LiDAR-Based Animal Recognition from Point Cloud Signatures

Joint Clustering, Tracking, and Temporal Classification on LiDAR Scans

Presented by:
Nicholas Tucker
TCKNIC006

Prepared for:
Dr Stacy Shield
Dept. of Electrical and Electronics Engineering
University of Cape Town

Submitted to the Department of Electrical Engineering at the University of Cape Town in partial fulfilment of the academic requirements for a Bachelor of Science degree in Mechatronics Engineering.

October 27, 2025

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.
3. This report is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.

Nicholas Tucker

Date: October 27, 2025

Word Count: 20305

Acknowledgements

As I close this chapter of my life and prepare to move overseas, I am profoundly grateful for the four years I have spent studying in Cape Town. These years have been more than an academic journey. They have moulded my character, deepened my resilience, and surrounded me with people who have enriched every part of the experience.

I would like to express my deepest gratitude to my supervisor, Dr. Stacy Shield, for her guidance, patience, and insight throughout this project. Her advice and regular meetings helped me to refocus and refine my approach, especially under the logistical constraints that influenced this study.

My sincere thanks also go to my co-supervisor, Dr. Amir Patel, for his experience in the field, his technical guidance, and his valuable feedback at critical stages of the work.

I am especially grateful to Daniel Jones for his help in setting up the LiDAR, familiarising me with it and its software, and even volunteering as a subject in the preliminary datasets. I also owe him an apology for breaking the LiDAR's tripod mount a day before his Master's presentation - thank you, Daniel, for your kindness and patience.

To all the dog owners who kindly allowed me to record them and their pets, especially Nikki, thank you for your cooperation and enthusiasm. And to Atlas, the most patient participant of all, thank you for being a very good boy.

To my family and friends, whose support, humour, and company carried me through long nights and challenging days - thank you for making this chapter of my life truly unforgettable.

Finally, to my parents, thank you for your unwavering love, belief, and support throughout my studies and life. None of this would have been possible without you.

Ad maiorem Dei gloriam.

Abstract

Monitoring nocturnal wildlife is difficult with conventional camera traps, which depend on illumination and often disturb animals. This thesis investigates whether animals can be classified from their point cloud signatures captured by LiDAR, without RGB fusion. Using a Livox Avia solid-state LiDAR, a proxy dataset was collected at a public dog park to reproduce the dynamic and erratic conditions, characteristic to that of the wild.

A modular pipeline is proposed that (i) jointly clusters points into regions of importance (ROIs) and tracks them over adjacent frames to maintain identities through occlusions, and (ii) classifies these cropped regions using a fine-grained spatio-temporal model: a DGCNN per-frame backbone with a lightweight temporal CNN operating on the per-frame logits. Evaluation is performed with recording-level cross-validation across independent sequences, and assesses the performance of each stage in the pipeline in terms of accuracy, robustness, and processing complexity and latency.

On perfectly cropped ground-truth ROIs, the classifier reaches **96-97%** accuracy, establishing an upper bound that demonstrates the discriminative power of the point cloud signatures alone for the three classes (`dog`, `human` and `atlas`, whom is one specific dog). With crops produced by the joint cluster-track stage, end-to-end accuracy is **82-88%**. Median per-frame latency is **2.45 s** ($\sim$0.41 FPS), and $\sim$3.0-3.6 s time-to-decision for short sequences. Errors mainly arise from the cluster-track stage, when there is excessive occlusion due to crowding and point sparsity, rather than from limitations of the classifier.

These results show that an affordable LiDAR can support reliable animal recognition in low-light, cluttered environments. The analysis identifies ways to close the gap to the upper bound, notably integrating intensity end-to-end, improving the setup of the LiDAR, and optimising the pipeline's algorithms. The outcome of this thesis is a success and motivates scaling to a larger dataset with nocturnal wild animals, ultimately toward a fully automated LiDAR trap capable of detecting and classifying wildlife in any environment.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

Monitoring wildlife in nocturnal environments remains one of the most challenging tasks in ecology. Conventional camera-based methods are not suitable for these conditions, as they often require some form of illumination that can disrupt animal behaviour and has poor visibility in dense vegetation. Using LiDAR as a sensing device provides an alternative: it is light-independent, non-intrusive, and capable of capturing structural information over extended ranges. LiDAR has increasingly been utilized in autonomous vehicles and robotics, however, its potential for species recognition is unexplored. The motivation behind this thesis is therefore to investigate whether animals can be identified purely from their 3D structure without relying on colour or texture. Due to constraints on accessing wild species such as cheetahs, caracals, and bat-eared foxes, a dataset made up of dogs with their owners was used as a substitute for testing the same complexity that is expected in natural wildlife settings.

1.2 Problem Statement

This research addresses the challenge of identifying and classifying moving objects from point cloud signatures captured in dynamic outdoor environments. The goal is to output a labelled set of tracked objects for each frame. The system must handle occlusions, sparse sampling at long ranges and edges of the field-of-view (FOV), and moments where objects merge or fragment. Evaluation focuses on three key metrics: (i) clustering and tracking consistency relative to ground-truth cuboids; (ii) per-frame and sequence-level classification accuracy; and (iii) overall decision latency.

1.3 Gap and Contributions

Previous research in LiDAR-based sensing has primarily been in autonomous driving and structured environments, where objects are well separated and motion is predictable. Existing animal classification methods are highly dependent on RGB or RGB depth fusion, which limits robustness in nocturnal or cluttered conditions. Few studies use LiDAR-only end-to-end pipelines that perform under crowding, irregular motion, and limited data. This thesis bridges that gap through five main contributions. First, it introduces a joint clustering–tracking framework that couples motion and appearance features to maintain consistent identities through occlusions and merges. Second, it develops a fine-grained temporal classifier using a DGCNN backbone with a lightweight temporal CNN capable of operating on short sequences. Third, it integrates both stages into a unified end-to-end pipeline that achieves decision speeds approaching real-time feasibility.

Fourth, it provides an evaluation of the system's performance on a realistic dataset that incorporates all of the main challenges expected in real wildlife environments. Finally, it identifies and describes further work aimed at improving the system to a greater degree.

1.4 Scope, Assumptions, and Limitations

This study evaluates a LiDAR point cloud-only pipeline without sensor fusion (no RGB). Data were collected in open settings where dogs and owners act as a proxy for a wildlife dataset. The system operates under the assumption that the background remains static. Key limitations is the Livox Avia's capabilities (the LiDAR used), the small dataset with limited subject diversity and class imbalance, and a short collection/iteration window (under one week) that constrained hyperparameter tuning and ablations. Latency and feasibility were analysed using recorded sequences rather than in a live deployment. Although these constraints did shape the experimental scope, the system was evaluated under realistic, dynamic, and ecologically relevant conditions.

1.5 Thesis Outline

Chapter 2 reviews night-time sensing for wildlife, LiDAR's current applications and how it is used in ecology, point-cloud learning methods, and points out the gaps that motivate this thesis's approach. Chapter 3 details the hardware and software used in the thesis that influences data capture and experimentation. Chapter 4 establishes the theoretical foundations (LiDAR data fields, learning on unordered point sets, evaluation metrics) and then develops the methodology, comparing alternative pipelines before motivating the adopted cluster–track–classify pipeline. Chapter 5 presents controlled experiments on preliminary datasets used to validate core components and develop the final pipeline. Chapter 6 introduces the main dog-park dataset, describes the joint clustering–tracking stage and the fine-grained classifier, and reports results. Chapter 7 interprets these results, analyzing latency, failure modes, and practical implications for field deployment, and outlines improvements. Finally, Chapter 8 draws conclusions and gives final recommendations for future work.

Chapter 2

Literature Review

2.1 Introduction

Nocturnal wildlife monitoring is fundamentally constrained by low light and cluttered habitats. This chapter reviews sensing and learning approaches that addresses these constraints, with focus on LiDAR as a light-independent method to be able to classify species from their point cloud signatures.

This literature review synthesises work across ecology, robotics, and autonomous driving to (i) give an overview of the predominant night-time sensing devices and their limitations (section 2.2); (ii) trace how LiDAR has moved from habitat mapping towards direct animal recognition (section 2.3); (iii) outline point-cloud machine learning challenges (section 2.4) and (iv) examine point-cloud learning approaches - point, voxel, projection, and graph families - together with spatio-temporal models (section 2.5). Sources prioritise peer-reviewed studies (2014–2025) and methods transferable to sparse, irregular ground-level scans.

2.2 Existing Sensing Devices for Nocturnal Wildlife

Monitoring and classifying nocturnal animals is difficult due to the inherent lack of light, cluttered habitats, and difficulty in capturing motion through vegetation. Ecologists are currently utilizing several sensing devices - infrared (IR) cameras, thermal imaging, acoustic sensors, radar, and increasingly, LiDAR - to overcome these constraints.

Infrared camera traps are the most established method, coupling PIR (Passive Infrared) motion sensors with no-glow flashes to capture grayscale images that does not disturb the animal. Deep CNNs trained on these can exceed 90 % accuracy [5, 6]. Recently models such as YOLOv8-Night [7] have learnt to focus on the most informative parts of an image using channel attention mechanisms, which improves robustness to occlusion and low contrast. However, IR systems are inherently range-limited: the near-infrared flash typically illuminates only a few metres ahead, causing the performance of the models to degrade rapidly with distance or in dense vegetation and poor weather.

Thermal cameras detect long-wave radiation from warm-bodied animals without the need for a flash. This also allows detection in total darkness but produces texture-less, 'ghost-like' silhouettes. High ambient temperatures or heated terrain can cause false positives; fusion with visible or RGB images partially alleviates this [8]. Although thermal systems with longer detection ranges exist, they are prohibitively expensive for large-scale or long-term field deployment, restricting their use in ecological research settings.

Acoustic sensors target species such as bats, birds, and frogs. Models trained on audio spectrograms from

these animals have achieved high accuracy [8], but these systems are highly species-specific and provide no morphological information.

Other approaches, such as white-flash cameras, radar, or multi-sensor fusion, offer their specific advantages. White-flash traps capture colour but disturb animals; radar penetrates foliage but lacks resolution for species-level classification.

(a) White-flash camera trap (bat-eared fox).
Photo: L. Bruce Kekule [9].

(b) IR flash camera trap (cheetah).
Photo: Mark Chynoweth (QCNR team, Djibouti camera-trap survey) [10].

(c) Thermal camera (hedgehog).
Source: NatureSpy [11].

Figure 2.1: Examples of night-time sensing modalities used in camera-trap studies: (a) white-flash (full-colour), (b) infrared flash (grayscale), and (c) thermal imaging.

Table 2.1: Comparison of common sensing methods for night-time animal classification

Modality	**Advantages**	**Limitations**
Thermal Imaging	Works in complete darkness; strong contrast for mammals	Lacks shape detail; prone to thermal clutter
IR Camera Traps	Widely available; many mature deep learning models around	Limited range; no depth; affected by occlusion/orientation
Acoustic Sensors	Non-invasive; effective for vocalising species	No visual/morphological data; range limited by noise
Radar	Can detect through foliage	Low resolution; rarely used for species classification
LiDAR	Captures 3D form, posture, and motion; light- and temperature-independent	Rarely used for direct classification; lacks colour/texture

Despite progress, all existing methods depend mainly on surface appearance (colour, texture, or heat) rather than structural form. LiDAR can remain reliable at longer ranges, and its output does not change depending on the level of light found in the environment. It offers a pathway to species classification based on shape, posture, and movement.

2.3 LiDAR in Ecology and Wildlife Monitoring

Habitat Mapping. LiDAR has become crucial for ecological research for capturing fine-scale three-dimensional structure. It was initially created and developed to map forestry and terrain; it is now also

purposed to assess habitats and its biodiversity (schematised in Fig. 2.2a). LiDAR-derived metrics such as canopy height, gap fraction, and vertical heterogeneity correlate strongly with vertebrate diversity and habitat quality [12, 13]. These metrics quantify features - light availability, resource distribution, and vegetation layering - that shape ecological niches in various ecosystems.

Inferring Wildlife from Structure. Many studies use LiDAR not to detect animals directly but to predict where they are likely to occur. High-resolution airborne and terrestrial surveys have been used to map microhabitats relevant to nesting or foraging species [14], or explain the distribution of ungulates (hooved animals) [15], and to include canopy closure or woody debris metrics in estimating the probability a species occupies a site [16]. In these applications, LiDAR acts as a proxy linking habitat structure to species presence (Fig. 2.2b).

Toward Direct Animal Classification. A smaller but growing body of work explores LiDAR as a sensing device to detect animals themselves. Yang et al. [17] demonstrated a ground-based LiDAR 'trap' capable of reconstructing 3D posture and motion, while Haucke and Steinhage [18] showed that adding depth information to RGB camera traps improves classification accuracy. These studies highlight the value of structural information (shape, size, posture and gait) in identifying species where colour or texture are unreliable (Fig. 2.2c).

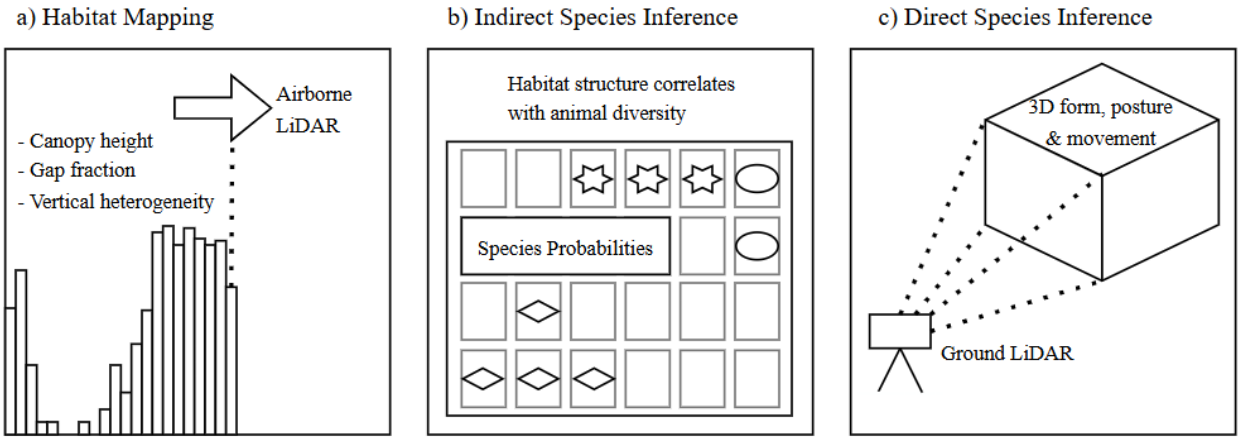


Figure 2.2: Evolution of LiDAR applications in ecology: (a) habitat and vegetation mapping, (b) species inference through structural proxies, and (c) direct 3D sensing of animals.

Despite its success in habitat modelling, LiDAR's potential for direct species-level classification, even under nocturnal conditions, remains largely unexplored:

- No benchmark datasets exist for ground-based LiDAR wildlife classification.
- Frame-by-frame animal reconstructions are rare, limiting temporal analysis of gait and motion.
- Methods optimised for urban or aerial domains have yet to be adapted for sparse, irregular ecological settings.

This thesis asks: *Can animals be identified solely by their 3D shape, posture, and movement using LiDAR?*

Answering this will further LiDAR's use from a habitat-mapping device into direct species-recognition with relevance to ecology, autonomous driving and robotics, agriculture, and human–wildlife conflict management.

2.4 Machine Learning Applications in Point-Cloud Data

Point clouds provide 3D spatial information for perception across autonomous driving, robotics and gesture recognition but their irregular, unordered nature makes learning fundamentally different from 2D images.

2.4.1 Applications Across Domains

In autonomous vehicles, LiDAR is the predominant sensor used for object detection, localisation, and semantic segmentation [19]. In robotics, LiDAR is also used in navigation and mapping in unstructured, GPS-poor environments [20]. These uses have driven specialised architectures that operate efficiently on point clouds.

2.4.2 Learning Models

Point-cloud architectures span over both supervised and unsupervised learning. Supervised models (e.g., PointNet, PointCNN) are trained on labelled 3D datasets for classification or segmentation [21]. Unsupervised approaches (e.g., autoencoders and generative/adversarial training) learn geometry-aware point-cloud features (embeddings that capture global shape and local surface structure) from unlabeled data. In practice, these features are invariant to point order and rotations, and remain stable under resampling/sparsity, so similar 3D shapes map to similar representations. They remain less mature and less widely adopted than the supervised approaches.

2.4.3 Key Limitations

- **Sparsity and irregularity.** Unlike pixels on a grid, points are unordered and have non-uniform density. This makes local aggregation and consistent feature extraction challenging.
- **Rotation and permutation invariance.** Models must be invariant to input ordering and orientation [21]; this typically requires symmetric aggregation and either spatial transformers or targeted augmentation.
- **Scalability and data gaps.** Processing large scenes is computationally demanding [19], and most public datasets are of urban road settings, limiting transfer to ecological settings.

2.4.4 Relevance to Ecology

Ecological scenes exhibit occlusion and highly non-uniform point cloud density. Moreover, while road-scene methods aim to classify all objects within a frame, this thesis aims to classify clusters corresponding to a specific animal.

2.5 Deep Learning for LiDAR-Based Classification

LiDAR-based classification approaches can be grouped into four broad architectural families: *point-based*, *voxel-based*, *multi-view and projection-based*, and *graph-based* methods. Each category represents a different strategy for handling the irregular structure and non-uniform density of point cloud data, with trade-offs in spatial resolution and computational efficiency.

The following subsections review each family in depth, comparing their strengths, limitations, and relevance to classifying animals.

Beyond these methods, **temporal and spatio-temporal models** extend the processing window to sequences of point cloud frames, capturing the motion of clusters that can be critical in distinguishing species with similar morphology but different movements.

2.5.1 Point-Based Architectures

Point-based neural networks operate directly on raw, unordered point sets without voxelization or projection, thus preserving the structure of the point cloud and avoiding discretisation artefacts. This, however, comes with specific challenges: with permutation invariance, dealing with non-uniform density, and keeping computation feasible on large-scale frames.

PointNet [1] addresses permutation invariance by applying the same pointwise multilayer perceptron (MLP)[1] to every point and aggregating with a symmetric function (max pooling) to form a global descriptor. Figure 2.3 (top, blue) illustrates the key stages: an input transform (T-Net) aligns the raw coordinates; a shared MLP lifts points to higher-dimensional features; an optional feature transform further normalises the feature space; and a global max pool yields a permutation-invariant 1024-D vector that feeds the classification head. For segmentation, PointNet concatenates this global descriptor back to each point's local feature (Fig. 2.3, bottom, beige), producing per-point scores. The architecture is powerful for capturing global shape but weak at modelling fine local structure, because neighbourhood relations are not explicitly encoded.

PointNet++ [22] fixes this limitation by introducing hierarchical set-abstraction layers (ball queries or k-NN) that learn features on progressively larger neighbourhoods. In practice, this adds the local geometric context that PointNet lacks.

Random Sampling: Efficiency vs Fidelity. Training on all points in a LiDAR scan is expensive, so point-based pipelines typically downsample the input before each forward pass, either at random or with farthest-point sampling (FPS), which iteratively picks the next point that is farthest from those already chosen to spread samples across the shape [22, 23]. While this reduces computation, naïve sampling can remove exactly the points that are needed to distinguish classes - an issue that worsens under sparsity or occlusion [23]. To limit this loss, importance sampling keeps points expected to be informative (e.g., high curvature or high reflectance intensity), or uses attention mechanisms that learn a weight for each point and preferentially retain high-weight ones [24]. Another strategy is hybrid dense–sparse sampling, where a coarse detector or simple heuristics define regions of interest (ROIs) that are sampled densely, while the

[1] **MLP**: a small stack of fully connected layers with nonlinearities that maps each point's $(x, y, z, ...)$ to a feature vector.

background is thinned more aggressively [25]. Finally, interpolation-based recovery processes a reduced set of anchor points and then propagates features to nearby unsampled points using inverse-distance or learned interpolation kernels (as in ICNN) so that an object's fine structure is reconstructed without processing the full cloud at every layer [26].

Advances in Feature Encoding. Relying only on coordinates (x, y, z) underuses LiDAR. Each return may also include intensity (reflectance strength, surface/material), a return number (from multi-echo systems, penetrations through vegetation), or even multispectral reflectances. Modern encoders include these channels together with the point coordinates to improve robustness to irregular sampling. In SE-PointNet++, a squeeze–excitation block first 'squeezes' features by global pooling to compute per-channel statistics and then 'excites' them with a small gating network that rescales channels - amplifying those most informative for the task and damping noisy ones [24]. Centroid-aware PointNet++ augments each local descriptor with features computed at the neighbourhood centroid, which encodes where that patch lies within the object/scene and helps disambiguate similar local shapes appearing in different contexts [25].

Beyond architecture, practical steps such as designing a model specific to a domain - where known environment/target information is introduced into the model (e.g., height-aware forestry models that ground-normalise points and encode vertical canopy–understory layers) [27] - and preprocessing (denoising, ground removal, intensity normalisation) [28], are as impactful for performance.

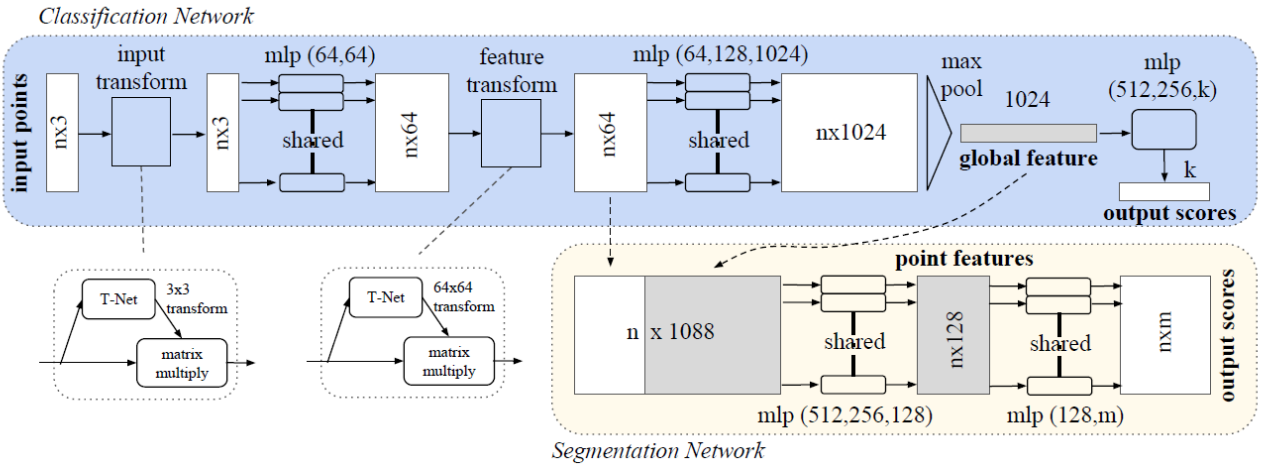


Figure 2.3: **PointNet overview** [1]. Top (blue): classification pipeline: input T-Net aligns coordinates; shared MLPs lift per-point features; a feature T-Net refines feature space; global max pooling yields a permutation-invariant descriptor for class prediction. Bottom (beige): segmentation pipeline: the global descriptor is concatenated to per-point features to output pointwise labels. The figure highlights why PointNet captures global shape well but needs hierarchical neighbourhoods (as in PointNet++) to model fine local structure.

