salida para las nubes de puntos filtradas y las *bounding boxes* de los objetos detectados. Además, se encarga de inicializar varios componentes clave, entre ellos: el modelo de clasificación, las estructuras de datos para el seguimiento de objetos y los parámetros configurables que controlan el comportamiento del *pipeline*. Estos parámetros incluyen las restricciones de tiempo, umbrales para la detección y seguimiento de objetos, configuraciones para la asociación de datos, así como opciones para habilitar o deshabilitar ciertas funcionalidades como la generación de *bounding boxes* o el cálculo de velocidad de los objetos detectados.

B.1 Procesamiento de nubes de puntos

El procesamiento de nubes de puntos comienza con la recepción de un mensaje del sensor, lo que activa el *pipeline* mediante la conversión del formato ROS2 PointCloud2 a estructuras compatibles con Open3D. A continuación, se realiza un submuestreo por voxelización, reduciendo la densidad de puntos sin comprometer la geometría. Posteriormente, se elimina el suelo utilizando el algoritmo RANSAC [23], seguido de un filtrado estadístico para descartar valores atípicos o ruido. Finalmente, se aplica un recorte espacial que restringe el análisis a una región de interés específica. Este conjunto de etapas transforma los datos en bruto en información estructurada, mejorando la eficiencia computacional sin perder fidelidad en la representación necesaria para la detección de objetos.

B.2 Detección y segmentación de objetos

Esta tarea se realiza mediante el algoritmo DBSCAN [24] el cual agrupa puntos espacialmente próximos entre sí, formando *clusters* que representan objetos en la escena. Para cada *cluster* identificado, el sistema crea una instancia de `TrackedObject`, que almacena la nube de puntos correspondiente, su centroide, marca temporal y características extraídas mediante descriptores *Fast Point Feature Histograms* (FPFH), facilitando la identificación y seguimiento del objeto a lo largo del tiempo.

B.3 Clasificación de objetos

El modelo recibe nubes de puntos normalizadas de los objetos segmentados y produce probabilidades para diferentes categorías. El proceso se realiza por lotes, para optimizar el uso de recursos, asignando a cada objeto la etiqueta correspondiente a la categoría más probable. Estas etiquetas se utilizan para filtrar objetos de interés y proporcionar información semántica durante el seguimiento.

B.4 Seguimiento de objetos

El seguimiento temporal de objetos mantiene la identidad de estos a lo largo del tiempo mediante

[2] `https://github.com/ppabli/rdltracker`

una combinación del algoritmo húngaro para asociación de datos y filtros de Kalman para estimación de estado. Cada objeto seguido almacena un vector de estado (posición y velocidad en 3 ejes) que se actualiza mediante predicción y corrección. La clase `TrackedObject` gestiona el estado dinámico, manteniendo historiales de movimiento y usando identificadores únicos para cada objeto. El sistema añade nuevos objetos cuando aparecen, mantiene temporalmente los no detectados para manejar oclusiones y elimina aquellos ausentes durante períodos prolongados. La asociación entre objetos se basa en una matriz de costes que combina distancias y características, estableciendo solo asociaciones por debajo de un umbral definido.

B.5 Publicación de resultados

El resultado final del *pipeline* se publica en forma de mensajes ROS2 que pueden ser consumidos por otros nodos del sistema. Se generan dos tipos principales de salidas: nubes de puntos filtradas, que contienen solo los puntos correspondientes a objetos de interés, y marcadores visuales para los objetos detectados que muestran información adicional como el identificador del objeto, su etiqueta de clase y su velocidad estimada. La figura 2 ejemplifica el resultado obtenido.

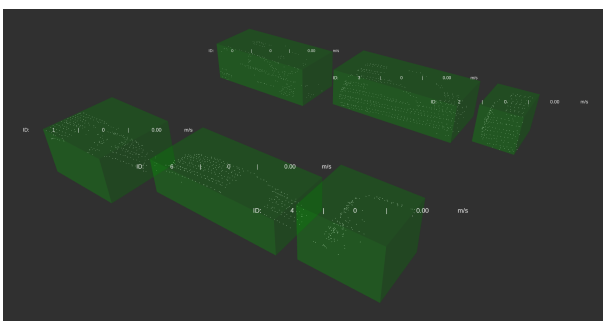

Fig. 2: Resultado del *pipeline*.