2.5.2 Voxel-Based Approaches

Voxel-based methods convert irregular point clouds into a regular 3D grid (made up of voxels), allowing the use of 3D CNNs on the point cloud. This discretisation introduces a central design trade-off, which is the size of the voxel. Small voxels preserve fine detail but increase the computational complexity of the model;

large voxels are more computationally efficient but introduce more quantisation artefacts. In what follows, 'sparse' refers to operators that compute only at non-empty voxels.

Foundational architectures. **VoxelNet** [2] partitions space into voxels, learns a feature per voxel with a PointNet-style Voxel Feature Encoding (VFE) block, and then applies 3D convolutions for detection. Figure 2.4 summarises this pipeline: voxel partitioning and per-voxel grouping; optional within-voxel random sampling; stacked VFE layers with elementwise max-pooling to obtain a single descriptor per voxel; assembly into a sparse 4D tensor that feeds a convolutional backbone and a region proposal stage. **SECOND** [3] improves this by using sparse convolutions, using only non-empty voxels to remove wasted computation while retaining accuracy.

Multi-scale voxelisation. A single voxel size often does not suit an entire scene. Dense vegetation benefits from fine voxels, whereas open terrain could allow bigger voxels. Voxel-FPN [29] addresses this by learning features on multiple voxel grids (e.g., $S, 2S, 4S$) and fusing them with a feature pyramid network. Fine branches capture the local detail; and the coarse branches provide context. This reduces sensitivity to point-density variation common in field scans, at the cost of higher memory due to operating with multiple grids concurrently.

Task-specific optimisations. Some tasks require preserving very local structure without paying the full 3D cost everywhere.**VoxelKP** [30] (human keypoint estimation) introduces: (i) sparse selective kernels that adapt the 3D convolutional receptive field size to the local structure - small for fine detail, larger for context, and (ii) box-based attention that concentrates computation inside candidate regions of interest (ROIs) while down-weighting background. This combination preserves fine details while avoiding unnecessary processing. This is transferable to wildlife classification where subtle morphology features are distinguishing.

Comparative perspective: voxels vs. pillars. Voxels retain full 3D structure but can be computationally heavy. Pillar-based variants (e.g., PointPillars [31]) collapse the vertical axis to form 2D 'pillars', enabling fast BEV processing. While it is efficient in urban road datasets, the loss of vertical detail is problematic in distinguishing between animals as their vertical profile matters. Hybrid voxel - pillar backbones partly mitigate this, but pure pillars remain vulnerable to vertical information loss.

Limitations and gaps.

1. **Fine-detail loss in sparse targets:** Quantisation can obscure subtle morphology especially at long range.
2. **Rigid voxel sizing:** Fixed grids are inefficient under mixed density; adaptive voxel sizes remain uncommon in practice.
3. **Multi-scale overheads:** Designs such as Voxel-FPN improve robustness but substantially increase memory and runtime, which is challenging for embedded, real-time systems.

4. **Domain shift:** Models designed to be trained on urban road datasets (e.g., KITTI, Waymo) degrade in natural habitats with dense vegetation, uneven terrain, and occlusions.

Relevance to ecological monitoring. For species-level classification, voxel pipelines must preserve the morphology needed to distinguish animals while remaining feasible for near–real-time deployment. A strategy could be a two-stage hybrid: use a coarse voxel (or pillar) backbone for global context and to find regions of interest (ROIs), then apply fine-voxel or a point-based approach within each region to recover detail. This balances efficiency with the precision needed to separate morphologically similar species in complex natural environments.

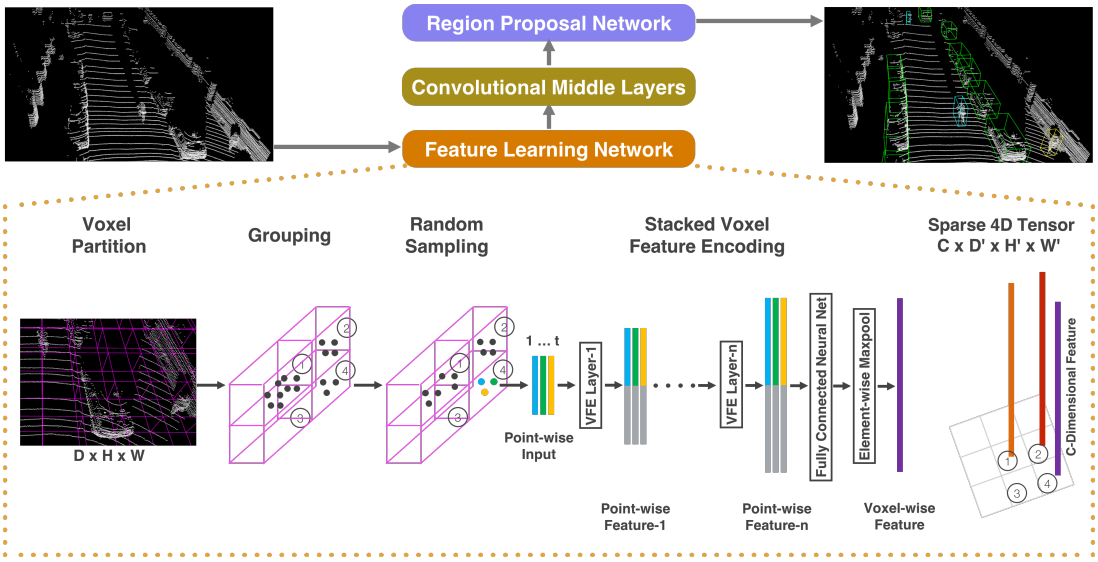


Figure 2.4: VoxelNet pipeline [2]. Bottom: (1) voxel partitioning of the point cloud; (2) per-voxel grouping; (3) optional random sampling within voxels; (4) stacked VFE layers (shared MLPs) with elementwise max-pooling to produce one descriptor per voxel; (5) assembly into a sparse 4D tensor $C \times D' \times H' \times W'$. Top: the sparse tensor is processed by a 3D convolutional backbone and a Region Proposal Network (RPN). SECOND [3] replaces dense 3D convolutions with sparse convolutions at this stage.

2.5.3 Multi-View and Projection-Based Approaches

Projection-based methods map 3D point clouds into 2D projections - most commonly bird's-eye view (BEV), front view, or range view (RV) - so that established 2D CNN's can be applied to them. This increases efficiency and the use of well-established models, but inevitably compresses the depth of point clouds.

Representative architectures. Early detectors (e.g., MV3D, PIXOR) operate on BEV feature maps to achieve real-time performance in driving scenes. The BEV projection flattens the point cloud space into a ground-plane grid, thereby reducing computation. YOLO4D [32] extends YOLO3D with temporal modelling (ConvLSTMs) over BEV inputs to improve consistency across frames; nevertheless, the BEV projection loses the distinction between objects that have similar footprints but differ in vertical profile or posture. To alleviate this information loss that comes with the BEV projection, Li [33] addresses this with Spatial–Semantic Feature Aggregation (SSFA): it takes shallow BEV maps for fine shape/position and deep BEV maps for category cues, aligns them to the same resolution (upsampling + 1Ö1 conv), fuses them

(concatenation + a light conv/attention block), and adds a supervised contrastive head that explicitly clusters features by class, bringing same-class features closer and separating different classes. Methods like these recover part of the lost detail but remain bounded by information lost by collapsing the z-plane.

Range view and cross-view. RV rasterises the LiDAR scan in the sensor's angular domain, producing an image where its pixels store range. RV tends to preserve near-field detail better than BEV, but becomes sparse at long range due to angular sampling [34]. Cross-view alignment can combine their complementary strengths. The Geometric Flow Module (GFM) [4] constructs correspondences between RV and BEV using the original 3D points, then performs attention-based fusion so that RV contributes vertical detail while BEV contributes global layout. Figure 2.5 illustrates the pipeline: spherical projection produces RV; top-down projection produces BEV; bidirectional mappings (R→B and B→R) share information across the two views before downstream inference.

Critique and comparative perspective. Comprehensive surveys [35, 23] characterise projection pipelines as computationally efficient and accurate in structured, high-density settings, but sensitive to (i) loss of vertical detail from BEV pooling; (ii) point sparsity, which degrades both BEV and RV for small or distant targets; and (iii) environment dependence, as models designed for urban road datasets often transfer poorly to cluttered ecological scenes with occlusion and non-rigid shapes.

Relevance to ecological monitoring. For species-level recognition, the animal's posture and vertical cues are critical, making pure BEV or RV insufficient. However, a design could be to use BEV/RV projection for efficient coarse detection, then refine each candidate with a high-resolution point-based or fine-voxel model, to recover its 3D structure required for reliable classification.

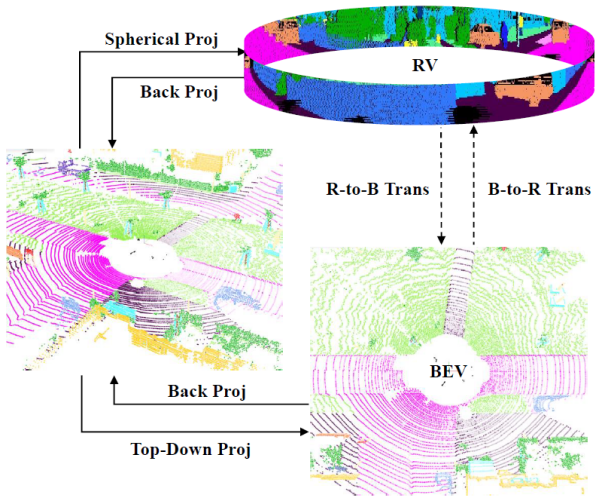


Figure 2.5: Bidirectional RV–BEV mapping [4]. Spherical projection yields range-view (RV) images; top-down projection yields bird's-eye view (BEV). Geometry-aware correspondences enable feature transfer in both directions (R→B and B→R), allowing RV's vertical/near-field detail to complement BEV's global spatial context, as in cross-view fusion modules such as GFM [4].

2.5.4 Graph-Based and Local Neighbourhood Methods

Graph and local-neighbourhood approaches model a point cloud as a set of nodes (points) connected by edges that encode spatial proximity (connect points that are close in 3D space) or feature similarity (connect points that have similar attributes). This representation preserves the native, irregular geometry without voxelising to a grid or projecting to 2D, and it lets the network reason explicitly about local surface structure that quantisation would blur.

DGCNN (dynamic graphs). **DGCNN** [36] first builds a k-nearest-neighbour graph from the point cloud, then learns an edge feature for each point–neighbour pair using a small MLP on the neighbour offset $(x_j - x_i)$ together with the central point x_i. These edge features are pooled (e.g., max) to give one updated feature per point, so the method captures local shape while remaining permutation-invariant (the pooling ignores input order). Crucially, after every layer the graph is rebuilt in the current feature space, so 'neighbours' become points that look similar in the learned representation, not just points that are physically close. This lets the network link semantically related parts (e.g., both ear tips) even if they are far apart in 3D, growing the effective receptive field and injecting global context. The trade-off is computational cost. Each layer needs a new k-NN search and edge computation, which can be heavy on dense scans.

Continuous kernels on points (KPConv). **KPConv** [37] applies convolution directly on the point cloud. Around a chosen support point, it looks at neighbours inside a small radius and aggregates them with a handful of learnable kernel points. In the rigid version the kernel points stay fixed (only the weights are learned); in the deformable KPConv version they can shift slightly during training to better fit local geometry. Because no voxel grid or projection is used, KPConv tends to preserve sharp boundaries and thin structures, which is helpful for subtle anatomy - though it does require fast neighbour searches and a sensible radius (too small loses context; too large adds cost and blur).

Strengths, limits, and open gaps. Graph-based architectures are among the most expressive models for non-uniform point sets and are good at retaining surface detail crucial for species-level distinctions. However, three practical issues remain:

1. **Scalability.** Dynamic graphs incur repeated neighbour searches (typically $O(N \log N)$), which restricts large point clouds.
2. **Density sensitivity.** Fixed k or radius neighbourhoods can over-smooth dense vegetation or under-sample sparse animal points; adaptive neighbourhood weights and density-aware weights help but are not commonly used.
3. **Under-use of auxiliary channels.** Many GNNs (Graph Neural Networks) still rely primarily on (x, y, z). Using intensity, return number, or multispectral LiDAR is underexplored despite likely benefits for fine-grained classification.

Graph-based methods preserve geometry well but are costly with neighbour-search computations and implementation complexity [38, 39].

Relevance to ecological monitoring. For nocturnal wildlife, where distinguishable information is subtle and sampling is irregular, graph methods could work well for ROI-level classification. First, detect candidate animal clusters (using voxels or projections), then run a lightweight dynamic-graph/KPConv classifier only inside those regions (ROI) to recover the fine details.

2.5.5 Spatio-Temporal Classification from Point Clouds

Most deep learning models for point clouds operate on single frames, yet for species classification based on movement or posture, temporality can provide more information then just the point cloud can. Animals that appear similar in a static frame can exhibit distinct gait, joint motion, or movement.

Importance of Temporal Continuity

Analysing sequences requires modelling how geometry evolves between frames (temporal continuity). Simple temporal pooling (averaging over time) or frame concatenation can aggregate history, but they often miss finer motion (like the movement of individual limbs). Sequence-aware models explicitly track change over time using recurrent memory (e.g., LSTMs) or temporal convolutions that slide along the time axis.

Fan et al. [40] address this with PSTNet, which introduces a point spatio-temporal convolution (PSTConv) that operates directly on raw points. For each frame, PSTConv aggregates local spatial features from neighbours, then concatenates these per-frame features in temporal order and fuses them, capturing motion at the same spatial resolution as the points themselves. Compared with pipelines that first learn spatial features (PointNet++) and then add a separate RNN, PSTNet links space and time concurrently and improves performance in dynamic scenes.

Spatio-Temporal Convolutional Models

Spatio-temporal models jointly encode 3D coordinates and their changes across frames, rather than treating time as a separate, later stage.

Point-based spatio-temporal convolutions. Wang [41] proposes PointMotionNet, a point-based pyramid that learns motion directly from raw points. Its Point-STC operator works in two simple steps: (1) per frame, it gathers each point's neighbours within a fixed radius and passes them through a small pointwise MLP (a few fully connected layers) to get local spatial features; (2) over time, it stacks those per-frame features in temporal order and mixes them to produce a descriptor that captures both shape and how it changes. Because it never voxelises or projects the data, there are no quantization artefacts. However, each frame must be well aligned and sufficiently sampled, which cannot be assumed in ecological environments with occlusion and sparse returns.

Recurrent Temporal Aggregation. El Sallab et al. [42] extended YOLO3D to YOLO4D by injecting a ConvLSTM over the BEV features. Two ways were compared: (i) frame stacking (concatenate BEV frames at input) and (ii) a recurrent ConvLSTM. The recurrent model improved temporal stability and accuracy under noise, but with more computational cost.

Memory-augmented temporal models. **StreamMOS** [43] carries information across time with two memories: a short-term feature bank that passes useful features from the last frame to the current one (helping through brief occlusions), and a long-term bank of past predictions used to refine the current output using a simple voting scheme. For single-object tracking, **Feng et al.** [44] use a bi-directional memory that feeds information from both earlier and later frames into the current step, keeping the tracker focused on the true target and reducing false switches to similar-looking objects in the scene. These designs improve consistency and robustness but increase RAM and latency.

Spatio-Temporal Feature Fusion. Yang et al. [45] insert lightweight temporal-convolution blocks at several points in the per-frame backbone, so features are aggregated across neighbouring frames (e.g., $t-1, t, t+1$) rather than decoded frame-by-frame. Early (shallow) blocks fuse motion with fine geometric features (edges, thin parts), whereas later (deeper) blocks fuse motion with more contextual features (object identity/parts). In effect, the temporal convolution learns patterns of consistent motion. Therefore, consistently moving structures are strengthened and noisy movements are ignored. This stabilises predictions across frames, as it preserves continuously moving body parts and reduces the impact of occlusions and noise.

Comparative Perspective. Point-based temporal models (e.g., PointMotionNet) preserve fine geometric structure by directly processing 3D points across time, but they depend on well-chosen neighbourhoods and consistent sampling within and between frames. Projection-based temporal models (e.g., YOLO4D) are computationally less costly; however, the projection inherently loses vertical information. Memory-augmented architectures introduce long-term temporal stability by storing past features, but they increase both computational load and storage requirements.

This chapter viewed current methods used to classify nocturnal animals as well as their limitations. It provided an overview of applications LiDAR is currently used in and its evolution hereto, and reviewed point-cloud learning approaches. It outlined the main architectural families: point-, voxel-, projection-, and graph-based models, as well as spatio-temporal models. Their trade-offs in accuracy, robustness, and computational cost, were weighed in regards to conditions relevant to this thesis, thereby identifying gaps for (animal) classification using point cloud signatures.

Chapter 3 presents the hardware and software used throughout the course of the thesis.

Chapter 3

Hardware and Software

3.1 Overview

This chapter outlines the hardware and software that was used in all stages of the research, including data acquisition, preprocessing, and model training that follows.

3.2 Hardware

3.2.1 Sensors and Mounting

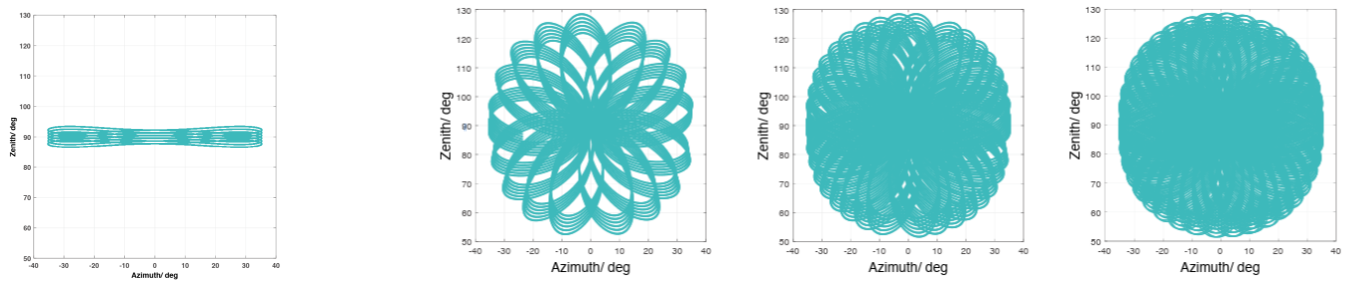
Data were captured using a Livox Avia solid-state LiDAR sensor, which forms the foundational device for this thesis. Its specifications and characteristics guided various design decisions through this thesis. The use of this LiDAR was not a design choice but rather what was available at the University of Cape Town at the time of the undertaking of this project. While a number of high-end LiDAR systems exist that offer more superior range and resolution, the Livox Avia strikes a favourable balance between cost, performance, and portability.

Higher-end systems such as the Velodyne Alpha Prime or Ouster OS1 offer improved precision at range and denser sampling but at substantially greater cost, often exceeding US$10,000. In contrast, the Livox Avia retails at approximately US$1,500–1,800 at the time of this thesis, providing an accessible, feasible and scalable alternative to traditional, more expensive LiDAR systems for ecological monitoring.

Table 3.1: Minimal Livox Avia specifications used in this thesis.

Laser wavelength / class	905 nm / Class 1
FOV (non-repetitive)	$70.4^\circ \times 77.2^\circ$
Range @ night (20% refl.)	260 m
Distance random error (1σ @ 20 m)	< 2 cm
Angular random error (1σ)	$< 0.05^\circ$
Beam divergence (V×H)	$0.28^\circ \times 0.03^\circ$
Point rate (dual / triple)	480k / 720k pts/s
Close-proximity blind zone	$\lesssim$1 m

The Livox Avia has two distinctive scanning patterns. (i) A non-repetitive 'rosette' pattern that gradually fills its $70.4^\circ \times 77.2^\circ$ field of view, and (ii) a repetitive pattern that follows fixed ray paths each sweep. In non-repetitive mode, sampling is denser near the centre and progressively covers new regions of space over time. Figure 3.1 illustrates the difference between the repetitive scan and the non-repetitive pattern.



(a) Repetitive pattern.

(b) Non-repetitive pattern. Coverage increases with time (left to right).

Figure 3.1: Angular sampling patterns in azimuth–zenith space. The repetitive pattern revisits the same rays each sweep. The non-repetitive pattern samples different rays each frame, progressively filling the FOV.

The LiDAR was mounted on a tripod at a height of 0.8-1 m above ground level. A GoPro camera was co-aligned with the LiDAR to provide a photo of the sensor's field of view during recording, though images from it was used only for reference and not for model training or ground truth. For practical reasons, all recordings in the experiments were conducted during the day to make recording ground truth easier. Nevertheless, the LiDAR's rays at 905 nm allows fully equivalent performance in complete darkness. Previous studies have reported improved performance at night due to less ambient light interference.

3.2.2 Compute and Power Setup

All LiDAR data acquisition, preprocessing, and model training was conducted on a single **Acer Nitro AN515-57** laptop. The system included an Intel Core i7-11800H CPU, 16 GB RAM, and an NVIDIA RTX 3050 GPU with 4 GB VRAM.

A 512 GB Solid-State Drive provided sufficient storage for all datasets and model checkpoints.

During data acquisition, both the laptop and the Livox Avia were powered by aG izzu 296Wh Portable Power Station with a AC-DC converter.

3.3 Software

3.3.1 Core Environment

Table 3.2: Software environment for LiDAR data capture and model training.

Item	Version / Notes
Operating System	Windows 11 Home (24H2)
Python	3.10.18 (Conda env: `torch-gpu`)
PyTorch	2.5.1 (CUDA 12.1, cuDNN 9)
Key Libraries	NumPy, SciPy, Pandas, scikit-learn, Open3D, Matplotlib
Capture Stack	Livox Viewer 0.11.0 + Livox SDK 2.6
Annotation	CVAT 1.8 (Docker 28.4.0)
IDE	Jupyter Lab

LiDAR data were captured and visualised using **Livox Viewer 0.11.0**, which interfaced directly with the Livox Avia for streaming and recording of `.lvx` files. All preprocessing, clustering, tracking, and model training were implemented in Python (PyTorch 2.5.1, CUDA 12.1) and run locally on a CUDA-enabled laptop GPU. Google Colab and their hosted GPUs were initially used, but compute limits and time-capped sessions often terminated runs mid-training. Moving the workflow to a local Jupyter Notebook fixed this and sped up training by 6–7× compared to just using the CPU. Annotations were created in **CVAT 1.8** running in a Docker 28.4.0 container.

3.3.2 Choice of Programming Language

Although MATLAB provides established toolboxes for point-cloud processing and visualisation, Python was used as the primary language for this project. The decision was driven by three factors:

1. **Deep learning integration:** Python natively supports deep learning frameworks such as PyTorch, allowing flexible development and training of deep learning models specific to the project's needs.
2. **GPU acceleration and flexibility:** Python provides access to CUDA so that the GPU can be used for computation, along with automatic mixed-precision training (`torch.cuda.amp`) to speed up processing and reduce memory usage. Its configurable data-loading pipelines (parallel workers, prefetching, etc.) gives greater control and allows the GPU to remain fully utilised during training, improving overall efficiency and throughput.
3. **Open-source ecosystem and flexibility:** Libraries such as Open3D, NumPy, Pandas, and Matplotlib provided a fully open, end-to-end workflow for preprocessing and training.

MATLAB was used only in the early stages for visualising point clouds.

The hardware and software configuration described here provided the foundation for all subsequent experiments. The next chapter builds on this system to describe the methodological framework and theoretical principles guiding data processing and model training.

Chapter 4

Theoretical Framework and Methodology

4.1 Chapter Purpose

This chapter presents the theoretical framework the thesis is built upon as well as the methodology taken. Section 4.2 outlines the outputs from the LiDAR (fields and their purpose), and introduces core machine-learning principles relevant to this thesis. Section 4.3 then details the system design, explains alternative pipelines considered, and justifies the adopted isolate/cluster to track to classify approach.

4.2 Theoretical Framework

4.2.1 LiDAR Data Model: Fields and Semantics

Each LiDAR point is represented as an array with the main subfields:

$$\texttt{timestamp}\ ,\ x\ ,\ y\ ,\ z\ ,\ \texttt{reflectivity}$$

The fields are:

- `timestamp` - acquisition time (ns). Used to order points in the sequence they are returned back to the LiDAR.
- x, y, z - Cartesian coordinates in the sensor frame (metres).
- `reflectivity` (intensity) - return strength (unitless). Provides information for property of the surface and angle of incidence.

Further subfields may exist but were not utilised in this work and are excluded here.

4.2.2 How Models Learn From Point Clouds

LiDAR sensors capture scenes as a set of points rather than a structured image grid. Each frame is an unordered collection of 3D points, where each point represents a laser return in space. Unlike an image, where each pixel's position is fixed on a 2D array, a point cloud has no inherent order or uniform density. This means that a learning algorithm cannot convolve a kernel across the data as in conventional image convolutions. Instead, the model must learn spatial relationships directly from the coordinates themselves.

Because this work sets out to classify species from labelled examples, a supervised learning model is used: models are trained on input–label pairs to minimise the loss between predictions and ground truth.

Core requirements. For a neural network to learn effectively from point clouds, it must satisfy two properties:

- **Permutation invariance.** A point cloud has no fixed ordering: the same object can be represented by the same points listed in any sequence. The network must therefore produce identical outputs regardless of input order. This is achieved by applying the same transformation (usually a shared MLP (Appendix A) or equivalent pointwise operator) to each point individually and then using a symmetric aggregation function (such as maximum or average) that ignores order.
- **Local geometry awareness.** Because points are irregularly spaced, spatial structure must be inferred from the relative positions of neighbouring points, not from a grid layout. The model therefore needs a mechanism to group nearby points and learn relationships such as shape, curvature, and surface continuity within each local region.

Architectural approaches. Different families implement these principles in different ways:

- Some networks operate directly on points themselves. They treat each point as an independent input, apply shared neural layers to extract per-point features, and then use order-independent pooling to summarise the entire object. These are known as **point-based networks** (e.g., PointNet, PointNet++).
- Others explicitly connect points to their neighbours, forming a graph that contains the spatial relationships between them. Each point becomes a node linked to its k nearest neighbours, and the model learns from the edge features that describe the differences between connected points. These are called **graph-based networks** (e.g., DGCNN, KPConv). By learning over edges, the model learns local geometric structure and surface.
- Some approaches regularise the data by discretising space into a uniform grid of small 3D cells (voxels) or by projecting the cloud into a 2D image representation such as a range or bird's-eye-view map. These are **voxel and projection-based models**. They allow the use of conventional convolutional networks but introduce quantisation artefacts.
- Finally, when processing sequences of frames, models can capture how the points move over time. These **spatio-temporal models** integrate motion through temporal convolutions, recurrent units (RNNs/GRUs), or temporal graph connections. They are particularly suited to recognising behaviours such as walking or running, where movement patterns define the class as much as geometry does.

Despite these architectural differences, all families optimise the same supervised task: given an unordered set of points (or a short sequence of such sets), predict its class.

Learning objective. Let $f : \mathcal{X} \to \mathbb{R}^K$ map the chosen input representation to K-way scores, with class probabilities $p = \mathrm{softmax}(f(\cdot))$, where K is the number of classes. Here $\mathcal{X}$ is whatever tensor the pipeline takes in (points, voxels, or 2D projections). Inputs are generically written as a tensor of shape $\bullet \times C$, where the leading dimension ($\bullet$) is the elements and C is the number of channels per element. An element is the unit of the representation (a point for point sets; a voxel for voxel grids; a pixel for 2D projections). Thus, C

is the feature depth attached to each element (i.e., the per-element measurements or statistics the pipeline provides), while the first dimension is what (how space is indexed) and how many such elements are present.

Channels are selected that:

- **Carry discriminative signal.** Channels should encode features that differ between classes but are consistent within a class. This means representing (a) shape: the object's geometric structure; (b) height profile: how elevation is distributed across the object; (c) reflectance: how strongly surfaces return the LiDAR rays; and (d) motion: how the representation evolves across frames.
- **Remain stable under nuisance factors.** Their values should change minimally when conditions unrelated to class vary (e.g. distance, uneven sampling, or viewpoint). Invariance or controlled equivariance to these factors prevents the model from learning shortcuts.
- **Can be normalised consistently.** Channels must admit a simple, repeatable scaling/centering so that their ranges and meanings are comparable across sessions, sites, and sensors, supporting reliable training, calibration, and transfer.

Avoid channels that are linked to the recording or location; they bias the model towards context rather than class and weaken generalisation.

Evaluation metrics. Given f and the chosen input channels/representation, performance with classification, localisation, and calibration metrics, reporting either *macro* (per-class average) or *micro* (pooled) scores is assessed in the report as noted.

- **Accuracy** - fraction of all samples predicted correctly (overall classification quality).
- **Precision** - among predicted positives, the proportion that are truly positive (reliability of detections).
- **Recall** - among true positives, the proportion recovered by the model (completeness).
- **F1-score** - harmonic mean of precision and recall (balances false positives/negatives).
- **Intersection over Union (IoU)** - overlap between a prediction and ground truth divided by their union; for detection, a prediction counts as correct if IoU exceeds a threshold (e.g., 0.5).
- **Expected Calibration Error (ECE)** - confidence–accuracy gap: bin predicted confidences (e.g., 10 bins) and take the weighted average of |accuracy − confidence| per bin (lower is better).

4.3 Methodology

4.3.1 Design Goals and Constraints

The objective is to develop a LiDAR-only classification pipeline that can operate reliably under nocturnal, low-light field conditions. Several practical and technical constraints shape the way the pipeline is constructed:

- **Portability.** All stages should ideally train and infer on a single laptop GPU (RTX 3050, 4 GB VRAM) without requiring large clusters or cloud infrastructure.

- **Robustness to sparse and irregular sampling.** The Livox Avia's non-repetitive scanning pattern produces uneven point densities and partial occlusions that must not disrupt classification.
- **Multi-target capability.** The final pipeline must scale to handle multiple concurrent target objects within the same frame.
- **Domain generalisation.** Models must learn the intrinsic 3D and motion characteristics of the animals themselves - rather than correlating with environmental background or capture location - to remain transferable across sites.

These requirements guided the evaluation of possible processing pipelines.

4.3.2 Pipelines Considered

Multiple designs were explored before settling on the selected cluster–track–classify pipeline. Each alternative offered distinct trade-offs in accuracy, scalability, and data requirements. A compact summary of the five alternatives is shown in Fig. 4.1.

1. End-to-End Scene Segmentation and Detection. This approach trains a single deep network to detect and classify all objects directly within a raw point cloud space, like 3D versions of YOLO. It is popular in structured domains such as autonomous driving (where large labelled datasets exist), it is not practical for this work:

- Requires bounding-box annotations for everything within a frame
- Performance degrades on sparse, non-uniform LiDAR scans typical of the Livox sensor.
- Single monolithic model gives limited interpretability and minimal control over individual stages.

2. Template or feature matching. Classical 3D pipelines compute fixed geometric descriptors on local patches (e.g., spin images, shape histograms, eigenvalue-based curvature from the neighbourhood covariance) and then pick the nearest template in a library. This is simple and interpretable, but would be difficult to implement in this project because:

- Descriptors depend on the full local neighbourhood; when the animal is partially occluded, the point cloud signature will change and no longer match the template.
- Running vs. crouching bends limbs and spine; templates are not learned to be invariant to these deformations, so the same species can look unlike its own templates.
- At longer distances or oblique angles, point density drops and noise increases; histogram/curvature values shift because sampling density changes and not the object.

In short, fixed descriptors match well under controlled, rigid, fully visible conditions, but they do not adapt to the occlusion, posture variation, and irregular sampling.

3. Projection-Only Approaches (Range View / Bird's-Eye View). Projection-based models project 3D points onto a 2D plane-commonly a range image or top-down BEV map-and apply efficient 2D convolutional networks. These methods are computationally efficient, but suffer from several limitations:

- Vertical features are compressed, reducing discriminability between similar species.
- Sparse returns lead to discontinuities and merged silhouettes when multiple animals overlap.
- Reprojection artefacts make tracking unstable across consecutive frames when point density varies.

Despite these drawbacks, BEV projections were later repurposed in this thesis as lightweight detection heads for coarse localisation (Chapter 5).

4. Clustering followed by per-frame classification. A simple alternative is a stagewise pipeline that operates on each frame independently: (i) coarse scene filtering to remove most background points; (ii) clustering (e.g., BEV/voxel masks) to isolate candidate objects; (iii) ROI cropping to focus on each candidate; and (iv) local classification of each ROI with a point/graph-based network. This has two major drawbacks:

- Without associating clusters across frames, small shape/density fluctuations could make the same animal flip labels frame-to-frame, producing unstable predictions.
- Gait and movement, which can be distinguishing for species, are ignored when frames are classified in isolation, reducing robustness.

5. Multi-Stage Modular Pipeline (Adopted). The adopted design follows a modular cluster-track-classify pipeline:

1. **Cluster (isolate):** Clustering identifies 3D regions that resemble individual animals or objects of interest (ROIs).
2. **Track:** A tracker maintains identity by linking the same cluster across consecutive frames, converting unordered frames into sequences.
3. **Classify:** Each tracked ROI is then classified by static or spatio-temporal models.

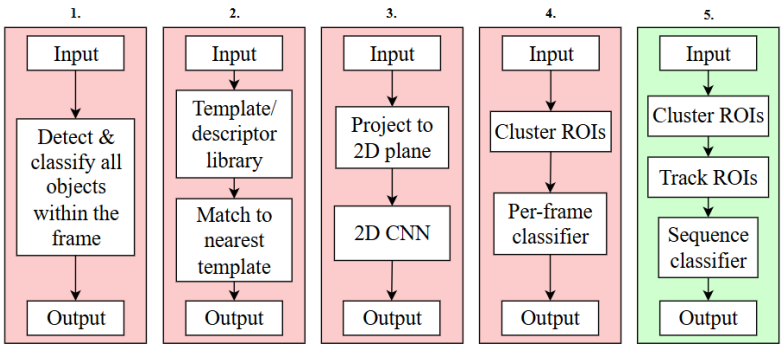


Figure 4.1: Thumbnail diagrams of the five pipelines considered: (1) end-to-end scene detection; (2) template/descriptor matching; (3) projection-only with a 2D CNN; (4) clustering then per-frame classification; (5) the adopted multi-stage localise-track-classify pipeline.

This modular approach provides several benefits:

- **Label efficiency:** Clustering removes background points and reduces manual labelling to a single label per ROI.
- **Computational efficiency:** Each stage operates on compact ROIs, making the system feasible on a laptop GPU.
- **Scalability:** Multiple clusters can be tracked independently.
- **Generalisation:** By decoupling localisation from classification, the model focuses and learns an animal's distinguishable features rather than leakage from the environment.
- **Interpretability:** Intermediate outputs (clusters, trajectories, logits) can be viewed separately to diagnose any issues and optimise each stage.

4.3.3 Data and Recording Strategy

Instead of synchronising a ground-truth camera to each frame captured by the LiDAR. The recording's file name was of the subject matter within each frame, as ground truth for annotating.

4.4 Summary

This section defined the design principles and compared alternative processing pipelines for LiDAR-based animal classification. The chosen modular cluster–track–classify approach balances accuracy, interpretability, and computational efficiency, while remaining scalable to multiple animals per frame, multiple settings and real-time applications. The following chapter presents preliminary experiments that develop and validate each stage of the pipeline before final implementation.

Chapter 5

Preliminary Experiments

This chapter presents three preliminary experiments conducted before acquiring and processing the main animal dataset. They serve as a prototype pipeline: (i) clustering raw LiDAR frames into object proposals, (ii) linking proposals via tracking, and (iii) classifying proposals into various classes. Together they establish feasibility, reveal challenges, and provide a baseline for the subsequent animal study.

5.1 Experiment 1: Clustering - Voxel vs Projection

5.1.1 Objective

The first stage of the pipeline is to extract object candidates directly from raw LiDAR frames. Vehicle-like objects from the KITTI dataset are the targets that are aimed to be clustered out of the environment.[1]

This step reduces about 100k irregular 3D points into a small number of candidate object proposals that the downstream modules can track and classify. Two alternative approaches were compared:

- **Voxel-based (3D):** discretises the LiDAR point cloud into a regular 3D grid of cubes (voxels), keeping the spatial geometry of the point cloud space, but at higher computational cost.
- **Projection-based (2D):** collapses the point cloud into either a *Range View* (spherical projection) or *Bird's-Eye View* (ground-plane projection), which is computationally efficient but removes the depth of the point cloud space.

This experiment investigates the trade-off between accuracy and computational efficiency when isolating and clustering object proposals.

5.1.2 Method Overview

Both clustering approaches follow a common processing pipeline designed to convert raw LiDAR point clouds into candidate object regions suitable for subsequent tracking and classification. The only difference between the two lies in their spatial representation - voxel-based (3D) versus projection-based (2D). An overview of the clustering workflow is shown to the right (Figure 5.1). The text below highlights only key design choices not explicit in the diagram.

[1] Only a subset of the sequences was available at the time of experimentation: 3 of the 12 target sequences could be downloaded. Results should therefore be interpreted as indicative on a limited subset rather than exhaustive over the full benchmark.

1. **Preprocessing:** Apply field-of-view and range masks to remove irrelevant points and to simulate the Livox Avia's ($\approx 70.4°$ horizontal $\times$ $77.2°$ vertical) window.
2. **Representation:** Encode the filtered points as either a 3D voxel grid or a 2D projection (range-view or bird's-eye view), allowing the use of CNNs.
3. **Dense prediction:** A U-Net model [a] is used for both 2D and 3D representations because its encoder-decoder structure balances their contextual features with their fine spatial features, which is important for segmenting the vehicle-like objects.
4. **Mask post-processing:** Threshold the network output, apply morphological opening/-closing and small-hole fill, then connect components to get candidates.
5. **Instance scoring & geometric filtering:** Score each cluster by mean probability; remove unrealistic proposals using filters such as minimum size.

[a] U-Net is an encoder-decoder CNN with symmetric skip connections that pass high-resolution features from the encoder to the decoder. This design allows it to combine fine-grained local features with a comprehensive global view, making it well-suited to dense segmentation (assigning a class label to each element in a representation). Originally, it was designed for biomedical image segmentation and it generalises to both 2D (BEV/RV) and 3D (voxel) inputs.

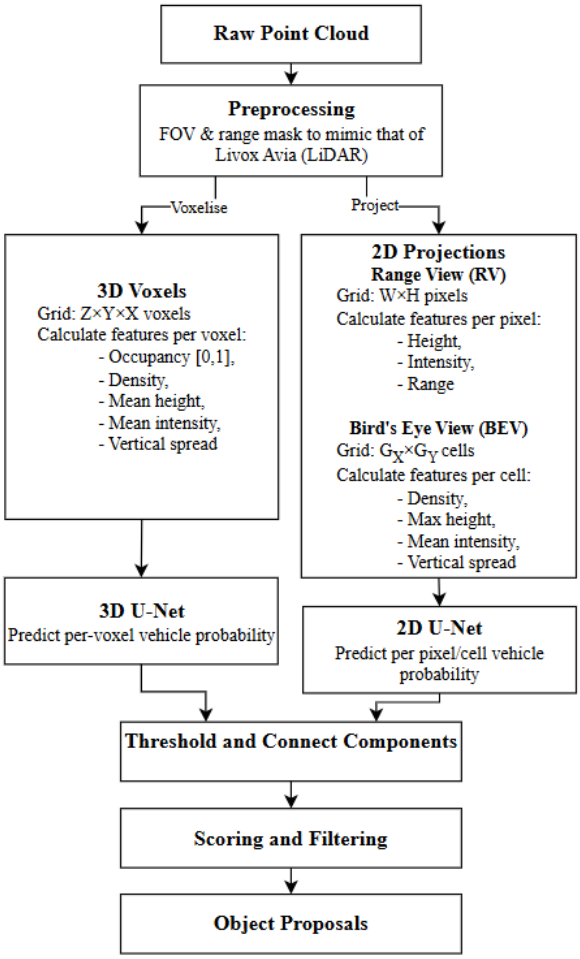


Figure 5.1: Overview of the clustering pipeline. Both voxel-based and projection-based representations share identical preprocessing, segmentation, and post-processing stages, differing only in how the point cloud is encoded prior to U-Net prediction.

This setup allows the direct comparison of both representations, while keeping all other factors constant.

5.1.3 Representations

Voxel grid.

The voxel-based representation grids the point cloud space into a regular lattice of 3D voxels. Each point is assigned to the voxel it falls into, and statistics are calculated over the set of points within each voxel. In this experiment, five channels are used:

1. **Occupancy:** binary indicator of whether the voxel contains any points. This stops the model from being dominated by empty space.
2. **Density (clipped count):** normalised number of points in the voxel, capped to avoid extreme outliers.

3. **Mean intensity:** average intensity of points in the voxel, which helps distinguish materials (e.g. building wall versus car body).

4. **Mean height:** average of the z coordinate; provides information on ground versus elevated objects.

5. **Vertical spread:** variance of the z coordinate, which also indicates if the voxel covers a flat patch (road) or a more volumetric region (car body).

The result is a compact 5-channel 3D tensor that preserves the structure of the point cloud. The drawback is that the grid can be memory-intensive: the size of the voxels must be chosen carefully to balance detail against computational cost.[2]

Projections.
Instead of discretising the full 3D space, the projection approach collapses the point cloud into 2D images that can be processed using 2D CNNs. Two views are considered:

- *Range View (RV):* Each point is projected onto a panoramic image indexed by azimuth (horizontal angle) and elevation (vertical angle). Each pixel stores six channels: range, intensity, Cartesian coordinates (x, y, z) of the return, and elevation angle ϕ. Azimuth θ is implicit in the column index of the RV image (with wrap-around), so it is not stored as a channel.

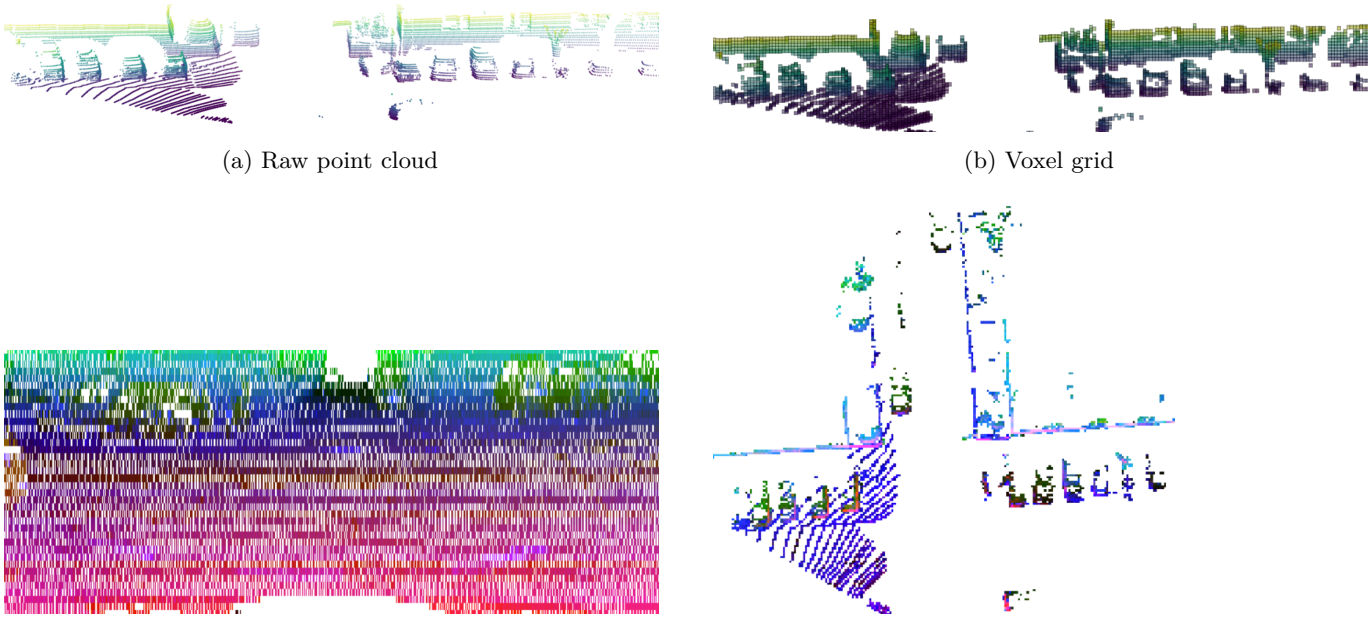
  This view effectively unwraps the sensor's spherical field of view into a rectangular image, preserving angular resolution and the radial structure of nearby objects. However, because all points sharing the same azimuth-elevation pair are mapped to a single pixel, depth variations along that ray are compressed, meaning that distant objects lying behind nearer ones become occluded in the image.

- *Bird's-Eye View (BEV):* Points are orthogonally projected onto the ground plane (XY) and accumulated into a fixed grid of cells (e.g., $G_y \times G_x$ at resolution `VOX_XY`). For each cell with n points, the following is calculated: maximum height $\max z$, mean intensity $\frac{1}{n}\sum I$, log-compressed count $\log(1+n)$, and vertical spread $(\max z - \min z)$. Empty cells are encoded as zeros. The resulting top-down map provides a clear footprint of objects. However, the BEV representation removes a lot of vertical structure, making it less effective for distinguishing objects that overlap in height or appear stacked within the same ground cell.

Both projection encodings result in lightweight 2D feature maps (4-6 channels) with a fixed image size, leading to fast inference with a small 2D U-Net.[3]. Compared to the voxel approach, they are less memory-intensive and orders of magnitude faster, but inevitably sacrifice the full spatial structure, leading to more fragmented or merged clusters in crowded scenes.

[2] **Voxel model (UNet3DCoarse):** 5 input channels (occ, count_norm, mean_i, mean_z, var_z); voxel size 0.2 m; ROI $(x, y, z) = (0{:}40, -25{:}25, -1.5{:}2.0)$ m; FOV $\approx 120° \times 40°$, range 0.5-60 m; base channels 32 (CUDA) / 16 (CPU); AdamW (lr 10^{-3}, wd 10^{-2}); cosine LR ($T_{\max}$=30, $\eta_{\min}$=10^{-5}); batch 1 with grad accumulation 4; epochs 30; loss: weighted CE on occupied voxels (class weights estimated); AMP enabled.