C. Optimizaciones

El *pipeline* ha sido diseñado considerando restricciones de tiempo real, fundamentales en aplicaciones como la navegación autónoma o la interacción con entornos dinámicos. Para ello, se emplean operaciones tensoriales optimizadas, se minimizan las conversiones entre formatos, se utiliza procesamiento por lotes durante la etapa de clasificación y se aplica submuestreo adaptativo. Estas estrategias permiten alcanzar un equilibrio eficaz entre precisión y eficiencia computacional adaptable al *hardware* disponible.

VI. Resultados experimentales

En esta sección se describen los resultados obtenidos en los experimentos relacionados con la medición de consumo de memoria, precisión y tiempos de inferencia de los modelos para la tarea de clasificación de objetos. Además, también se evalúa su desempeño en el contexto de una aplicación que procesa datos LiDAR en tiempo real.

A. Análisis del consumo de recursos

La tabla II presenta un análisis comparativo del consumo de memoria entre los modelos evaluados, considerando dos métricas clave: el tamaño del modelo tras su carga y el consumo promedio de memoria por objeto procesado. Estas métricas permiten evaluar tanto la eficiencia del modelo durante su inicialización como su comportamiento en ejecución continua.

Tabla II: Consumo de memoria por modelo.

Modelo	Mem. objeto (MB)	Tam. modelo (MB)
PointNet	2.215	0.703
PointNet++	2.825	0.614
PointPillars	0.931	47.142
PointRCNN	0.825	46.425

PointNet y PointNet++ exhiben una huella de memoria significativamente reducida en cuanto al tamaño del modelo, resultado de su diseño compacto, el uso de operaciones simétricas globales y la ausencia de módulos convolucionales pesados. Estas arquitecturas operan directamente sobre nubes de puntos no estructuradas, sin requerir voxelización ni representaciones intermedias, lo cual se traduce en una estructura simple y altamente eficiente.

En contraste, PointPillars y PointRCNN requieren una cantidad considerablemente mayor de memoria para su carga. Esta diferencia se explica por la presencia de componentes complejos, como convoluciones 2D/3D, *backbones* profundos, módulos de detección en múltiples etapas y representaciones espaciales jerárquicas. Además, el uso de *frameworks* modulares como OpenPCDet, aunque favorece la extensibilidad y precisión, introduce una sobrecarga adicional en términos de consumo de memoria. A diferencia de estos, PointNet y PointNet++ fueron implementados de forma monolítica, centralizando la configuración en un único archivo, lo cual minimiza la ocupación de memoria a costa de perder modularidad e interoperabilidad de sus componentes.

Respecto al consumo de memoria por objeto durante la inferencia, se observa un comportamiento inverso: los modelos más complejos (PointPillars y PointRCNN) muestran una mayor eficiencia. Este fenómeno puede atribuirse al uso de técnicas de optimización como el procesamiento por lotes, la reutilización de *buffers* y tensores intermedios y la ejecución paralela en GPUs. Asimismo, la codificación espacial jerárquica y la compresión de características contribuyen a reducir el volumen de datos temporales por muestra. Por el contrario, PointNet y PointNet++, al carecer de representaciones espaciales explícitas y depender en mayor medida de operaciones independientes por punto, tienden a replicar estructuras internas para cada instancia, lo cual incrementa su consumo relativo de memoria en tiempo de ejecución.

B. Análisis de tiempos de inferencia

La tabla III contiene los tiempos promedio de inferencia por objeto y escena, una métrica fundamen-

tal en aplicaciones donde las restricciones de latencia son críticas, como en robótica, vehículos autónomos o sistemas de *edge computing*.

Tabla III: Tiempos de inferencia por modelo.

Modelo	Objeto (S)	Escena (S)
PointNet	0.005	0.019
PointNet++	0.159	0.612
PointPillars	0.026	0.099
PointRCNN	0.266	1.030

PointNet se posiciona como el modelo más eficiente en términos de tiempo de inferencia. Esto se debe a su arquitectura minimalista, que omite el modelado de relaciones espaciales locales. Al operar directamente sobre nubes de puntos sin estructura, mediante funciones de agregación simétricas, el modelo produce representaciones invariantes al orden con un número reducido de parámetros, lo que permite una paralelización efectiva en GPU.