[3] **Projection models (BEV/RV):** SmallUNet2D, base channels 24 (CUDA) / 16 (CPU); inputs-BEV: 4 ch, RV: 6 ch; AdamW (lr 10^{-3}, wd 10^{-4}); cosine LR ($T_{\max}$=20, $\eta_{\min}$=10^{-5}); batch 4; epochs 20; loss $0.6\times$ BCE-with-logits (`pos_weight` estimated from train set) $+\,0.4\times$ Dice; AMP enabled. BEV grid: $(-10{:}70, -35{:}35, -2{:}3)$ m at 0.2 m; RV size: 32×512.



(a) Raw point cloud

(b) Voxel grid

(c) Range-view projection

(d) Bird's-eye-view (BEV) projection

Figure 5.2: Different representations of the same LiDAR frame: (a) raw point cloud, (b) voxel grid, (c) range-view (RV) projection, and (d) bird's-eye-view (BEV) projection.

5.1.4 Evaluation Metrics

Two families of metrics are used to evaluate each approach:

- **Dense segmentation:** the per voxel/pixel/cell correctness of predictions, using Intersection-over-Union (IoU), F_1, Average Precision (AP), and Expected Calibration Error (ECE).
- **Instance-level proposals:** the quality of the detected clusters, measured by proposal AP, precision, recall and purity. AP is the area under the precision-recall curve obtained by sweeping the cluster-confidence threshold. Purity is the fraction of the predicted cluster's points/voxels that belong to the same ground truth instance, averaged over the matched clusters.

All models are evaluated under identical field-of-view, splits, and an IoU threshold of 0.5 for instance matching.

5.1.5 Results

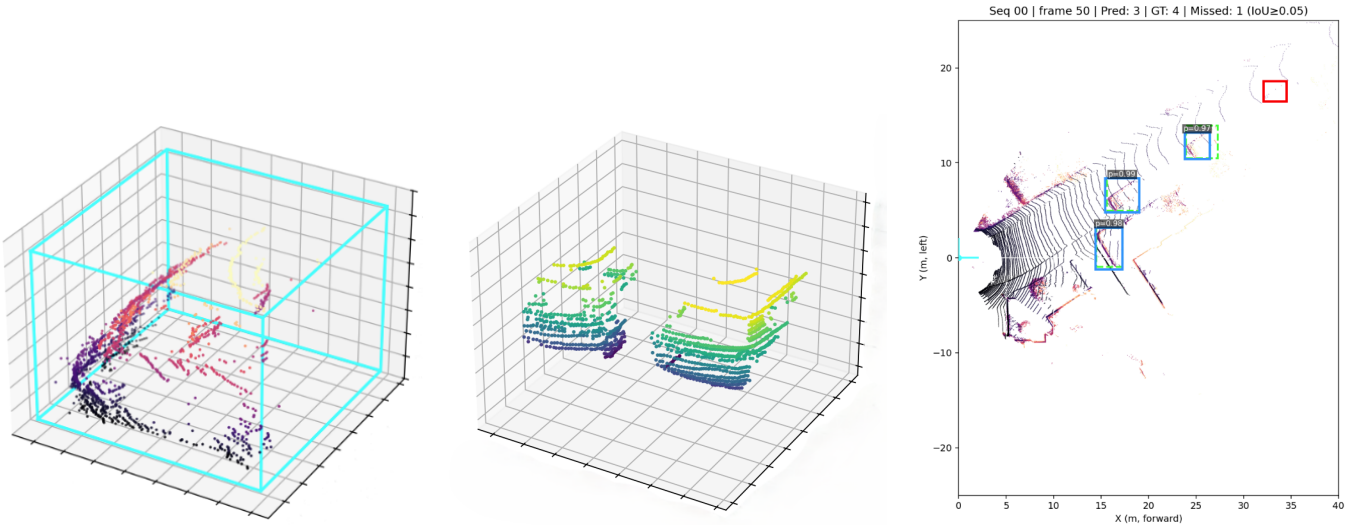
The dense segmentation results are shown in Table 5.1, while the instance-level proposal results are reported in Table 5.2. Qualitative examples are shown in Figure 5.3.

Table 5.1: Dense segmentation metrics (validation set).

Backbone	AP (%)	IoU (%)	F_1 (%)	ECE (%)
Voxel (3D)	96.7	97.3	98.6	0.17
BEV (2D)	89.5	89.8	94.6	0.06
RV (2D)	90.8	90.3	94.9	1.31

Table 5.2: Instance-level proposal quality (IoU ≥ 0.5).

Backbone	Proposal AP (%)	Precision (%)	Recall (%)	Purity (%)
Voxel (3D)	72.5	97.8	71.0	68.6
BEV (2D)	44.1	89.5	46.2	68.0
RV (2D)	22.9	65.6	22.0	67.2



(a) Voxel proposals: clean one-blob-per-car.

(b) BEV projection: two nearby cars merged.

(c) BEV footprint with bounding boxes from the voxel model.

Figure 5.3: Qualitative comparison. (a) Voxel representation yields well-separated clusters. (b) BEV sometimes merges adjacent cars. (c) Voxel-derived boxes visualised in BEV to show footprint alignment.

5.1.6 Complexity and Runtime

Table 5.3: Complexity and runtime comparison. The parameter count, per-frame latency (p50/p90/p95), FPS, and peak GPU memory is reported. All results measured with batch size 1 on RTX 3050 Laptop GPU.

Backbone	Params (M)	p50 [ms]	p90 [ms]	p95 [ms]	FPS @ p50	Peak VRAM [MB]
Voxel (3D)	8.56	215.10	243.40	254.10	4.65	1005
BEV (2D)	0.157	5.40	5.50	5.60	185.19	221
RV (2D)	0.157	1.23	1.68	1.93	810.43	28

5.1.7 Discussion

The voxel approach achieved the highest accuracy and proposal average precision (AP), producing clean, one-blob-per-object detections. However, this came at a cost of approximately 215 ms per frame (4–5 FPS). The projection backbones were much faster (BEV $\sim$5 ms, RV $\sim$1 ms per frame) but often yielded fragmented or merged detections in crowded scenes. These results show the core trade-off: voxel representations keep the full 3D structure but are computationally expensive, whereas projection methods are much quicker and computationally cheaper at the cost of accuracy.

5.2 Experiment 2: Tracking

5.2.1 Objective

This experiment evaluated whether a single moving subject could be reliably clustered, tracked, and subsequently classified when recorded with the Livox Avia. It represents three important firsts in this thesis: (i) the first use of the Livox Avia that will be used for the forthcoming experiments, (ii) the first manual annotation of point cloud frames, and (iii) the first end-to-end test of the clustering, tracking and classification pipeline. The subject was a person walking perpendicular to the LiDAR's FOV at a constant pace, with either a *small* or a *large* laundry basket, at three distances (close, mid, far), giving six recordings in total.

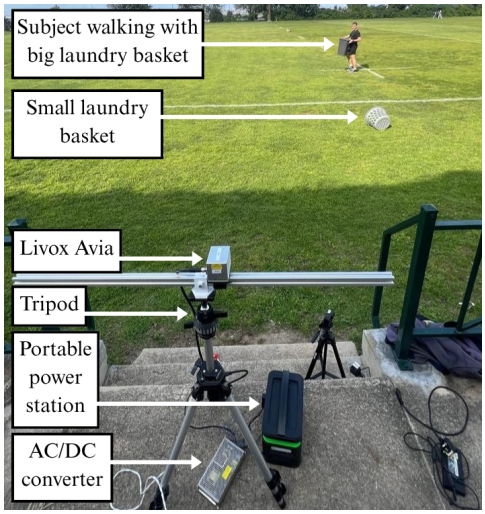


Figure 5.4: Annotated data-collection setup. Labelled items: Livox Avia on a rail and tripod, portable power station, and AC/DC converter; the subject walks with a large laundry basket and a smaller basket is placed in the field. The laptop is out of frame; a GoPro and a laptop charger at the right edge are present but not labelled.

Before moving to uncontrolled environments, this scenario provides a way to (a) validate the full pipeline on the actual LiDAR to be used later (Livox Avia), (b) characterise how the Livox Avia's non-repetitive pattern impacts framing, density, and centroid localisation, (c) provide a first annotation workflow and ground-truth linking, and (d) test a tracker with a single, unambiguous identity. Using two basket sizes adds a simple shape-based classification task that is close enough to the later animal morphology problem to test whether geometry and intensity are sufficient per frame.

5.2.2 Sensor and Data Conversion (Livox Avia)

Each Livox Avia recording was first converted from the native `.lvx` format into `.las` files within the Livox Viewer Software, then split into consecutive per-frame `.pcd` files containing the fields `xyz`, optional `intensity`, `tag`, and `line`. This provides a compact, lossless storage, fast random access to individual frames, and most importantly, direct compatibility with NumPy and PyTorch.

The Livox Avia's non-repetitive beam path gradually shifts after each cycle, with about 24 000 points per revolution, so that subsequent revolutions fill previously unsampled regions of the FOV. This can be seen in Figure 3.1b in chapter 3.

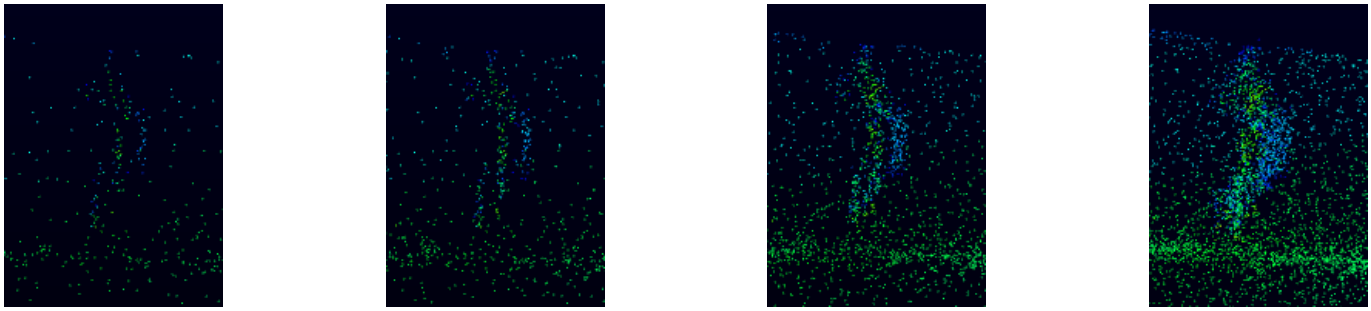
To construct usable frames, each recording was segmented by point count rather than elapsed time, so that

there is a consistent number of samples per frame. Three point counts per frame were considered:

- **240k points/frame (10 revolutions):** high density but noticeable motion smear, as moving subjects appeared stretched along their trajectories.
- **120k points/frame (5 revolutions):** balanced the compromise between density and temporal resolution; subjects appeared sharp with sufficient point density.
- **48k points/frame (2 revolutions):** minimal motion smear, but lower per-frame detail and reduced stability at long range.

120k points per frame was selected as it balanced the trade-off between point cloud density of the ROI and resolution. However, the optimal point budget is inherently task-dependent: dense frames favour detailed shape classification, while sparse frames benefit tracking and motion continuity.

If future datasets have more range- or angle-dependent sparsity, frames could instead be defined dynamically using a continuous stream of points, ensuring each cluster retains a sufficient number of samples for classification. However, this would require redesigning both the data loaders and how the gerneral pipeline flows.



(a) 24k points / frame

(b) 48k points / frame

(c) 120k points / frame

(d) 240k points / frame

Figure 5.5: Frame construction comparison on Livox Avia. Larger frames (240k points) smear moving subjects, while 120k balances resolution and point density.

5.2.3 First Annotations (CVAT Cuboids)

Annotation was completed in this experiment to provide ground truth for object localisation and tracking. Several 3D labelling tools were initially experimented with:

- **Supervisely:** provided semi-automatic cluster tracking but required a paid subscription for large-scale use.
- **LATTE:** had persistent compatibility issues during setup.
- **3D-BAT:** incompatible with the dataset's point cloud-only format and lack of ground-truth camera calibration files.

CVAT (Computer Vision Annotation Tool) was ultimately selected for its stability, open-source accessibility, and native support for 3D cuboid and track ID annotations on point clouds. A total of **207 frames** were annotated, each containing labelled objects with 3D bounding cuboids defining class and track ID.

The cuboid metadata were exported from CVAT as JSON files. Finally a python script was created to link these JSON files to their corresponding frames, with the file directory of the dataset shown as follows:

```
Finished_dataset/
|-- close_big/                      # name of bucket
|   |-- index.jsonl                 # frame manifest for this sequence
|   |-- labels/                     # per-frame labels (centres, classes, etc.)
|   |   |-- close_big_0000000000.json
|   |   |-- close_big_0000000001.json
|   |   |-- ...
|   |-- points/                     # per-frame point clouds (binary for loaders)
|       |-- close_big_0000000000.bin
|       |-- close_big_0000000001.bin
|       |-- ...
|-- close_small/
|   |-- index.jsonl
|   |-- labels/ ...                 # *.json
|   |-- points/ ...                 # *.bin
|-- ...
```

Listing 5.1: Final directory structure of the annotated LiDAR dataset.

This structure enables consistent access to both the raw point-cloud frames and labels during evaluation and training. Metrics such as the centroid localisation error and Hit@r are calculated from this cuboid metadata.

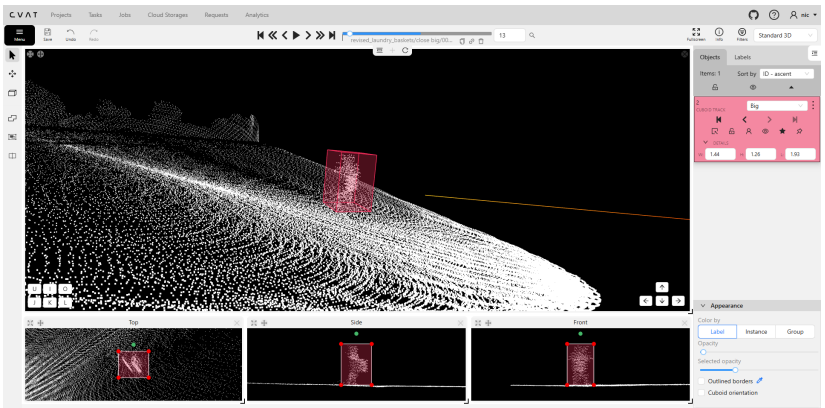


Figure 5.6: Example of CVAT cuboid annotation on Livox point clouds. The cuboids provide ground-truth centres and labels for evaluating localisation accuracy and creating crops of the ROIs.

5.2.4 Dataset & Storage Layout

Every stage of the pipeline produces its own output, making the pipeline reproducible and allowing any step to be repeated without affecting the others.

- Raw recordings (`.bin`): raw point cloud frames.
- Per-frame arrays (`.npz`): compressed NumPy tensors with preprocessed point clouds (`xyz`,`intensity`). It provides a compact container that the data loaders and PyTorch can use directly.
- `detections.csv`: one row per processed frame with the detector's peak centre (x, y), confidence score, a pointer to the saved ROI crop (`roi_npz`), ROI point count, distance, and azimuth angle. This is between clustering and tracking.
- ROI crops (`.npz`): point subsets centred on the proposed instances (ROIs); used by the classifier so it does not have to re-segment the full frame.
- `tracks.csv`: per-frame constant-velocity Kalman filter states $(x, y, \dot{x}, \dot{y})$ with tracking metadata (e.g., bucket, frame id, confirmed flag) for trajectory visualisation and sequence construction.

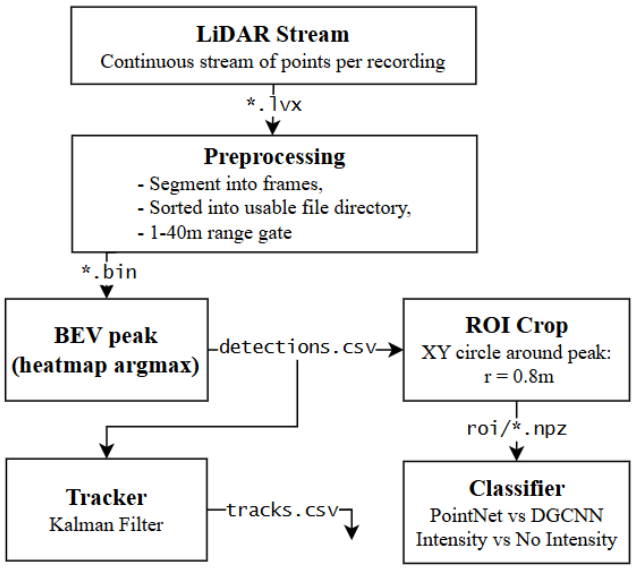


Figure 5.7: End-to-end pipeline. Livox `.lvx` is framed and range-gated during preprocessing, producing per-frame point arrays (XYZ[+I]). A TinyFCN BEV detector writes `detections.csv`, that integrates with (a) a Kalman-filter tracker that writes `tracks.csv`, and (b) ROI cropping on the original frame points to `roi/*.npz`, which are classified by a model (PointNet/DGCNN). In a full system the classifier would take in sequences of ROI crops with help from the tracker rather than single-frame crops; here we evaluate single-frame ROIs for simplicity.

5.2.5 Clustering for Tracking (BEV Heatmap Approach)

Guided by Exp. 5.1, the objective is to locate the target object's centroid per frame ideally at real-time speed. Rather than dense 3D segmentation, each LiDAR frame is rasterised into a bird's-eye-view (BEV) grid, and a lightweight head predicts a heatmap, in which the peaks indicate object centroids.

Representation. Five vertical statistics are calculated for each BEV cell: log-density, $z_{\max}$, $\bar{z}$, σ_z, and the count of occupied z-bins. These features are inexpensive to calculate, mitigate Livox range/azimuth sparsity, and retain the ground-plane geometry required by the tracker.

Architecture (BEV heatmap head). A small, resolution-preserving Fully Convolutional Network (FCN) outputs a single-channel BEV heatmap:

$$\mathrm{Conv}_{3\times3}(5\to32)\ \Rightarrow\ \mathrm{Conv}_{3\times3}(32\to64)\ \Rightarrow\ \mathrm{Conv}_{3\times3}(64\to64)\ \Rightarrow\ \mathrm{Conv}_{1\times1}(64\to1),$$

all with ReLU, stride 1, and the same padding (∼57k params). With no downsampling, the output is aligned with the BEV cells and runs at very low latency.

Why not a U-Net here? The target is a single, well-localised centroid peak in BEV and not a dense object mask. A shallow, stride-1 FCN is therefore a better fit than an encoder-decoder for four reasons: (i) **Alignment:** with no down/up-sampling, the heatmap pixels stay exactly aligned with BEV cells (peak maps directly to world (x, y), avoiding quantisation drift and peak blurring that can arise from pooling and decoding; therefore lets the Kalman filter treat it as a simple point measurement. (ii) **Latency/size:** the FCN head (∼57k params) is orders of magnitude smaller and faster than a U-Net of a comparable receptive field, which matters because detection runs every frame before the rest of the pipeline. (iii) **Small Dataset:** with limited data, an FCN also reduces overfitting and training complexity. (iv) **Sufficient context:** for centroiding, the receptive field can also be grown cheaply (e.g., extra 3×3 layers or dilations) without the overhead of a full encoder-decoder.

By contrast, U-Nets work well when full masks are required. A U-Net is used in Exp. 5.1 as it is a dense segmentation setting, as in assigning a label to every voxel/pixel. If this stage later needs full masks, a shallow U-Net or a dilated FCN would then be appropriate.

Training target & loss. The ground-truth cuboid centres are mapped to BEV coordinates and rendered as a normalised Gaussian disk (Fig. 5.8) with pixel radius

$$r_{\mathrm{pix}} = \mathrm{round}\Big(\tfrac{\texttt{POS\_RADIUS\_M}}{\texttt{VOX\_XY}}\Big),$$

and a spread σ chosen so that most mass lies within r_{pix} ($\sigma \approx r_{\mathrm{pix}}/2$ is used in practice). This produces a smooth target $y_{ij} \in [0, 1]$ whose peak sits on the centroid and tapers radially; the FCN predicts a heatmap of the same size with $\hat{y}_{ij} = \sigma(s_{ij})$ from logits s_{ij}.

Because naturally almost all the BEV pixels are background, a sigmoid head is optimised with weighted binary cross-entropy to prevent the background from dominating. Let the per-pixel weight be:

$$w_{ij} = 1 + \lambda\, y_{ij} \quad (\lambda = 4),$$

so pixels near a peak (where $y_{ij} \approx 1$) contribute $\sim 5\times$ more to the loss than further away background ($y_{ij} \approx 0$). The objective is

$$\mathcal{L} = \frac{1}{HW} \sum_{i,j} w_{ij} \Big[-\ y_{ij} \log \hat{y}_{ij} - (1 - y_{ij}) \log(1 - \hat{y}_{ij}) \Big].$$

Intuitively, the Gaussian target gives the network a broad 'hill' to climb, while the weight w_{ij} concentrates learning on the central region so the predicted peak becomes sharp and well-localised. The choice λ=4

worked reliably across the data (larger values sharpen the peak further but can destabilise training).

For multiple objects, the supervision becomes a multi-peak heatmap done by combining per-centre Gaussians (e.g., $y_{ij} = \min\{1, \sum_k G_k(i,j)\}$ or the probabilistic union $1 - \prod_k (1 - G_k(i,j))$). Inference then selects the top-K local maxima or applies non-maximum suppression[4], scaling the same head to multi-object frames.[5]

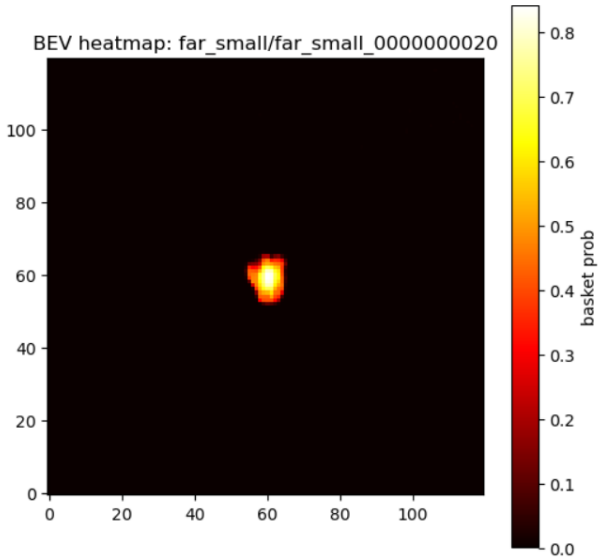


Figure 5.8: BEV heatmap produced by the FCN. Brighter regions indicate higher confidence; the bright Gaussian-like peak aligns with the annotated centroid. The visible spread roughly matches r_{pix} from the target construction. In multi-object scenes, multiple peaks would appear before NMS/top-K selection.

Inference & ROI extraction. Local maxima are extracted from the heatmap above a confidence threshold, then filtered via non-maximum suppression (NMS) with a BEV radius matched to the expected inter-centroid spacing. For each surviving peak, a fixed-radius (0.8 m) is cropped around the ROI and passed downstream. Simple filters such as minimum peak heat, minimum ROI point count, local peak prominence, ring-contrast, and minimum vertical z-span helps get rid of flat ground patches and boundary clutter, and improves tracking at minimal per-frame computational cost.

Tracker integration and scalability. Each accepted peak initialises or updates a Kalman filter track (Section 5.2.6). Although this experiment contains a single subject, the detector-tracker interface scales to multiple objects by (i) allowing multiple peaks in the heatmap, (ii) using NMS/top-K selection, and (iii) maintaining one filter per identity.

This design is computationally efficient as it is a projection-based model. Futhermore, it restores the spatial stability required for tracking. The heatmap provides precise, well-aligned centroids; and trains in minutes on a laptop GPU and runs in real time.

[4]**Non-maximum suppression (NMS).** To avoid multiple detections of the same object, BEV peaks are sorted by score, the strongest are kept, and any other peak within a fixed radius r is deleted; then repeat on the rest. r is set roughly to the object/ROI diameter so duplicates are removed but separate objects remain. (Soft-NMS lowers neighbours' scores instead of deleting them.)

[5]Reproducibility: AdamW (lr 2×10^{-3}), batch 8, 25 epochs; weighted BCE with w=1+4y; BEV cell `VOX_XY` = 0.10 m (5 channels); target radius `POS_RADIUS_M` = 0.6 m; ROI = 0.8 m; AMP when available.

Unsupervised baselines (abandoned). Before adopting the BEV heatmap approach, **DBSCAN** and simple **KNN**-based grouping over (x, y, z) were experimented with, as the small dataset was initially considered insufficient for supervised learning.

- **DBSCAN** suffered from range-dependent density variation: when it was adapted for near-range objects it over-merged distant ones, and when tuned for far-range data it often fragmented nearby clusters.
- **KNN** centroiding was similarly unstable, drifting toward ground patches and occasionally producing false positives near the FOV boundary.

Because both the tracker and the downstream classifier assume that each ROI is a target and is consistently centred on it, those inconsistencies were unacceptable. Therefore, the supervised BEV heatmap was used for robustness and repeatability, even with limited training data: on the validation split it achieved **Hit@0.6 m = 100%** and a **mean centroid error of** ≈ 12.7 **cm**.

5.2.6 Tracker Design

Each bucket was processed independently, producing exactly one confirmed track per sequence. The design goal was to maintain the object's identity across time, avoiding fragmentation and ensuring consistent IDs. The sampling period between consecutive frames was fixed at $\Delta t = 0.5\,\text{s}$, corresponding to the typical spacing of the BEV detections. Although a single-object tracker could, in theory, link ROIs sequentially without formal filtering, a framework was developed to enable future scalability to multi-target scenes.

Tracking and motion prior. Object trajectories were maintained using a constant-velocity Kalman Filter (KF) on the BEV plane. Each object's state was represented as $\mathbf{x} = [x\ y\ \dot{x}\ \dot{y}]^\top$, where (x, y) denote the centroid position and $(\dot{x}, \dot{y})$ its planar velocities. The state transition model $\mathbf{F} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ predicted the next position given the estimated velocity, while the process noise $\mathbf{Q}$ accounted for unmodelled accelerations. At each step, a new centroid detection $\mathbf{z} = [x\ y]^\top$ was used to correct the prediction using the standard Kalman update equations. The measurement model $\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$ linked the state to the observed position, and the measurement covariance $\mathbf{R}$ was estimated from the BEV heatmap's confidence spread. This set up smoothed the trajectories and improved robustness to uncertainty. The complete formulation and details are presented in Appendix B.

Births: entry-ring gating and seeding. A new track could only be born near the FOV boundary, within the outer radial ring $[R - r_{\text{entry}},\ R]$, or at the start of the sequence if the subject was already visible. This enforces the physical constraint that an object cannot appear mid-frame unless entering from the edge of the FOV. Candidate detections meeting this condition must remain for N_{spawn} consecutive frames before being promoted to an active track. New births during the final N_{last} frames were forbidden to prevent short-lived end-of-sequence fragments.

Even with only one target, this logic would be the same for multiple targets.

Association and gating. After each prediction step, the filter propagated the state and covariance forward:

$$\mathbf{x}_{k|k-1} = \mathbf{F}\mathbf{x}_{k-1|k-1}, \qquad \mathbf{P}_{k|k-1} = \mathbf{F}\mathbf{P}_{k-1|k-1}\mathbf{F}^\top + \mathbf{Q}.$$

Given a new detection $\mathbf{z}$, the likelihood of a match was evaluated via the Mahalanobis distance:

$$d^2 = (\mathbf{z} - \mathbf{H}\mathbf{x}_{k|k-1})^\top \mathbf{S}^{-1}(\mathbf{z} - \mathbf{H}\mathbf{x}_{k|k-1}), \qquad \mathbf{S} = \mathbf{H}\mathbf{P}_{k|k-1}\mathbf{H}^\top + \mathbf{R}.$$

A χ^2 gate at 95% confidence in 2D ($d^2 \leq \chi^2_{2,0.95} \approx 5.99$) defined the valid association region. If multiple detections passed the gate, the smallest d^2 was selected.

Rescue and canonicalisation. If a confirmed track was temporarily unmatched but a detection lay within a small Euclidean radius ($\leq$1.5 m), a one-frame 'rescue' linked the pair to prevent fragmentation due to a single missed detection. If duplicate tentative tracks arose (e.g., from a split BEV peak), a canonical representative was selected based on the following hierarchy: confirmed status, then track age, number of successful hits, and finally lowest uncertainty $\mathrm{tr}(\mathbf{P})$.

Complexity. In the sequences there is at most one confirmed track and a few tentative ones per bucket, so all operations are $O(1)$ per frame and execute in real time on a CPU. More generally, the same logic is $O(1)$ per track; multiple objects are handled by instantiating one filter per object with the same gating/association rules, and the system remains real time for small M.

5.2.7 Metrics

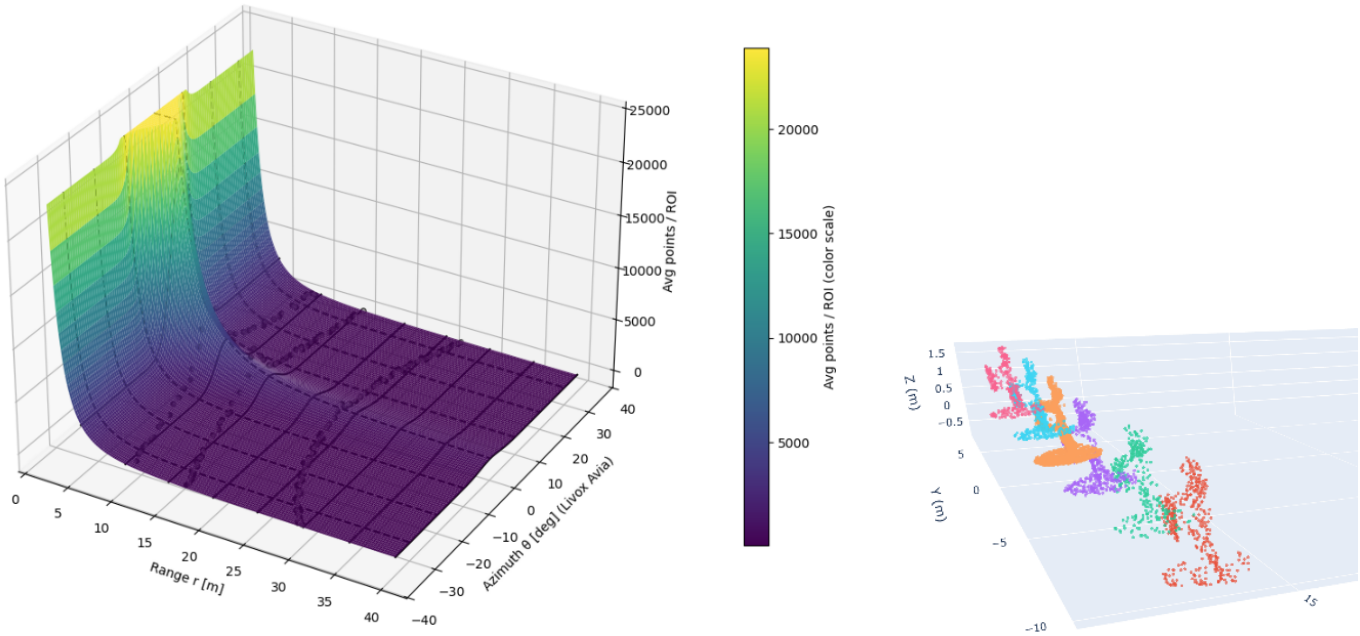
Tracking performance was evaluated against the ground truth cuboid centres using four metrics:

- **Coverage (%):** proportion of frames containing a confirmed track ID. High coverage indicates reliable persistence without fragmentation.
- **Spawn latency (frames):** number of frames required before a detection is promoted to a confirmed track.
- **Error (m):** Euclidean distance between predicted and annotated centres, reported as both mean and median to record bias.
- **Hit@r (%):** fraction of frames where the centroid lies within a fixed radius r of the ground truth.

Table 5.4: Per-bucket tracking metrics combining detector-tracker centre error (mean/median) with overall coverage. Coverage denotes the percentage of frames with a confirmed ID, and Hit@0.60 is the proportion of frames with centroid error ≤ 0.60 m.

Bucket	Frames	Coverage (%)	Mean err [m]	Median err [m]	Hit@0.60 (%)
close_big	20	95.0	0.124	0.111	100.0
close_small	19	94.7	0.140	0.136	100.0
mid_big	29	96.6	0.144	0.126	96.6
mid_small	30	96.7	0.107	0.097	100.0
far_big	51	98.0	0.139	0.110	96.1
far_small	50	98.0	0.113	0.109	100.0
Overall	199	**97.0**	**0.127**	**0.114**	**98.5**

All six buckets achieved over 94% coverage, showing stable, uninterrupted tracking across the diverse ranges. Centroid localisation errors remained below 0.15 m on average, with median errors under 0.12 m for all cases. The high Hit@0.60 scores (96-100%) show that clustering stage achieves consistent localization of the predicted from the ground-truth positions. The results demonstrate that the BEV heatmap detector combined with the Kalman filter tracker provides precise and reliable clustering and tracking even under sparse LiDAR sampling.



(a) Average LiDAR points per ROI as a function of range r and azimuth θ (Livox Avia frame). Black dots are per-ROI counts measured from the tracking dataset; the coloured surface is a smooth fit $N(r, \theta)$ used to extrapolate point density beyond the samples.

(b) Example tracked trajectory for bucket `mid_big`. Each colour represents a consecutive frame sampled every fifth timestep.

Figure 5.9: Point density model and an example tracked trajectory.

5.2.8 Classification of Basket Size

The classification part of this experiment distinguished *small* vs. *large* baskets. Because basket size is geometric rather than behavioural or motion-based, temporality was not exploited here (and the dataset was too small too divide up into sequences). Each ROI was thereby classified independently. Three architectures were tested:

PointNet-style Classifier (Set-based)

The PointNet family of networks treats each ROI as an unordered set of 3D points, rather than a fixed grid or image. Each point $\mathbf{x}_i = [x_i, y_i, z_i, I_i]$ is processed independently through a small multi-layer perceptron (MLP) (shown in Appendix A) whose weights are shared across all points. This produces a per-point feature vector $\mathbf{h}_i$ that encodes geometric and intensity information. Because the same MLP is applied to every point, the network is invariant to the order of points.

Feature aggregation. After the MLP, a symmetric pooling operation (typically a global maximum or average) aggregates all $\mathbf{h}_i$ into a single feature vector $\mathbf{g}$ that summarises the entire ROI:

$$\mathbf{g} = \max_i \mathbf{h}_i.$$

This descriptor captures the overall 3D structure of the object. Additional simple ROI-level features such as point count or axis-aligned size ranges may optionally be concatenated to $\mathbf{g}$ for better generalization.

Classification head. The global descriptor $\mathbf{g}$ is passed through a lightweight MLP head to produce class logits $\boldsymbol{\ell} \in \mathbb{R}^K$, where $K = 2$ (`small`/`large`). Class probabilities are then obtained via the softmax function $p = \text{softmax}(\boldsymbol{\ell})$.

This architecture is simple, fast, and inherently robust to varying point counts. It is especially effective when discriminative features lie in the overall shape or size of the object. Intensity values are z-scored per ROI to remove exposure bias and often improve separability by providing information on material (e.g., plastic vs. fabric). The main limitation is that local geometric relationships between nearby points are not explicitly modelled before pooling - fine details such as rim edges or handles can be lost, especially when sampling is sparse.

PointNet++ Classifier (Hierarchical Set-based)

Concept. PointNet++ extends the original PointNet by introducing a hierarchical feature extraction mechanism that captures local geometric structures at multiple spatial scales. Instead of processing all points independently, the network performs iterative sampling and grouping: representative points are selected, and local neighbourhoods are formed using radius- or k-nearest neighbour searches. A small PointNet module extracts features that describe the local curvature, density, and spatial context within each region.

Hierarchical abstraction. Features are progressively aggregated across layers, allowing the model to focus on both fine-grained local and global features. This bottom-up hierarchy is conceptually similar to

convolutional feature pyramids in images, and allows the network to see finer details rather than just overall shape.

Although it is more expressive than PointNet, PointNet++ is sensitive to non-uniform point densities and sparse sampling, which is typical in those from the Livox Avia. At long ranges or angles far from the centre, where point density is sparser, the local grouping step can become unstable or degenerate. Nevertheless, it provides an informative upper bound for how hierarchical set-based models perform under these types of conditions.

Graph-based Classifier (DGCNN-style)

Concept. Unlike PointNet, which processes each point independently, graph-based models explicitly encode local geometric structure. Each frame is represented as a dynamic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the vertices correspond to points and edges connect each point to its k nearest neighbours ($k \approx 16$). This neighbourhood graph is recomputed at each layer to capture progressively richer geometry as the features evolve.

EdgeConv operation. Local structure is captured using the *EdgeConv* operator. For every point i, features are computed by contrasting its representation $\mathbf{h}_i$ with that of its neighbours $\mathbf{h}_j$:

$$\mathbf{e}_{ij}^{(\ell)} = \phi^{(\ell)}\Big([\mathbf{h}_i^{(\ell-1)} \,\|\, (\mathbf{h}_j^{(\ell-1)} - \mathbf{h}_i^{(\ell-1)})]\Big), \quad \mathbf{h}_i^{(\ell)} = \max_{j \in \mathcal{N}_k(i)} \mathbf{e}_{ij}^{(\ell)}.$$

Here, $\phi^{(\ell)}$ is a small shared MLP and $\mathcal{N}_k(i)$ denotes the neighbours of point i. The subtraction term shows the contrast between points - how each neighbour differs from the centre point - while the max operator aggregates responses within the neighbourhood, making the operation translation-invariant.

Feature aggregation. Several EdgeConv layers (e.g., 64-128-256 channels) are stacked, and their intermediate outputs are concatenated (dense skip connections) to retain geometric information at different scales. A final global max + mean pooling aggregates all node features into a single graph-level descriptor, which is then passed to an MLP classifier to produce logits $\boldsymbol{\ell} \in \mathbb{R}^K$ and corresponding probabilities $p = \mathrm{softmax}(\boldsymbol{\ell})$.

By modelling local contrasts before pooling, graph-based methods capture fine-grained shape cues such as basket rims, handles, or curved edges (features often lost with PointNet). This makes them more robust in distant or partially occluded views where fewer points are available. The trade-off is computation though: each EdgeConv layer requires building and processing kNN graphs, so memory and training time scale roughly with both N (points) and k.

Effect of intensity. Across the point and graph-based models, adding per-point intensity (`XYZI` vs. `XYZ`) consistently increased accuracy. Intensity correlates with material/texture (basket vs. clothing/skin) and therefore differentiates basket points from the person even when point clouds are sparse.

Table 5.5: Overall classification metrics over K folds.

Model	Intensity	Prec	Rec	F1	F1	Acc (mean±std)
Point	No	0.929	0.929	0.929	0.929	0.9290 ± 0.0187
Point	Yes	0.945	0.945	0.945	0.944	0.9440 ± 0.0301
Graph	No	0.925	0.924	0.924	0.924	0.9237 ± 0.0431
Graph	Yes	**0.959**	**0.960**	**0.960**	**0.959**	$\mathbf{0.9594 \pm 0.0125}$
PN2	Yes	0.837	0.826	0.826	0.827	0.8278 ± 0.0701

Table 5.6: Grouped accuracies (means across folds). Range bins: `close`, `mid`, `far`. Angle bins: `center`, `offcenter`.

Model	Intensity	Accuracy @ Range			Accuracy @ Angle	
		Close	Mid	Far	Center	Offcenter
Point	No	0.978	0.983	0.891	0.940	0.928
Point	Yes	1.000	0.983	0.911	0.979	0.918
Graph	No	0.918	0.982	0.890	0.950	0.899
Graph	Yes	0.980	0.983	0.940	0.960	0.959
Pointnet++	Yes	0.971	0.900	0.698	0.870	0.760

Table 5.7: Model complexity and inference latency per ROI.

Model	Intensity	Params	Latency mean (ms)	Latency median (ms)
Point	No	109,059	0.35	0.34
Point	Yes	109,123	0.35	0.34
Graph	No	353,027	5.20	5.18
Graph	Yes	353,155	5.21	5.19
PointNet++	Yes	1,462,147	47.82	37.04

5.2.9 Discussion

This experiment confirmed the end-to-end feasibility of detecting, tracking, and classifying objects using the Livox Avia LiDAR within a compact, reproducible pipeline. The Kalman tracker achieved near-perfect coverage with predictable degradation at greater distances where the non-repetitive scan pattern yields sparser returns.

Classifier performance. Table **??** shows that all point-based and graph-based architectures successfully separated `small` and `big` object classes, with the graph-based DGCNN outperforming both PointNet and PointNet++ across all metrics. Incorporating intensity consistently improved precision and recall, as reflectivity provides an additional discriminative cue beyond shape alone. The PointNet++ variant, while more expressive in principle through its hierarchical local feature extraction, underperformed on this sparse, low-density dataset. Its reliance on neighbourhood grouping and radius-based sampling made it sensitive to range-dependent sparsity - particularly for far and off-centre objects with fewer points, whereas PointNet remained more robust. However, DGCNN's explicit modelling of local geometry helped mitigate this effect the most. Absolute F1 scores remained modest (0.89-0.94) given the simplicity of the dataset. The limited number of labelled examples, especially across varying ranges and viewing angles, prevented the networks

from learning stable representations of the same object under different sampling densities. Objects appearing near the edges of the field of view or at longer ranges were represented by far fewer points, thereby weakening the structure of the object and caused occasional misclassification between the two size classes.

Summary. Despite a small and tightly controlled dataset, the system demonstrated stable tracking and meaningful classification capability using only LiDAR data. The findings validate the feasibility of the proposed Livox-based approach and motivate the use of spatio-temporal networks and broader sampling ranges in the main animal dataset to achieve robust species- and behaviour-level recognition.

5.3 Experiment 3: Fine-Grained Classification

5.3.1 Objective

The aim of this experiment is to classify isolated, tracked ROIs into movement categories: **walking**, **running**, **jumping**, and **side-shuffling**. A single participant performed these four motions around the LiDAR moving around in different directions across the field of view (FOV). This setup was chosen to test the full pipeline under faster, non-linear motion and to quantify how much temporality helps beyond the per-frame shape of an object. The questions going into this experiment were: (i) can the BEV-peak clustering and tracking still produce clean, identity-consistent ROIs under non-linear motion? and (ii) how much do temporal classifiers improve over per-frame classifiers?

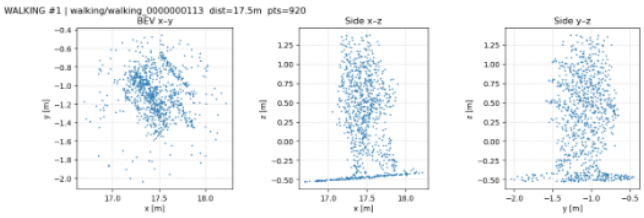
A single participant performed the four movements on a flat field. The set contains 487 frames within $d \leq 35\,\mathrm{m}$ of the sensor (further ranges up to $\sim$90 m are left for future work).

5.3.2 Pipeline Changes for Dynamic Motion

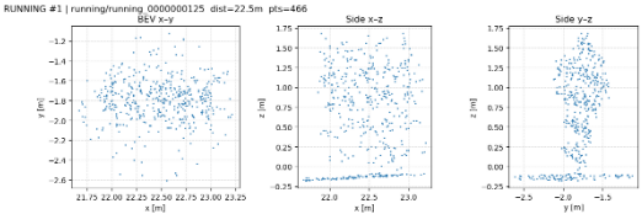
Relative to the basket study, clustering and tracking were adjusted to cope with faster motion and greater frame-to-frame displacement.

Framing and crop radius. Frames are again divided by 120k points each (Sec. 5.2.2). Because trajectories are faster here, the ROI crop radius was enlarged from $0.8\,m$ to $1.0\,m$ (XY), preventing truncation when the BEV peak leads/lags across successive frames.

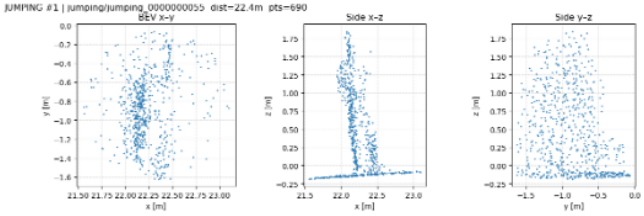
Clustering (BEV heatmap) changes. The same BEV heatmap regressor from the previous experiment was used, with two minor changes designed to improve stability for this dataset: (i) a slightly larger Gaussian target radius (`POS_RADIUS_M`) was used when creating the training targets, which enlarges the supervision zone around the ground-truth centroid. This reduces sensitivity to small label noise and allows the network to produce smoother peaks when the object moves rapidly between frames. (ii) A higher minimum point-count threshold to suppress background patches, preventing spurious peaks from being interpreted as valid detections. This does effectively remove most noise in the dataset, but future implementations should replace it with an adaptive, learned confidence measure that accounts for range/angle-dependent sparsity.



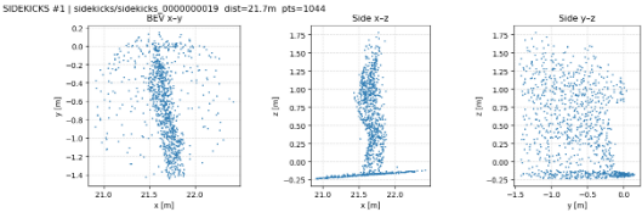
(a) Walking



(b) Running



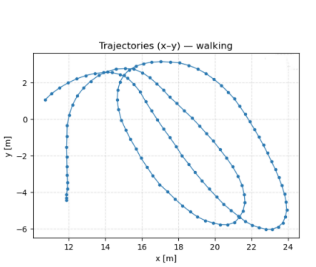
(c) Jumping



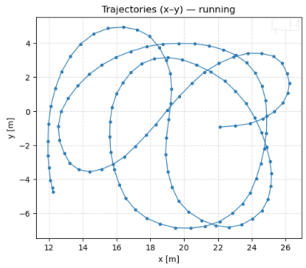
(d) Side-shuffle

Figure 5.10: Example ROI point clusters for each movement type, shown in BEV (x-y) and side (x-z, y-z) projections. Each frame captures a single tracked subject within a 1 m ROI crop.