PointPillars, caracterizado por una arquitectura orientada a la eficiencia computacional, es el segundo modelo con mejores tiempos de inferencia. Su rendimiento proviene de la conversión de datos 3D en una representación basada en "*pillars*", lo que permite emplear redes convolucionales 2D, reduciendo significativamente la carga computacional. Este diseño favorece además la compatibilidad con arquitecturas aceleradas modernas y promueve una paralelización más eficiente.

En contraste, PointNet++, aunque mejora la capacidad de modelar estructuras locales mediante una jerarquía espacial, incurre en mayores costes computacionales. El uso de agrupamiento a múltiples escalas, búsqueda de vecinos y extracción de características introducen pasos secuenciales difíciles de paralelizar, particularmente en *hardware* sin soporte nativo para búsquedas espaciales eficientes.

Finalmente, PointRCNN presenta el mayor tiempo de inferencia, limitando su capacidad de procesamiento a menos de un objeto por segundo. Esta latencia es coherente con su enfoque de detección en dos etapas: la primera genera propuestas 3D mediante segmentación semántica, mientras que la segunda realiza un refinamiento de las cajas predichas mediante agregación local. Ambas fases, altamente dependientes entre sí, restringen la paralelización y aumentan el coste computacional.

C. Análisis de precisión

La tabla IV presenta un resumen comparativo de las métricas de rendimiento asociadas a cada uno de los modelos evaluados. Se incluyen medidas de precisión, *recall* y F1, tanto en su versión macro como ponderada, con el objetivo de proporcionar una evaluación integral de la capacidad de clasificación de estos.

Aunque todos los modelos han sido sometidos al mismo *pipeline* de evaluación, cabe destacar que difieren sustancialmente en sus características arquitectónicas: algunos están diseñados específicamente para tareas de clasificación, mientras que otros priorizan funcionalidades más complejas como la detección de objetos. Esta divergencia de propósitos debe ser tenida en cuenta al interpretar los resultados.

PointRCNN destaca como el modelo con mayor precisión en todas las métricas evaluadas. Esto refleja la efectividad de su arquitectura basada en etapas y refinamiento local, diseñada específicamente para una detección de objetos altamente precisa.

PointPillars obtiene un rendimiento notablemente similar al de PointRCNN, a pesar de su menor complejidad arquitectónica. Este resultado indica que su diseño basado en "*pillars*" captura la mayoría de las estructuras discriminativas relevantes. De esta forma, y teniendo en cuenta los resultados de la sección VI-B, se constituye como una alternativa eficaz en contextos donde la eficiencia computacional también es prioritaria.

PointNet mantiene un rendimiento elevado, lo que resalta la capacidad de su mecanismo de agregación global para capturar patrones relevantes a pesar de su simplicidad. Además, su bajo número de parámetros y su estructura determinista favorecen la generalización, particularmente en conjuntos de datos bien estructurados.

Por último, PointNet++ muestra el rendimiento más bajo. Teniendo en cuenta los resultados de su predecesor, PointNet, esta limitación puede estar asociada a una configuración subóptima de hiperparámetros, a la pérdida de información en el muestreo jerárquico, o a una menor adecuación al dominio del *dataset* utilizado. Su sensibilidad a la densidad de los puntos y la variabilidad espacial también podrían haber afectado negativamente a su desempeño en este escenario experimental.

D. Análisis del procesamiento en tiempo real

Evaluando la aplicación descrita en la sección V, se ha identificado que para que cumpla con los requisitos de operación en tiempo real, es indispensable contar con aceleración mediante GPU en las tareas con mayor carga computacional. En concreto, esas tareas están relacionadas con los procesos de inferencia llevados a cabo durante la fase de clasificación. Cabe destacar que el resto de funcionalidades del *pipeline* desarrollado logran ejecutarse en tiempo real empleando únicamente la CPU, gracias a las estrategias de optimización y a las estructuras de datos escogidas. En nuestro caso de estudio, la condición de tiempo real viene dada por las características del sensor emulado, el Velodyne HDL-64E [25], el cual realiza giros a una velocidad de 10 *frames* por segundo.

Este análisis permite inferir las restricciones de *hardware* que serían necesarias en un escenario real y comparable al planteado, sirviendo como referencia para futuras implementaciones.

VII. Conclusiones

En este estudio se ha llevado a cabo un análisis y evaluación del rendimiento de diversos modelos de DL aplicados al proceso de clasificación de nubes de puntos 3D, utilizando el *dataset* KITTI. La eva-

Tabla IV: Métricas de precisión de los modelos evaluados.