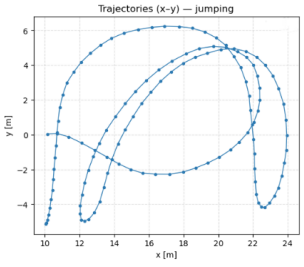
Tracker (CV model) changes. The tracker retained the same Kalman filter algorithm as described in Section 5.2.6, using the same entry-ring birth logic and Mahalanobis gating. Here, it is implemented as a single-ID reference tracker because each recording contains only one moving subject. However, the pipeline can account for multiple objects per frame: the BEV heatmap and NMS handles multiple peaks per frame; one ROI (and one `detections.csv` row) is recorded per peak; and the tracker (produces `tracks.csv` from `detections.csv`) supports multiple rows per frame with distinct `track_id`s. The process noise terms for velocity $(q_{\dot{x}}, q_{\dot{y}})$ were also increased to handle higher accelerations shown in the movement sequences (e.g., running or sudden turns). This made the filter more responsive to rapid trajectory changes without compromising stability. Because of all this, the tracks remained continuous and complete, with no fragmented trajectories.



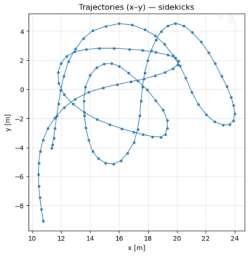
(a) Walking



(b) Running



(c) Jumping



(d) Side-shuffle

Figure 5.11: Tracked trajectories (x-y) for the recordings of the four motion types. Each node corresponds to a frame in time.

5.3.3 Sequence Construction and Splits

Tracks are decomposed into non-overlapping sequences of $L = 4$ frames with a stride of 4. Sequences inherit the track label (`walk`/`run`/`jump`/`side`). The same k-fold (`folds.json`) validation split is used for reproducibility

across methods.

5.3.4 Methods

All methods below use intensity, motivated by Experiment 2 results showing consistent gains from adding it as a feature.

M1 - Per-frame Graph Classifier with Majority Vote

Each frame in a sequence is classified independently using a DGCNN (EdgeConv) backbone, identical to the graph architecture introduced in Section 5.2.8. For a sequence of L frames, the model outputs a set of per-frame class probabilities $\{p_t(y)\}_{t=1}^{L}$, where each $p_t(y)$ is a probability distribution over the K movement classes. The sequence-level prediction is then obtained by averaging these probabilities across time:

$$\hat{y}_{\text{seq}} \;=\; \arg\max_{y} \frac{1}{L} \sum_{t=1}^{L} p_t(y),$$

which is equivalent to a probability-weighted majority vote across frames. This approach emphasises classes that are consistently predicted over time, while reducing the influence of occasional noisy or ambiguous frames.

The use of mean (rather than hard majority) voting generally produced higher stability and accuracy, as it preserves the relative confidence of each frame's prediction rather than counting all frames equally.

Increasing the number of frames L would improve robustness and accuracy by averaging over longer temporal windows, but also increases the amount of training data needed, computational load and latency - limiting real-time feasibility. In future deployments, this trade-off could be investigated by using fewer points per frame.

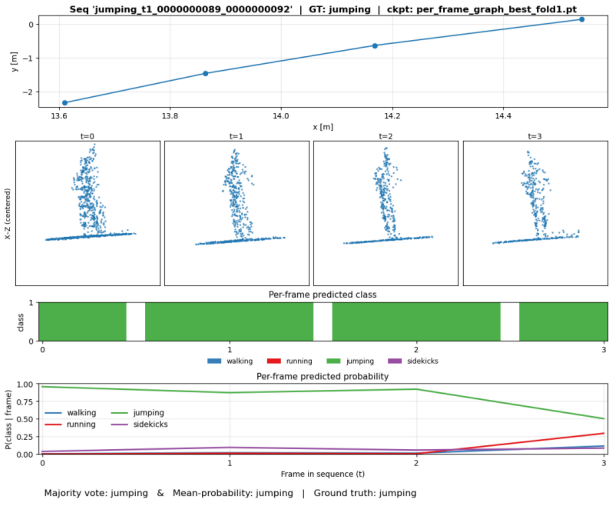


Figure 5.12: Example of the **M1** per-frame DGCNN classifier on a `jumping` sequence. Top: (x, y) tracks. Middle: per-frame point clouds (x, z). Bottom: predicted classes and probabilities. The model correctly identifies `jumping` by majority and mean-probability votes.

M2 - Stacking Frames of a Sequence with a Point-based Classifier

In this approach, all L=4 consecutive frames of a sequence are stacked into a single point cloud. Each point retains its spatial coordinates (x, y, z) and intensity I, and is assigned an additional channel indicating the frame index (a frame colour in **??**). This gives an extra dimension of time, allowing the network to see motion without requiring an explicit temporal model.

Because stacking frames significantly increases the total point count, a **PointNet**-style model is used instead of a graph-based model to keep computation tractable (as described in Section 5.2.8).

This method implicitly captures motion, since the stacked frames contain shifted copies of the same subject. But it does not preserve per-frame structure, making it difficult for the network to disentangle the dynamics of the sequence from the static point clouds. As a result, performance was found to be sensitive to class imbalance in the merged samples, particularly when one motion type contributed more frames or denser point regions than another.

While it is computationally simple, this method does highlight a key limitation of a purely spatial model as temporality is only indirectly represented.

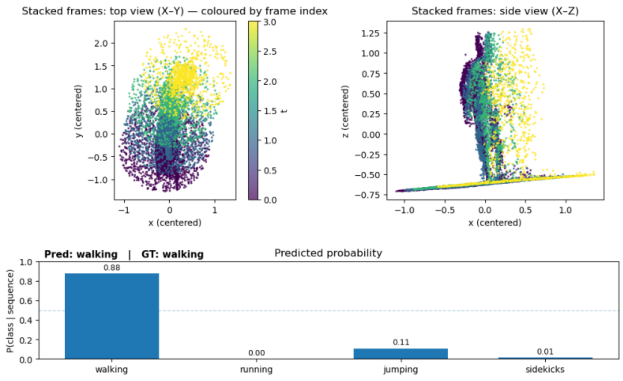


Figure 5.13: Example of the **M2** stacked-frames PointNet classifier. The left panels show the same ROI stacked over t=4 frames and coloured by frame index: top view (x, y) (left) and side view (x, z) (right). The bottom bar chart gives the sequence-level class probabilities; here the model correctly predicts `walking`.

M3 - Two-stage: Spatial Logits into a Temporal Classifier This method separates spatial and temporal classification into two sequential stages. First, a strong per-frame spatial classifier, in this case, DGCNN is trained independently to predict movement classes from single ROI frames. Instead of taking the final softmax probabilities, which are normalised to sum to one and therefore lose information about the relative confidence. The pre-softmax activation vectors (logits) $\ell_t \in \mathbb{R}^K$ are extracted for each frame, where K=4 corresponds to the four classes. Each logit vector, ℓ_t, contains the raw, unnormalised scores output by the network's final layer before the softmax operation. These values preserve both the sign and magnitude of the model's output for each class, providing a richer and more continuous representation of the model's internal belief than the discretised class probabilities.

In the second stage, these temporal sequences of logits $[\ell_1, \ldots, \ell_L]$ are fed into a temporal head that learns how the class evidence evolves across frames. Two architectures were explored:

- a **1D CNN** with small convolution kernels and dilations, capturing short-range temporal patterns (e.g., periodic gait motion), and

- a **GRU** (Gated Recurrent Unit) capable of modelling longer and variable-length sequences

In practice, the 1D CNN variant was used as it trained faster, used fewer parameters, and was the same as the GRU's accuracy for this experiment. It also supports a dynamic sequence-length. Therefore, objects at longer range or far from the centre of FOV (sparser point clouds) could potentially be processed over longer windows (larger L) for robustness, while near/central objects could use shorter windows to minimize latency. A GRU is still useful when long-range temporal dependencies or streaming/variable-length inputs matter. Its recurrent state carries information across many frames, which helps at far range or with dropped/misaligned frames.

Temporal aggregation. This design explicitly decouples the learning of spatial structure (handled by the DGCNN model) from its temporal dynamics (handled by the 1D CNN), making it both modular and compute-efficient. It allows temporal smoothing and motion classification to be trained without reprocessing raw point clouds, since only low-dimensional logits are required. Furthermore, the temporal head learns to weight frames by their discriminative contribution, down-weighting ambiguous or transition frames.

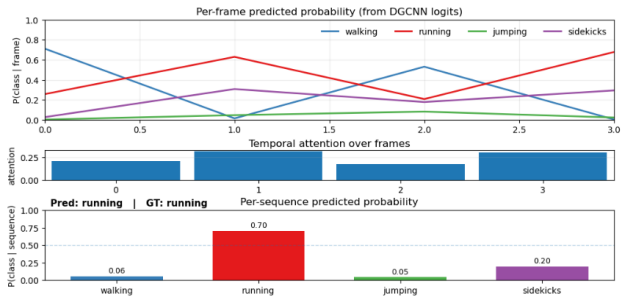


Figure 5.14: Example sequence processed by the Logits and Temporal model. Top: Per-frame class probabilities from the base DGCNN. Middle: Learned temporal attention weights showing which frames the temporal head focuses on. Bottom: Final sequence-level class probabilities after temporal model, correctly predicting `running`. This illustrates how the temporal head smooths inconsistent frame predictions while amplifying stable trends across time.

Overall, this two-stage pipeline was highly effective. It reduced computational cost while having very high accuracy.

M4 - Frame-to-Vector Temporal CNN (F2V-TCN)

This model concurrently learns spatial and temporal representations in a single end-to-end architecture, unlike M3 which depends on pre-trained per-frame logits.

Stage 1: Frame to Vector. For each ROI frame the (x, y, z) per sequence is recentred and the points are split into an 8×16 polar grid inside $R_{\max}$=1 m. For every bin, 11 statistics are calculated: counts, height and intensity means/stds, normalised radius, density, per-frame velocity (v_x, v_y), and a sinusoidal time code $(\sin \frac{2\pi t}{T}, \cos \frac{2\pi t}{T})$ where t is the frame index and T the sequence length. This embeds the frame's position within the sequence in a smooth, wrap-around way. An MLP then maps each bin's statistics into a small embedding (a learned vector summary of that bin). The bins are then max and mean pooled, and then concatenated, passing through a linear layer to get one compact 96-D frame embedding f_t; *max* keeps the strongest local response, *mean* keeps the typical level, so f_t is an order-invariant summary of the whole frame. Thus, f_t is a learned mix of the strongest per-channel signal anywhere in the frame, and the average per-channel context over the frame (from the mean).

Stage 2: Vector to Class. Let $\{f_t\}_{t=1}^{T}$ be the per-frame embeddings and $m_t \in \{0,1\}$ indicate whether frame t is valid (1) or padding (0). A lightweight 1D temporal CNN (TCN) runs along the sequence to produce features $y_t \in \mathbb{R}^{128}$ for each frame:

$$y_{1:T} \;=\; \mathrm{TCN}(f_{1:T}).$$

These features are summarized with masked pooling (ignoring padded frames):

$$y_{\max} \;=\; \max_{t:\, m_t=1} y_t, \qquad y_{\mathrm{mean}} \;=\; \frac{1}{\sum_t m_t} \sum_t m_t\, y_t.$$

In parallel, from the per-frame centre-of-mass velocity $\mathbf{v}_t$ (from the tracker), four simple, robust speed features are also used:

$$s \;=\; \Big[\; \overline{\|\mathbf{v}\|},\;\; \max_t \|\mathbf{v}_t\|,\;\; \mathrm{std}(\|\mathbf{v}\|),\;\; \big\|\textstyle\sum_t \mathbf{v}_t\, \Delta t\big\| \;\Big].$$

The classifier sees the concatenation

$$z \;=\; \big[\, y_{\max},\; y_{\mathrm{mean}},\; s \,\big],$$

which mixes short-range temporal patterns (from the TCN) with cheap, stable global kinematics (overall motion magnitude and variability). This combination is especially helpful when sequences are short or the per-frame tokens are noisy. A MLP maps z to the class logits $\boldsymbol{\ell} \in \mathbb{R}^4$.

Why M4 does not work. The shortfall for this method is the early spatial bottleneck. Turning each frame into a single 96-D vector by coarse binning and pooling removes exactly the sparse, directional limb cues that separate the distinguishing characters of the different movements. The movement inputs are COM-based, which change little during a limb movement and these limb dynamics cannot be reconstructed. In contrast, the methods that preserve this finer spatial detail (M3) provide their temporal stage already-discriminative signals; The TCN must discover them after compression, which is harder given the data size.

Switching to ST-GCNN / PSTNet. Spatio-Temporal Graph Convolutional Neural Networks (ST-GCNNs) extend frame-wise graph CNNs by constructing k-NN graphs not only within each frame (spatial structure) but also across adjacent frames (temporal continuity). This allows each layer to aggregate features from both spatial and temporal neighbours. However, this graph expansion increases computational cost roughly with $N \cdot k$ per layer, causing caomplexity and latency to grow with the number of points, neighbours, and layers.

PSTNet (Point Spatio-Temporal Network) operates directly in (x, y, z, t) space using learned spatio-temporal kernels over local space–time neighbourhoods. PSTNet-style models are typically more expressive than M4 and can be lighter than ST-GCNN.

If moving beyond M4, the number of points per frame should be reduced (e.g., to ∼24k instead of 120k) while extending the sequence length, to keep latency feasible.

5.3.5 Results

Clustering and tracking. With the expanded dataset and changed parameters, the cluster stage achieved a perfect localisation rate (Hit@0.6 m = 100%). All clusters were tracked perfectly throughout each recording.

Per-method accuracy. Table 5.8 summarises mean sequence-level accuracy (± standard deviation) across the K=5 cross-validation folds. The per-frame graph baseline (M1) had a strong performance, and using the mean probability rather than hard majority voting improved both stability and overall accuracy (from 87.4% to 88.3%). The stacked-frame PointNet (M2) collapses multiple frames into one point cloud, then global pooling averages over all these frames, thereby removing all temporality and degrading the static structure, so M2 underperforms. The two-stage Logits and Temporal model (M3) achieved near-perfect results (**99.1%±1.8**), showing that motion can be embedded effectively using the cached frame logits with a tiny 1D-CNN head. The Frame-to-Vector TCN (M4) reached 64.7%±3.7, also underperforming with this dataset.

Table 5.8: Fine-grained movement classification results. Mean sequence accuracy ± std across K=5 folds. All models use four classes (`walking`, `running`, `jumping`, `sidekicks`) and XYZI input channels.

Method	Sequence Accuracy (%)
M1 - DGCNN + Majority Vote (hard)	87.4 ± 3.4
M1 - DGCNN + Mean Probability Vote	88.3 ± 3.4
M2 - Stacked Frames + PointNet	62.2 ± 7.0
M3 - Logits+Temp (1D CNN head)	**99.1 ± 1.8**
M4 - Frame-to-Vector TCN	64.7 ± 3.7

Class-wise trends. Across folds, confusion matrices (Table 5.9) showed that `walking` and `running` were consistently recognised with perfect or near-perfect recall. Most misclassifications occurred between the `running` and `jumping` movements, which do have similar a motion. The smaller `side shuffling` class showed the highest variability, mainly due to a smaller data subset.

Table 5.9: Compact confusion matrices (rows = GT, cols = Pred). Abbrev.: W=walking, R=running, J=jumping, S=sidekicks.

	W	R	J	S
W	32	0	0	0
R	0	27	2	2
J	2	2	23	3
S	0	2	1	15

(a) Seq. agg. (maj.) acc 87.4%

	W	R	J	S
W	32	0	0	0
R	0	27	1	3
J	2	0	24	4
S	1	2	0	15

(b) Seq. agg. (mean) acc 88.3%

	W	R	J	S
W	32	0	0	0
R	0	14	1	16
J	5	4	14	7
S	0	5	4	9

(c) PointNet (seq) acc 62.2%

	W	R	J	S
W	32	0	0	0
R	0	31	0	0
J	0	0	30	0
S	0	1	0	17

(d) Temporal head acc 99.1%

	W	R	J	S
W	32	0	0	0
R	0	21	6	4
J	6	12	9	2
S	0	12	0	7

(e) F2V-TCN acc 64.7%

Complexity and latency. Table 5.10 compares latency, model size, and complexity for all methods.

Table 5.10: Model complexity and runtime comparison. Reported values are per-frame or per-sequence latency (p50/p95), FPS at p50, parameter count, approximate multiply-accumulate operations (MACs), and peak GPU memory (RTX 3050 Laptop GPU, batch size 1).

Model	p50 [ms]	p95 [ms]	FPS @ p50	Params (M)	MACs (M)	Peak VRAM [MB]
DGCNN (per-frame)	1.25	5.19	568.9	0.17	5402.6	38.8
PointNet (stacked seq)	1.35	3.14	709.7	0.18	489.9	59.2
Logits and Temporal CNN (M3)	0.9	1.17	1042.9	0.07	0.7	23.7
FV-TCN (spatio-temporal)	1.31	2.19	721.3	0.17	489.0	25.0

Summary. The two-stage **Logits and Temporal CNN** method (M3) was both the most accurate and computationally efficient method. It was advantageous for a small dataset, achieving very good latency per frame and very good generalisation across folds.

5.3.6 Discussion

With well-localised ROIs and continuous, non-fragmented tracks, incorporating temporality substantially improved the overall system accuracy. Among all the tested methods, the two-stage **Logits and Temporal CNN** achieved the best results across accuracy and computational efficiency. However, several limitations remain: (i) the small, single-subject dataset and limited range ($d \leq 35$ m) restrict generalisation; (ii) rapid motion introduces intra-frame distortion and can cause tracking errors within clusters, degrading performance in purely spatial models; and (iii) short sequences (L=4) restrict the temporal context available to the model. Intensity further improves class separability, particularly under sparse point clouds. Future extensions should explore longer sequences (L>4), greater sensing ranges (30–60 m), more erratic, multiple objects per frame that will better reflect wildlife settings.

5.4 Synthesis Across Experiments

5.4.1 Pipeline Integration

Experiment 1 established a clustering framework to isolate ROIs. *Experiment 2* extended this by refining clustering, adding tracking, and introducing a per-frame classifier, creating a foundation to add temporality. *Experiment 3* then built upon this pipeline to evaluate temporal classification methods with a more dynamic dataset, producing more accurate sequence-level predictions.

Collectively, these experiments incrementally developed each stage of the pipeline and quantified trade-offs between accuracy, robustness and latency. The next chapter further adapts the pipeline to a dataset collected at a dog park, assessing its performance under realistic, crowded and erratic conditions.

Chapter 6

Main Dataset

6.1 Aim and Scope

This chapter extends the validated pipeline from Chapter 5 to a larger and less controlled dataset collected at a public dog park. The aim is to evaluate whether the proposed cluster-track-classify pipeline can operate robustly under realistic, unstructured conditions involving multiple interacting subjects, partial occlusions, and irregular motion. This chapter outlines and explains the data-collection process, the design and implementation choices, followed by the quantitative results[1].

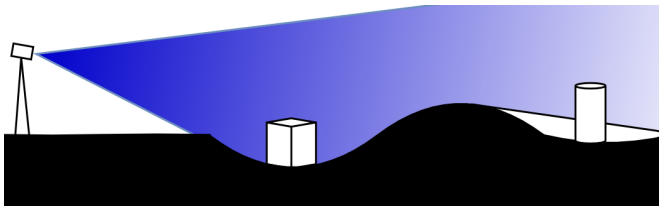
6.2 Purpose of the Main Dataset

The main dataset is designed to test the system's ability to identify and classify all moving objects, being humans and dogs, within a single sequence of frames. However, a secondary goal that really tests the pipeline is to determine whether one individual dog, named `Atlas`, which appeared across more recordings, can be distinguished from other dogs based solely on its point cloud signature. Therefore, the classification task comprised of three classes: `dog`, `human`, and `Atlas`.

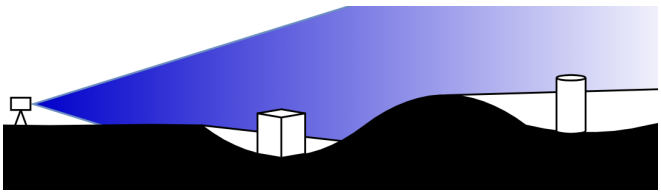
6.3 Environmental Context and Constraints

The initial plan was to collect data from several animal species at a cheetah sanctuary. However, logistical and time constraints prevented this from happening, so recordings were instead captured at a busy public dog park. Owners were asked to walk past the LiDAR with their dogs, resulting in a naturally dynamic and unpredictable environment. Dogs often ran, changed direction abruptly, interacted with one another, or moved very close to their owners, sometimes appearing as a single merged cluster. These uncontrolled behaviours provided a realistic and challenging setting that directly tested the robustness of the proposed pipeline and motivated the development of the joint clustering-tracking stage described later. This dataset could be seen as more challenging than the planned wildlife dataset in some ways, as dogs share similar proportions and gaits, unlike distinct species such as cheetahs and caracals whose differences are more pronounced. Their frequent close-range interactions and overlapping motion also makes it more challenging.

[1]Code and annotated dataset: github.com/NichTucker/LiDAR-Based-Animal-Recognition-from-Point-Cloud-Signatures. Key code blocks are in Appendix C.



(a) Recommended: elevated and slightly downward-tilted sensor covering the target zone.



(b) Used: low, parallel mounting. Wastes rays on distant background and increases occlusions.

Figure 6.2: LiDAR setup and preprocessing. (a) Recommended LiDAR setup. (b) Actual field setup that proved sub-optimal. (c) Effect of the range-bearing taper mask in BEV.

(a) Dogs present in dataset.

(b) *Atlas* (individual-of-interest) shown as its own class.

Figure 6.1: Dataset subjects and the individual Atlas, excluding the human owners. Left: dogs seen across sessions. Right: **Atlas**, which appears more often in recordings and forms its own class alongside **dog** and **human**.

6.4 Data Collection and Annotation

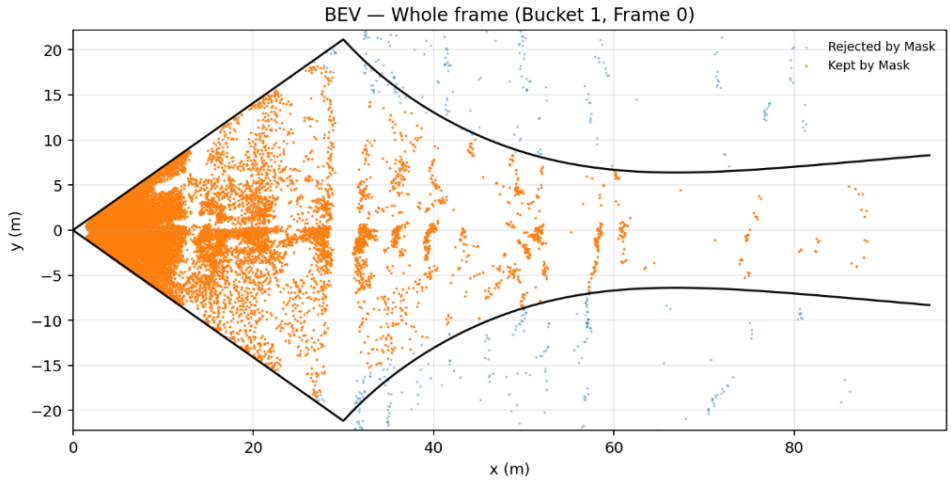
6.4.1 Acquisition Setup

Recordings were performed using a Livox Avia mounted on a low tripod approximately 0.8-1.0 m above ground level. The LiDAR was positioned with its FOV parallel to the ground. This was found to be sub-optimal.