Modelo	*Accuracy*	Precisión (M)	*Recall* (M)	F1 (M)	Precisión (W)	*Recall* (W)	F1 (W)
PointNet	0.989	0.964	0.953	0.958	0.989	0.989	0.989
PointNet++	0.932	0.812	0.746	0.774	0.930	0.932	0.929
PointPillars	0.994	0.966	0.983	0.974	0.994	0.994	0.994
PointRCNN	0.997	0.986	0.981	0.984	0.997	0.997	0.997

luación integral de los modelos a partir de distintas métricas ha permitido identificar fortalezas y debilidades específicas de cada arquitectura. Los resultados evidencian patrones de comportamiento diferenciados que pueden orientar la selección del modelo más adecuado en función de los requisitos particulares de cada implementación.

PointNet se posiciona como el modelo más ligero y rápido, consecuencia directa de su diseño simplificado. Esta característica lo convierte en un candidato atractivo para implementaciones donde los recursos *hardware* son limitados o donde los requisitos de latencia son estrictos. Sin embargo, esta ventaja no compromete significativamente su capacidad predictiva, que se mantiene en niveles competitivos respecto a alternativas más complejas, al menos en este contexto de estudio.

PointPillars mostró un buen equilibrio entre precisión y eficiencia computacional, destacando por sus rápidos tiempos de inferencia y un consumo de memoria moderado. Su diseño arquitectónico le permite realizar inferencias de manera más eficiente que modelos como PointNet++ y PointRCNN. En nuestro contexto experimental, PointPillars resultó ser una opción sólida cuando se busca un buen desempeño con una menor carga computacional.

PointRCNN alcanzó la mejor precisión en todas las métricas evaluadas, lo que lo convierte en el modelo más preciso en el marco de este estudio. Sin embargo, su mayor coste computacional y sus elevados tiempos de inferencia lo hacen menos adecuado en escenarios donde la eficiencia en tiempo real es crucial. Su arquitectura de doble etapa, aunque efectiva, introduce limitaciones en términos de paralelización y velocidad.

PointNet++ plantea interrogantes respecto a su adecuación para este tipo de datos, mostrando un rendimiento inferior al esperado considerando su capacidad teórica. Este comportamiento sugiere que su efectividad puede depender de parámetros específicos de implementación o de características particulares del *dataset*, lo que destaca la necesidad de realizar futuras revisiones y ajustes.

Por otra parte, el *pipeline* desarrollado permitió determinar que, en nuestro escenario de procesamiento en tiempo real, todos los modelos requieren aceleración mediante GPU para completar la inferencia dentro de los límites temporales establecidos.

Las observaciones y resultados derivados de este estudio subrayan la importancia de considerar múltiples dimensiones en la evaluación de modelos. La elección entre arquitecturas debe fundamentarse en un análisis de los distintos *trade-offs*, siempre en relación con los requisitos específicos de cada contexto de aplicación. Estos hallazgos, aunque válidos para el dominio experimental considerado, requieren de validación adicional para extrapolarse a otros escenarios con características diferentes.

VIII. Trabajo futuro

Los resultados obtenidos en este estudio abren diversas líneas para investigaciones futuras. Se propone profundizar en el análisis del rendimiento subóptimo observado en PointNet++, mediante estudios orientados a la exploración sistemática de hiperparámetros y cambios a nivel de arquitectura del modelo. De igual forma, ampliar el análisis incorporando nuevas arquitecturas o modelos permitiría establecer un *benchmark* más representativo y exhaustivo. De manera complementaria, también sería interesante incluir el análisis de modelos sobre los que se han aplicado técnicas avanzadas de optimización, como cuantización o poda, y su comparación con el modelo base. Asimismo, se considera la evaluación del comportamiento de los modelos en distintas plataformas *hardware*, en particular, dispositivos orientados a *edge computing* o aceleradores especializados. Este enfoque permitiría explorar su aplicabilidad en entornos reales con restricciones operativas.

Agradecimientos

Este trabajo ha recibido apoyo financiero de la Consellería de Cultura, Educación e Ordenación Universitaria (acreditaciones ED431C-2022/16, ED431G-2019/04), que reconoce al CiTIUS-Centro Singular de Investigación en Tecnologías Inteligentes como un Centro de Investigación del sistema universitario gallego, y del Fondo Europeo de Desarrollo Regional (ERDF). Así mismo, el Ministerio de Economía y Competitividad, Gobierno de España (subvenciones PID2019-104834GB-I00 y PID2022-141623NB-I00) ha proporcionado su apoyo.