This is because most laser rays projected outward beyond the effective field of view, 'wasting' density on far background clutter. As a consequence, subjects at medium and long ranges were under-sampled, with body parts occasionally disappearing between frames. Small ridges and bumps in the field also intermittently blocked the lower body, partially occluding parts of the subjects. Also subjects in the foreground covered the subjects behind them more frequently causing more occlusion.

To compensate for this, a range-bearing taper mask was applied during preprocessing. This mask progressively removed peripheral points at longer ranges, concentrating the sampling on the central region where subjects had a greater point cloud density (Fig. 6.2).

With this, if this dataset were to be collected again: The LiDAR should be elevated (∼4-5 m above the



(c) Range–bearing taper mask (black envelope) suppresses far-periphery points and concentrates sampling near the central activity region.

Figure 6.2: Acquisition geometry and preprocessing (cont.).

capture area) and tilted slightly downwards so that there is dense coverage of the area and occlusions are reduced. The boundary of the FOV of the LiDAR should be marked on the ground so owners know where they are in the frame and where to walk. Set up in a shaded position with a table or bench to reduce glare on the laptop screen and for ease of recording. Having someone to help out with the recording of the dataset would also make the process more efficient and manageable.

6.4.2 Dataset Overview

The final dataset comprised:

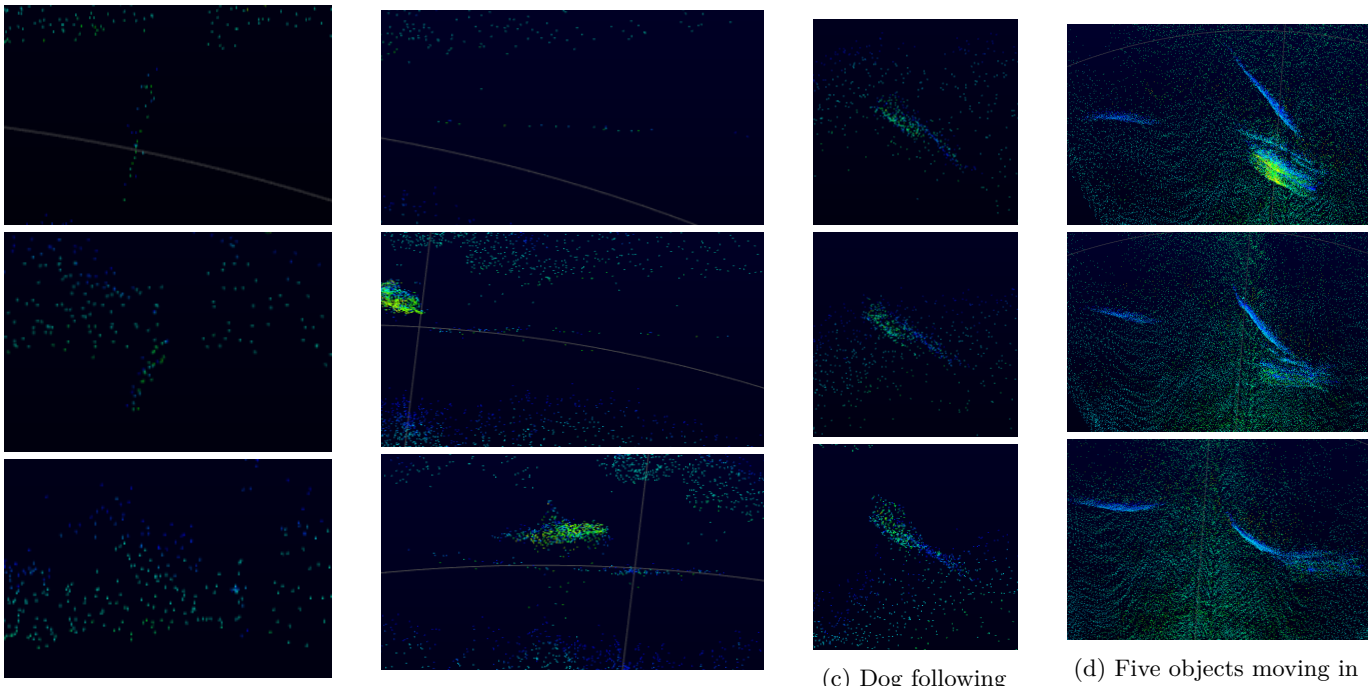
- **Frames:** 859 LiDAR frames in total;
- **Clusters:** 1 814 annotated clusters, with up to six objects visible per frame;
- **Labels:** 3-D cuboids manually annotated for three classes - `dog`, `human`, and `atlas`;
- **Annotation effort:** approximately 14 hours of manual labelling using CVAT;

This dataset offered a significantly more chaotic and realistic test environment than the controlled preliminary experiments, introducing interactions of subjects, irregular motion, and occlusion, and can be seen in Figure 6.3.

6.5 Design Constraints and Choices

Time and annotation trade-off. Given limited project time, two alternatives were considered:

1. A low point count per frame (24 k) paired with a full spatio-temporal model (e.g., PSTNet or ST-GCNN);
2. A higher point count (120 k) reusing the best-performing stages/methods from the preliminary experiments.



(a) Dog disappearing out of frame

(b) Dog running at full pace. Produced one sparse line of points.

(c) Dog following closely behind its owner. Appears as a single cluster.

(d) Five objects moving in close vicinity to each other.

Figure 6.3: Three-frame sequences illustrating frequent capture difficulties. Within each column frames are ordered top to bottom as $t-1$, t, $t+1$.

Option 1 would have increased annotation effort by roughly 5× and required creating a new classifier from scratch. Therefore, **Option 2** was selected, with the point count reduced to 72 k points per frame as a compromise.

6.6 Joint Clustering–Tracking Method

6.6.1 Motivation

The preliminary pipeline treated clustering/isolating and tracking as two separate stages. In these crowded and erratic scenes, however, this did not work. When objects came into close vicinity, ROIs merged; the objects also moved at different velocities and frequently were occluded causing fragmented ROIs; this conversely affected the tracking stage and as a result the final classification. When viewed from a bird's-eye view, each object can be visualized as a *moving density* that persists over time. This suggested using clustering and tracking in a single, mutualistic stage. Let the tracking guide the identification of the same cluster in sequential frames, and let the emergent cluster cores, in turn, update the tracks.

6.6.2 Method overview

For each frame, (i) the FOV is reduced to more informative areas with masks (Figure 6.2c), (ii) points are embedded into a discriminative feature space, and (iii) points are assigned to a small set of seeds in two passes: a tight *core* pass to anchor identities, followed by a relaxed *grow* pass to recover fringes. Seeds can be created (birth) or removed (death) mid-sequence.

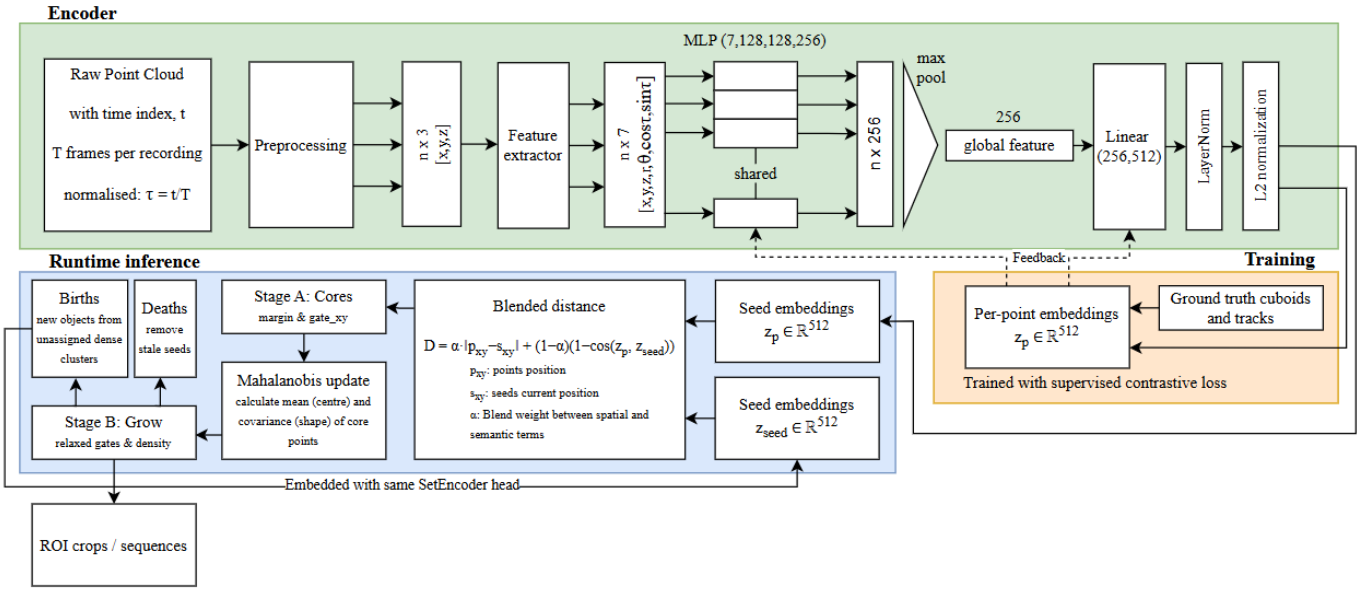


Figure 6.4: Overview of the joint clustering-tracking stage. Each frame's point cloud is encoded into a shared feature space, where seeds represent evolving object hypotheses. Core points are matched to existing seeds using spatial-semantic gating, while peripheral points are added through relaxed Mahalanobis growth. Object prototypes are updated through exponential moving averages of position and appearance, enabling stable track identities through occlusions and merges.

Figure 6.4 provides a reference for the joint clustering-tracking framework described in this section. It summarises how raw point clouds are embedded, associated, and updated over time to maintain coherent object tracks across the complex scene dynamics .

(1) Masking. A tapered range FOV mask is applied. The taper concentrates sampling density in the central activity region (Fig. 6.2c).

(2) Per-point embedding. The objective here is to provide each raw point with a short descriptor that embeds its relationship with the local geometry, its BEV position, and coarse temporal context so that points from the same object are similar, while points from different objects are dissimilar, even under occlusion and changing density.

Each 3D point is converted to a compact 7D descriptor

$$[x,\ y,\ z,\ r,\ \theta,\ \cos\tau,\ \sin\tau], \qquad r = \sqrt{x^2 + y^2},\ \ \theta = \operatorname{atan2}(y, x),$$

where $(\cos\tau, \sin\tau)$ is a continuous form of the frame index τ on the unit circle. This lets the network know roughly where it is in time (early/late in a short window) while being smooth from one frame to the next and periodic across window boundaries. Including intensity as an eighth feature would have provided valuable information about angle of incidence, likely improving separation between overlapping objects (e.g., dog vs. human contact) and aiding robustness under varying point density, with differences in angle of incidence and material.

The 7D feature vector is put through a MLP to a embedding and then ℓ_2-normalised:[2]

$$\mathbf{z} = \mathrm{MLP}(x, y, z, r, \theta, \cos\tau, \sin\tau) \in \mathbb{R}^d, \qquad \tilde{\mathbf{z}} = \frac{\mathbf{z}}{\|\mathbf{z}\|_2}.$$

This MLP is trained using a supervised contrastive objective that makes embeddings from the same object or cluster to lie close together in feature space and those from different objects to be well separated. At inference, the network is frozen and used purely as a feature extractor, producing stable, view-invariant descriptors for each point. Normalisation places all embeddings on the unit hypersphere so that cosine similarity becomes a dot product, $\cos\angle(\tilde{\mathbf{z}}_i, \tilde{\mathbf{z}}_j) = \tilde{\mathbf{z}}_i^\top \tilde{\mathbf{z}}_j$. This normalisation keeps similarity values consistent across frames and viewpoints, preventing large variations in magnitude when the same object is seen from different angles or distances. As a result, the later fusion of geometric and appearance features becomes numerically stable and easier to interpret.

(3) Seed embedding aggregation. Once per-point embeddings are obtained, they must be summarised to describe each tracked object or *seed*. A seed represents an evolving hypothesis about an object in motion, and its descriptor should therefore capture the object's position and short-term changes in shape or motion.

At birth, a seed's descriptor is initialised as the elementwise mean of all embeddings within its spatial support at the current frame:

$$\mathbf{z}_{\text{seed}}^{(0)} = \frac{1}{|\mathcal{P}_{\text{seed}}^{(0)}|} \sum_{i \in \mathcal{P}_{\text{seed}}^{(0)}} \tilde{\mathbf{z}}_i,$$

where $\mathcal{P}_{\text{seed}}^{(0)}$ is the set of points assigned to the seed during its first detection. This provides a balanced initial estimate of the object's appearance and position before any temporal smoothing.

As the object persists across frames, the seed descriptor is updated to reflect recent changes. All point embeddings associated with that seed over a short temporal window W are aggregated by elementwise max-pooling:

$$\mathbf{z}_{\text{seed}}^{(\tau)} = \max_{i \in \mathcal{P}_{\text{seed}}^{(\tau)}} \tilde{\mathbf{z}}_i, \qquad \mathcal{P}_{\text{seed}}^{(\tau)} = \bigcup_{t'=\tau-W+1}^{\tau} \{\text{points assigned to the seed at } t'\}.$$

Here, $\mathcal{P}_{\text{seed}}^{(\tau)}$ collects all points associated with the same seed over the last W frames, and each $\tilde{\mathbf{z}}_i$ encodes its local shape, BEV position, and where the point occurs within the short temporal window. The elementwise maximum preserves the strongest activations found during that period, therefore making the most distinctive geometric features (such as a dog's arched back or a human's vertical silhouette) remain represented even when intermittently visible. This makes the seed embedding both temporally stable and responsive to gradual appearance changes as the object moves.

Max-pooling is preferred over averaging because it is inherently robust to changing views. In point clouds, where only subsets of an object's surface are visible in each frame, it ensures that the seed's prototype remains

[2] ℓ_2 **normalisation:** given a nonzero vector $\mathbf{z} \in \mathbb{R}^d$, we set $\tilde{\mathbf{z}} = \mathbf{z}/\|\mathbf{z}\|_2$, where $\|\mathbf{z}\|_2 = \sqrt{\sum_j z_j^2}$. This constrains all embeddings to unit length so that cosine similarity equals the dot product, removing distortions from differences in raw magnitude.

representative even under partial views or self-occlusion. For example, when an animal briefly turns away from the sensor, the max-pooled descriptor still encodes salient features from earlier frames, allowing reliable re-identification when it reappears.

Each seed maintains two evolving estimates: its spatial state: the object's position in BEV space, and its appearance descriptor, which is its recent local geometry through its embedding. After the initial core assignment (below), both are refined jointly to improve stability and adapt to motion. The seed centre is moved toward the median position of its currently assigned core points using an exponential moving average (EMA):

$$\mathbf{s}_\tau \;\leftarrow\; (1-\lambda)\,\mathbf{s}_{\tau-1} \;+\; \lambda\,\tilde{\mathbf{p}}_{\text{core}},$$

where λ controls the update smoothness and $\tilde{\mathbf{p}}_{\text{core}}$ denotes the median of the core points' BEV positions. This smooths the track trajectory by minimizing erratic jumps from outlier points or temporary misassignments. Immediately after this positional update, the prototype $\mathbf{z}_{\text{seed}}$ is recomputed from the new core points via max-pooling. The result is a continuously evolving, noise-resistant representation that tracks both where the object is and what it looks like.

(4) Two-pass assignment (core to grow). Given a point with BEV coordinates $\mathbf{p}_{i,xy}$ (the point's (x, y) in metres), embedding $\tilde{\mathbf{z}}_i$ (its unit-length descriptor), and a seed with centre $\mathbf{s}_{k,xy}$ (the seed's current (x, y)) and prototype $\mathbf{z}_k^{\text{seed}}$ (the seed's unit-length appearance vector), their spatial state and appearance descriptor is combined with:

$$d_k(i) = \underbrace{\alpha\,\|\mathbf{p}_{i,xy} - \mathbf{s}_{k,xy}\|_2}_{\text{spatial}} + \underbrace{(1-\alpha)\,(1 - \tilde{\mathbf{z}}_i^\top \mathbf{z}_k^{\text{seed}})}_{\text{appearance}}, \qquad \alpha = \alpha(\text{range}),$$

with α decreasing mildly with range to compensate for sparser geometry at distance.

Core pass (higher precision). Each point is tentatively assigned to $\arg\min_k d_k(i)$, but accepted only if three conditions hold: (i) the point lies within a tight spatial gate around $\mathbf{s}_{k,xy}$; (ii) the mixed distance $d_k(i)$ is below a threshold; and (iii) the best seed is sufficiently better than the runner-up (margin test). Accepted core points update the seed centres (EMA).

Grow pass (higher recall). Points not accepted in the core pass are reconsidered under relaxed spatial gates, subject to two conditions: (i) a per-seed Mahalanobis ellipse in BEV (a shape-adaptive gating region derived from the core points' mean $\boldsymbol{\mu}_k$ and covariance $\mathbf{C}_k$, which accepts points whose squared distance in that metric is below a threshold), computed from the covariance of the core points. Unlike a fixed Euclidean radius, the Mahalanobis distance scales and orients the gate according to the spread and correlation of the cluster, thereby allowing elongated or tilted objects to be modelled more accurately and reducing false merges in crowded scenes.

$$\sqrt{(\mathbf{p}_{i,xy} - \boldsymbol{\mu}_k)^\top \mathbf{C}_k^{-1} (\mathbf{p}_{i,xy} - \boldsymbol{\mu}_k)} \le \tau_{\text{Mah}},$$

and (ii) a light k-NN density check to ignore isolated background points. Points that satisfy these conditions are attached to the seed. This recovers limbs/tails and sparse fringes while limiting drift across to close proximity objects.

(5) Births and deaths. Dense, compact groups among the still-unassigned points that are sufficiently far from all seed centres spawn new seeds (births). Seeds that receive no points for several consecutive frames are removed (deaths).

Why a joint stage rather than a Kalman filter? In preliminary experiments with more structured motion, a constant-velocity Kalman filter stabilised track centres well enough. In the present dataset, however, motion is strongly non-Gaussian (abrupt stops, reversals), objects frequently overlap, and their motion changes rapidly. Under these conditions, the Kalman filter can over-commit to an incorrect prediction during contact and pull a track into a neighbour, or diverge when its appearance changes (object turns around for example) but the object remains stationary.

(6) Outputs. The stage returns per-point cluster IDs and updated seed centres/IDs for the next frame. Crops around seed centres are exported as ROIs for the classifier and for evaluation.

6.6.3 Tracking and Clustering Performance

The summary in Table 6.1 reflects a first-pass configuration: due to time constraints, hyperparameters were not tuned and results with stable, minimally adjusted defaults[3] are reported.

Table 6.1: Joint clustering–tracking summary (mean±std across buckets/sequences).

Metric	Value	Notes
PointAcc	0.8373 ± 0.2295	pointwise assignment accuracy
B^3F1	0.7605 ± 0.1366	clustering consistency
Hit@0.60	0.5737 ± 0.3869	centre within 0.60 m
CentreErr (mean)	0.2492 ± 0.0618 m	matched pairs
CentreErr (median)	0.2314 ± 0.0741 m	matched pairs

6.7 Fine-grained Classification

This section reuses the **M3** approach from subsection 5.3.3 in Chapter 5 for fine-grained classification on cropped ROIs. The goal is to classify every object in a sequence of frames into one of three classes: **dog**, **human**, or the specific dog **Atlas**. (i) A per-frame point-cloud classifier is separated from (ii) a lightweight temporal head that aggregates framewise evidence over short windows. Due to time constraints, hyperparameters were not tuned; results are reported using stable, minimally adjusted defaults.[4]

[3] Cluster-Track stage: Near/far blend α_{near}=0.60, α_{far}=0.40 with split at 25 m; spatial gate ASSIGN_MAX_DIST=3.0 m with adaptive factor k=0.04; Mahalanobis threshold τ_{Mah}=2.8; short temporal window W for prototype updates; EMA smoothing for centres λ=0.30; births enabled with BIRTH_MIN_PTS=28, BIRTH_EPS_XY=0.9 m, minimum distance from existing seeds 1.2 m, density quantile 0.50; deaths enabled with DEATH_MAX_MISSES=6. Further tuning (e.g., α schedule, gating radii, density thresholds) is expected to improve performance.

[4] Per-frame graph (DGCNN): k-NN graph with k=16; batch 24; 12 epochs; Adam, η=10^{-3}, StepLR (step 6, γ=0.5); gradient clip 1.0; AMP on; per-ROI cap 2048 points (centered). Temporal head: linear projection → 1D conv (kernel 3) → tanh-attention pooling → MLP; batch 16; 25 epochs; Adam, η=10^{-3}, StepLR (step 12, γ=0.5); clip 1.0; max sequence length 64.

Temporal aggregation. For sequences, the per-frame logits are cached for each track and either: (i) the softmax over a fixed horizon $t \in \{3, 4, 5\}$ is averaged and (ii) the softmax over the frames is averaged and stopped as soon as the running argmax is stable for K=1 frame and its confidence exceeds 0.60 (capped at $t_{\max}$=8), and is known as dynamic stopping.

Figure 6.5 shows the fine-grained classifier within the broader pipeline. The diagram shows how sequences made up of three per-frame ROIs are processed.