Referencias

[1] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas, "Pointnet: Deep learning on point sets for 3D classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[2] Ying Li, Lingfei Ma, Zilong Zhong, Fei Liu, Michael A Chapman, Dongpu Cao, and Jonathan Li, "Deep learning for LiDAR point clouds in autonomous driving: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 8, pp. 3412–3432, 2020.

[3] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in neural information processing systems*, vol. 30, 2017.

[4] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 12697–12705.

[5] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li, "PointRCNN: 3D object proposal generation and detection from point cloud," in *Proceedings of the IEEE/CVF*

conference on computer vision and pattern recognition, 2019, pp. 770–779.

[6] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun, "Vision meets robotics: The KITTI Dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[7] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, pp. eabm6074, 2022.

[8] Marc Uecker, Tobias Fleck, Marcel Pflugfelder, and J Marius Zöllner, "Analyzing deep learning representations of point clouds for real-time in-vehicle lidar perception," *arXiv preprint arXiv:2210.14612*, 2022.

[9] Yujun Lin, Zhekai Zhang, Haotian Tang, Hanrui Wang, and Song Han, "Pointacc: Efficient point cloud accelerator," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 449–461.

[10] Lin Bai, Yecheng Lyu, Xin Xu, and Xinming Huang, "Pointnet on FPGA for real-time lidar point cloud processing," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2020, pp. 1–5.

[11] Haotian Tang, Zhijian Liu, Xiuyu Li, Yujun Lin, and Song Han, "Torchsparse: Efficient point cloud inference engine," *Proceedings of Machine Learning and Systems*, vol. 4, pp. 302–315, 2022.

[12] Y Wang, T Shi, P Yun, L Tai, and M Liu, "Pointseg: Real-time semantic segmentation based on 3D lidar point cloud.," *arXiv preprint arXiv:1807.06288*, 2018.

[13] Andres Milioto, Ignacio Vizzo, Jens Behley, and Cyrill Stachniss, "Rangenet++: Fast and accurate LiDAR semantic segmentation," in *2019 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2019, pp. 4213–4220.

[14] Tiago Cortinhal, George Tzelepis, and Eren Erdal Aksoy, "Salsanext: Fast, uncertainty-aware semantic segmentation of lidar point clouds," in *Advances in Visual Computing: 15th International Symposium, ISVC 2020, San Diego, CA, USA, October 5–7, 2020, Proceedings, Part II 15*. Springer, 2020, pp. 207–222.

[15] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham, "Randla-net: Efficient semantic segmentation of large-scale point clouds," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11108–11117.

[16] Barret Zoph and Quoc V Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[17] Guohao Li, Mengmeng Xu, Silvio Giancola, Ali Thabet, and Bernard Ghanem, "LC-NAS: Latency constrained neural architecture search for point cloud networks," in *2022 International Conference on 3D Vision (3DV)*. IEEE, 2022, pp. 1–11.

[18] Han Liu, Yuhao Wu, Zhiyuan Yu, Yevgeniy Vorobeychik, and Ning Zhang, "Slowlidar: Increasing the latency of lidar-based detection using adversarial examples," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 5146–5155.

[19] OpenPCDet Development Team, "OpenPCDet: An Open-source Toolbox for 3D Object Detection from Point Clouds," `https://github.com/open-mmlab/OpenPCDet`, 2020.

[20] "Open3D - A Modern Library for 3D Data Processing," `https://www.open3d.org/`, última consulta: 07/05/25.

[21] "pickle - Python object serialization," `https://docs.python.org/es/3.13/library/pickle.html`, última consulta: 07/05/25.

[22] PyTorch Developers, "Adam algorithm - PyTorch documentation," `https://pytorch.org/docs/stable/generated/torch.optim.Adam.html`, última consulta: 07/05/25.

[23] Martin A. Fischler and Robert C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, 1981.

[24] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al., "A density-based algorithm for discovering clusters in large spatial databases with noise," in *kdd*, 1996.

[25] "The KITTI Vision Benchmark Suite: Sensor setup," `https://www.cvlibs.net/datasets/kitti/setup.php`, última consulta: 08/05/25.