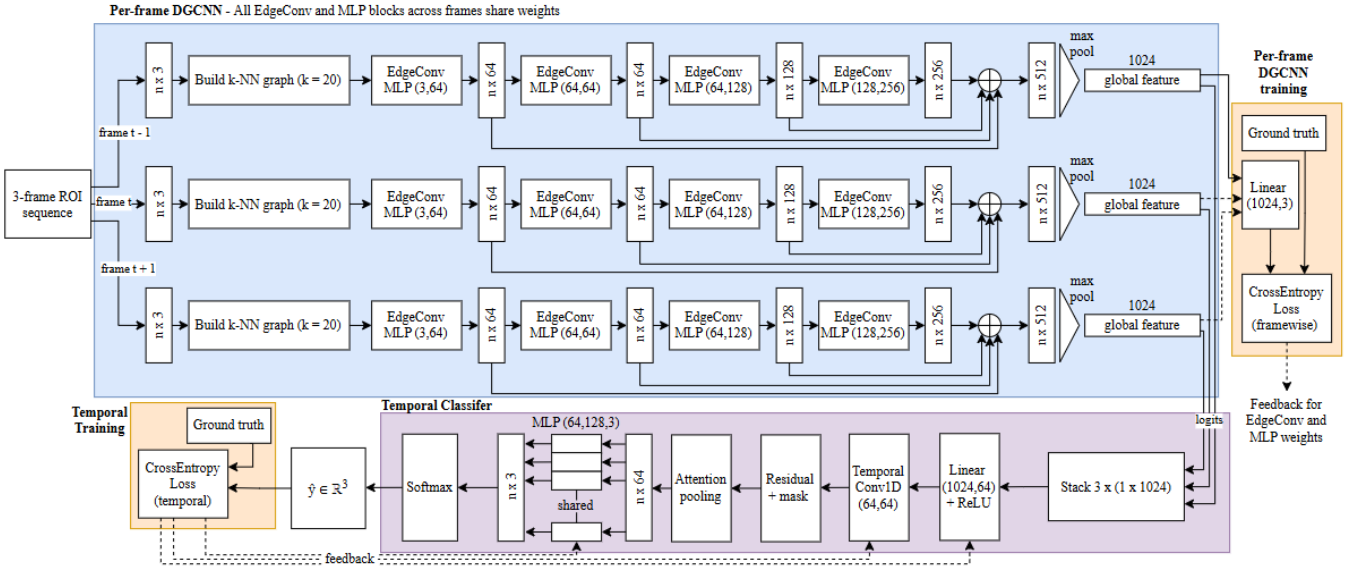


Figure 6.5: Fine-grained classification pipeline. A per-frame DGCNN produces features for each cropped ROI, sharing EdgeConv and MLP weights across three frames. The temporal head aggregates these via attention and 1D convolution to form a sequence-level prediction, trained with both frame-wise and temporal cross-entropy losses.

6.7.1 Data and splits

From the joint cluster-track stage (Sec. 6.6), crops per tracked object are exported. Within each bucket, contiguous recordings are formed; K-fold splits are made at the recording level so that tracks do not leak across folds. Two versions of the classifier are evaluated:

1. **GT crops:** upper bound using crops from the ground-truth cuboids and tracks.
2. **End-to-end:** crops produced by the joint cluster-track stage.

6.7.2 Results

Overview. Table 6.2 reports the upper bound using the perfectly cropped ground-truth (GT) ROIs, while Table 6.4 summarises the end-to-end performance where crops are outputted by the joint cluster-track stage. They also compare different sequence lengths and dynamic stopping for the GT ROIs and end-to-end ROIs. For interpretability, the performance on **Dog** vs. **Atlas** is contrasted to show the added difficulty of individual-level classification.

Table 6.2: Comparison of per-frame and sequence-level results using **ground-truth (GT) crops** across 5 folds.

Metric	**Per-frame**	t=4	t=5	t=6	**Sequence (best)**
Overall Accuracy (%)	94.8	96.4	**96.8**	96.2	96.8
Macro Precision	0.976	0.981	**0.982**	0.979	**0.982**
Macro Recall	0.983	0.986	**0.967**	0.974	0.967
Macro F1	0.980	0.984	**0.971**	0.976	0.971
Calibration (ECE / Brier)	0.042 / 0.083	–	–	–	–
Centre Accuracy (%)	98.6	99.1	**99.3**	98.7	99.3
Off-centre Accuracy (%)	90.7	94.5	95.8	**96.0**	96.0

Per-class performance (Precision / Recall / F1):

Class	**Per-frame**	t=4	t=5	t=6
Dog	0.970 / 0.970 / 0.970	0.972 / 0.986 / 0.979	**0.959 / 0.987 / 0.973**	0.961 / 0.982 / 0.972
Human	0.990 / 0.990 / 0.990	**1.000 / 1.000 / 1.000**	**1.000 / 1.000 / 1.000**	0.995 / 0.995 / 0.995
Atlas	0.960 / 0.960 / 0.960	0.940 / 0.840 / 0.887	**0.936 / 0.815 / 0.871**	0.921 / 0.832 / 0.874

Table 6.3: Compact confusion matrices (rows = GT, cols = Pred) for fine-grained classification (on GT ROI crops) at different temporal horizons. Abbrev.: D = Dog, H = Human, A = Atlas.

	D	H	A
D	26	0	0
H	0	126	0
A	39	1	29

(a) Per-frame Acc = 81.9%

	D	H	A
D	13	0	0
H	0	45	0
A	15	0	7

(b) t=3 frames Acc = 81.3%

	D	H	A
D	12	0	0
H	0	42	0
A	12	0	6

(c) t=4 frames Acc = 83.3%

	D	H	A
D	10	0	0
H	0	30	0
A	10	0	5

(d) t=5 frames Acc = 81.8%

	D	H	A
D	14	0	0
H	0	47	0
A	12	0	16

(e) Dynamic Acc = 87.5%

Table 6.4: Comparison of per-frame and sequence-level fine-grained classification results on validation end-to-end ROI crops (5-fold mean).

Metric	**Per-frame**	t=3	t=4	t=5	**Dynamic**	**Notes**
Overall Accuracy (%)	81.9	81.3	83.3	81.8	**87.5**	micro average (VAL)
Macro Precision	0.797	**0.889**	0.833	0.833	0.833	mean over 3 classes
Macro Recall	0.807	**0.833**	0.778	0.778	**0.833**	
Macro F1	0.720	**0.822**	0.722	0.722	0.778	
Calibration (ECE / Brier)	0.131 / 0.304	–	–	–	–	lower is better
Centre Accuracy (%)	87.9	100.0	87.5	87.5	100.0	$\lvert\theta\rvert \leq 7^\circ$
Off-centre Accuracy (%)	64.3	57.1	75.0	66.7	57.1	
Mean Frames to Decision	–	–	–	–	3.7	dynamic stop (τ=0.6, K=1)

Per-class performance (Precision / Recall / F1):

Class	**Per-frame**	t=3	t=4	t=5	**Dynamic**
Dog	0.400 / **1.000** / 0.571	**0.667** / **1.000** / **0.800**	0.500 / **1.000** / 0.667	0.500 / **1.000** / 0.667	0.500 / **1.000** / 0.667
Human	0.992 / **1.000** / 0.996	**1.000** / **1.000** / **1.000**	**1.000** / **1.000** / **1.000**	**1.000** / **1.000** / **1.000**	**1.000** / **1.000** / **1.000**
Atlas	**1.000** / 0.420 / 0.592	**1.000** / 0.500 / **0.667**	**1.000** / 0.333 / 0.500	**1.000** / 0.333 / 0.500	**1.000** / **0.500** / **0.667**

Table 6.5: Compact confusion matrices (rows = GT, cols = Pred) for fine-grained classification (on end-to-end ROI crops) at different temporal horizons. Abbrev.: D = Dog, H = Human, A = Atlas.

	D	H	A
D	26	0	0
H	0	126	0
A	39	1	29

(a) Per-frame Acc = 81.9%

	D	H	A
D	13	0	0
H	0	45	0
A	15	0	7

(b) t=3 frames Acc = 81.3%

	D	H	A
D	12	0	0
H	0	42	0
A	12	0	6

(c) t=4 frames Acc = 83.3%

	D	H	A
D	10	0	0
H	0	30	0
A	10	0	5

(d) t=5 frames Acc = 81.8%

	D	H	A
D	14	0	0
H	0	47	0
A	12	0	16

(e) Dynamic Acc = 87.5%

6.7.3 Latency and Computational Complexity

Median end-to-end per-frame latency is **2.45 s** (p50), corresponding to **∼0.41 FPS**.

Table 6.6: End-to-end latency overview.

	p50 (s)	p90 (s)	p95 (s)	FPS @ p50
End-to-end per frame	2.45	3.53	3.92	0.41

Table 6.6 shows the total time needed to process one LiDAR frame through the full pipeline. The values are the median (**p50**), 90th percentile (**p90**), and 95th percentile (**p95**) latencies,n showing both typical and worst-case performance. The final column converts the median latency to an equivalent frame per second of around 0.4. This table provides a top-level measure of how quickly the full system can produce a result.

Stage shares (p50). To understand which parts of the pipeline contribute most to total processing time, the median per-frame latency (2.45 s) was decomposed into individual stages, as summarised in Table 6.7. Each value represents the average proportion of runtime spent in that stage during normal operation. The assignment stage, which is responsible for comparing all active points to existing object seeds, dominates computation. Births corresponds to the initialisation of new tracks whenever unassigned point clusters are detected and verified and embedding covers the forward pass of the per-point encoder that generates the geometric and spatial descriptors used by both tracking and classification. Classification adds only a small cost since it operates on the small cropped ROIs extracted after clustering. 'Other' accounts for operations like file input/output, CPU/GPU memory transfers, logging, and operations that keeps frame counters and temporary buffers in sync.

Table 6.7: Per-stage median latency and share at p50.

Stage	Median (ms)	Share (%)
Assignment	1385	56.5
Births	533	21.8
Embedding	186	7.6
Classification	82	3.4
Other	264	10.8

Complexity. To show how runtime scales with the complexity of each scene, Table 6.8 the Pearson correlation

coefficients (r) between total per-frame latency and several variables that approximate computational load is calculated. These measurements were recorded to confirm whether the system's runtime behaves as theoretically expected and to identify which factors most strongly influence processing cost. N_{masked} is the number of active LiDAR points remaining after masking. K_{seeds} is the number of tracked objects currently active in the scene. The other two terms, $N \log N$ and $N{\cdot}K$, were calculated from these values for every frame to approximate the complexity of the two dominant computational patterns in the pipeline: the $N \log N$ term is the neighbourhood or tree-based operations (e.g., cKDTree density queries) that scale slightly faster than linearly with the number of points, while the $N{\cdot}K$ term represents dense point-seed comparisons during assignment, whose cost grows with both the number of points and tracked objects.

High correlation values ($r \approx 0.87$) for both N_{masked} and the $N \log N$ proxy means that frame time increases almost proportionally with the number of active points and the associated neighbourhood-search computations. In contrast, the weak correlations ($r \approx 0.16$) for K_{seeds} and $N{\cdot}K$ show that the number of tracked objects adds less computation, confirming that per-point computations dominate runtime rather than per-track updates.

Table 6.8: Correlations with total per-frame time.

Variable	Pearson r
Effective points (N_{masked})	0.872
Seeds (K_{seeds})	0.159
Proxy $N \log N$	0.872
Proxy $N{\cdot}K$	0.160

Overall, these results highlight that optimisation efforts should prioritise reducing point density and improving the efficiency of spatial search operations.

Streaming time-to-decision. Let Δt be inter-frame spacing and L_{frame} the median per-frame latency (2.45 s).

- Fixed horizon t=3 $L_{\text{total}} \approx L_{\text{frame}} + (t-1)\Delta t = 2.45 + 0.60 \approx \mathbf{3.05\,s}$ for Δt=0.3 s.
- Dynamic stopping (mean 3.7 frames): $L_{\text{total}} \approx 2.45 + 1.11 \approx \mathbf{3.56\,s}$.

This estimates how long it takes the system to produce a decision once a target appears in view. With a 3-frame horizon and 0.3 s frame spacing, the first decision arrives in roughly 3.0-3.5 s, and is the effective response delay for near-real-time use.

Summary. This section provides a concise quantitative view of system performance. These results form the basis for the next chapter, which analyses them and their implications, identifies performance bottlenecks, and discusses what improvements could further improve the system's accuracy and real-time feasibility.

Chapter 7

Discussion and Evaluation of Results

7.1 Chapter Purpose

This chapter critically evaluates the results reported in Chapter 6. How well the system meets the stated objective of species and individual classification, and its real-time feasibility, is accessed. Successful and failure cases to specific technical and practical factors are identified and explained; factors that threaten the validity of the results are dicussed, and recommendations with the highest expected return are given.

7.2 Evaluation on the system objectives

Recognition quality. With the perfect GT ROI crops, the classifier achieves **96-97%** accuracy (Table 6.2), demonstrating that 3D shape and motion alone are sufficient for distinguishing the three classes (`dog`, `human`, `atlas`). End-to-end accuracy with the outputted crops is **82-88%** (Table 6.4), showing that the pipeline is robust under realistic, complex dynamics, but exposes a gap relative to the upper bound. Although the system reliably separates humans from dogs, its precision drops between the two dog classes (`dog` vs. `atlas`) as dogs have similar proportions and gait. With the end-to-end pipeline, most `atlas`/`dog` errors arise from errors introduced during the cluster–track stage rather than true visual or behavioural similarity between classes. The classifier can clearly distinguish Atlas when provided with perfect GT crops, but these similarities become problematic once exported sequences and crops are imperfect or inconsistent. Small localisation errors, like off-centre ROIs or fragmented tracks, reduce the discriminative information available to the classifier, making two distinct dogs appear more alike in the feature space. Accordingly, the observed gap is explained by (i) partial or off-centre crops of Atlas caused by seed drift and edge-of-FOV sparsity, (ii) short or fragmented tracks where premature deaths and rebirths reset the seed prototype, yielding inconsistent ROIs across frames, (iii) contamination during the grow pass that passes background points or ignores true ROI points, and weakens the Atlas descriptor, and (iv) class imbalance (fewer `atlas` sequences), which biases uncertain predictions toward the generic `dog` class.

Identity continuity. The joint clustering–tracking stage has a B^3F1 of ≈ 0.76 and median centre error of $\sim$0.23 m (Table 6.1), which is sufficient for producing stable crops; however, residual merges and splits still propagate errors downstream, occasionally degrading the input quality of the classifier.

Latency target. In this pipeline, each classification decision is based on a short temporal sequence of three consecutive frames, as this window size gave the most consistent performance. But in theory, longer sequences would allow the model to aggregate more temporal context, but the combination of a limited dataset and

occasional cluster–track errors made three frames the most reliable choice. Extending the horizon (e.g., t=5 or t=6) provided diminishing returns due to missing or fragmented tracks, and less data to train on. Thus, t=3 was a pragmatic balance. It has sufficient temporal context for stable classification and keeps latency low. Median end-to-end per-frame time is **2.45 s** (Table 6.6), with an estimated streaming time-to-decision of **∼3.05 s** for t=3 or **∼3.56 s** with dynamic stopping (Sec. 6.7.3). This system is near-real-time, but remains too slow for applications requiring immediate feedback.

The core goal is thus achieved in principle (upper bound) and remains usable end-to-end, but performance is currently limited by the ability to isolate and consistently track each object across frames, rather than by the discriminative strength of the classifier itself.

7.3 The gap between GT cuboids and the full end-to-end pipeline

The GT and end-to-end gap can be attributed to three factors, each supported by evidence in Chapter 6:

A. Crowding and overlap degrade association

Confusion matrices show that `Atlas` was frequently classified as `Dog`, while `Dog` recall remained high (Tables 6.5, 6.4). This asymmetry arises from two main causes. First, the blended distance metric $D = \alpha\, d_{xy} + (1-\alpha)(1-\cos\theta)$ is insufficiently selective during close interactions or overlaps; without intensity, seeds occasionally over-merge, producing merged ROI crops that bias classification. Second, the dataset contains far fewer `Atlas` sequences than generic dogs, so the classifier is underexposed to that class and more prone to defaulting to `dog` under uncertain or incomplete crops. The failure exemplars in Fig. 7.2 confirm this, while the sustained accuracy on GT crops indicates that the classifier itself is not the bottleneck but rather the localisation and class imbalance preceding it.

B. LiDAR setup

Tracks near the edges of the field of view were often unstable, with some trajectories lasting fewer than three frames before disappearing. The Livox Avia's low, parallel (with the ground) FOV combined with its non-repetitive scanning pattern produces uneven spatial coverage: point density drops sharply toward the FOV edges, while foreground subjects and bumps in the terrain cause frequent occlusions (Figure 6.2b). Objects are thus often only partially visible, resulting in incomplete point clouds and fragmented tracks. This can be seen in Fig. 6.3 and by the larger gap between GT and end-to-end performance for off-centre cases in Tables 6.2 and 6.4.

C. Computation concentrates in data association

Profiling shows that the Assignment and Births stages together account for approximately 78% of total per-frame runtime, whereas Classification makes up only about 3.4% (Table 6.7). This imbalance arises because dense point–seed comparisons and repeated neighbourhood searches that dominate computation, both of which scale with the number of effective points in the frame. As shown in Table 6.8, total frame time correlates strongly with N_{masked} and with the analytical $N \log N$ proxy ($r \approx 0.87$), but only weakly with

the number of active seeds K_{seeds} ($r \approx 0.16$). These correlations show that runtime is driven primarily by per-point operations rather than by the number of tracked objects.

7.4 Qualitative inspection of clustering success and failure

The clustering–tracking stage establishes the entire pipeline, since all subsequent classification depends on the quality and continuity of these ROIs. To better understand the measured results, qualitative inspection was performed across several representative sequences. Figure 7.1 shows a typical successful cluster–track, while Figure 7.2 depicts afailure case. Each ROI is overlaid with its ground-truth cuboid.

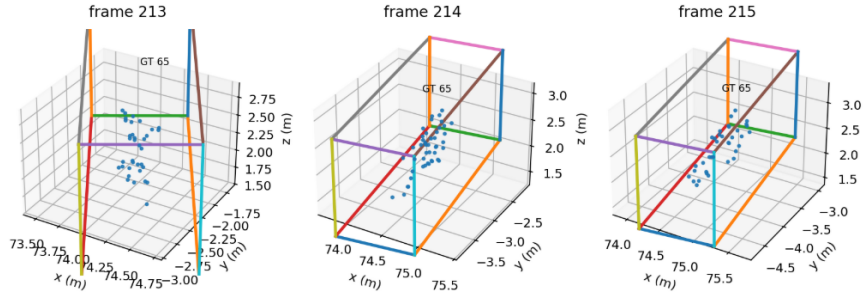


Figure 7.1: Example of a successful cluster track of a dog. The centroid remains stable and continuous across frames, showing clear spatial separation from the background and smooth motion between successive positions. Such sequences produce high-quality ROIs for fine-grained classification.

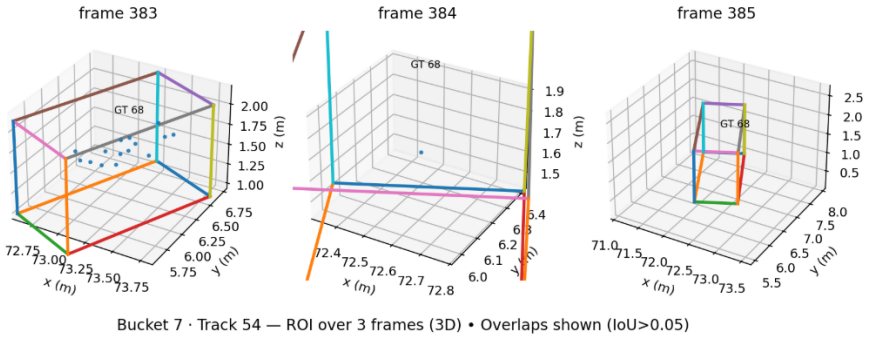


Figure 7.2: Example of an unsuccessful cluster track. The dog becomes partially occluded as LiDAR rays are blocked by another subject, while the remaining visible region lies in a low-density sector of the scan. The resulting cluster fragments, leads to a short, unstable track that limits temporal aggregation and increases classification variance.

When point density is sufficient and separation from other objects is clear at least once within the sequence, the cluster–track pipeline maintains stable identities and accurate centroids. Brief or partial occlusions are often recoverable, as long as the object reappears within a few frames and enough core points are re-established to realign the seed descriptor. However, when point density drops or occlusion persists, the blended distance metric loses discriminative power, leading to cluster merges or premature track terminations. These observations further cement that the limitation of the LiDAR's capabilities and the way it was setup

for the dataset causes inherent instability in the algorithm itself, however the accuracy on the GT cuboids proves that it is recoverable.

7.5 Validity of results

Internal validity. All metrics were computed using recording-level splits to prevent any leakage across folds (Sec. 6.7), so that training and validation samples remained fully independent. The upper-bound experiment further confirms that the classifier's capacity is independent of localisation quality and does not rely on environmental context or scene leakage for recognition. This validates the architectural choice to separate localisation from classification and demonstrates that the model learns the objects point cloud signature rather than using background leakage, making it more transferable to new environments. However, hyperparameters within the pipeline were not experimented with. These values were set empirically during development, meaning that the reported performance represents one reasonable configuration rather than an optimised one. Accordingly, the observed trends can be considered reliable, but the absolute metrics should be viewed as conservative estimates until a more exhaustive hyperparameter study is done.

Construct validity. The dog-park dataset served as an intermediary for the intended wildlife recordings. In some respects, it presents an even more challenging test: multiple dogs moving erratically around each other with similar body proportions and gait, making the intra-species discrimination between **dog** and **atlas** potentially more difficult than inter-species separation such as cheetah versus caracal. Nevertheless, the owners (`human`) were easily distinguishable, providing a useful reference class. Overall, the experiment fully tests whether point cloud signatures are sufficient to distinguish morphologically and behaviourally similar classes, even if the setting differs from the intended one.

External validity. The limited dataset (859 frames, ∼1.8k clusters) and small amount of classes does constrain the generalisability of these results to broader ecological settings. Because intensity was not included for both clustering and classification, conclusions, particularly when there are objects in close proximity or occlusion, should be regarded as a lower bound on achievable performance. Therefore, a larger, more diverse dataset with intensity, added as a feature, would be necessary to confirm the robustness of these findings across species, environments, LiDARs, and set-ups.

7.6 Improvements

Recommendations are ranked by the expected impact on the end-to-end gap and latency, grounded in the evidence above.

1. **Integrate intensity end-to-end.** Add intensity to the per-point feature channels for both clustering–tracking and classification. This addresses **A** (section 7.3) by restoring surface and material separability during crowding and occlusion. For example, Atlas has a bright, reflective white coat, whereas other dogs of similar size have darker coats, thus with less reflectance. This would also allow the system to distinguish overlapping subjects (because of the difference in angle of incidence). Imple-

mentation complexity is low, requiring only minor network input modifications, but offers substantial gain.

2. **Reduce effective points before association.** One method could be to use a one-time ground-plane calibration scan of the empty environment before recording, then subtract or mask that static background during live operation to remove most/all background points, and therefore addresses **C** (section 7.3). A light voxel or grid pre-filter (e.g., 5–10 cm resolution) can homogenise density and discard background points. Together, these steps directly reduce computational load, since point–seed comparisons scale with $N \log N$. Preprocessing could reduce per-frame latency by up to 30–40% with negligible loss of accuracy.

3. **Set-up of LiDAR.** Elevate the LiDAR by approximately 4–5 m and tilt it downward by 10–15°. This will increase coverage uniformity and effectiveness, and reduce occlusion from objects passing in front of each other and uneven terrain, addressing **B** (section 7.3). Mark zones on the ground (e.g., with cones or tape) so that subjects (humans) know where to remain within the field of view.

4. **Association efficiency.** The data association stage can be made faster by reducing the number of distance checks between points and seeds. A coarse-to-fine approach, like using voxel grids, can limit comparisons to only nearby regions instead of evaluating every point against every seed. Performing distance and Mahalanobis updates in parallel on the GPU, rather than sequentially in Python loops, would also further reduce computation time. Finally, limiting how often birth tests are run in sparse areas would prevent redundant density checks. Together, these optimisations directly target the main runtime bottleneck shown in Table 6.7, which currently accounts for roughly 56–78% of total processing time.

5. **Hyperparameter refinement.** Several parameters, particularly the spatial–semantic blending weight α, Mahalanobis gating threshold, and birth/death persistence limits, were not changed. Although α currently varies with range to compensate for sparser geometry at distance, it remains constant across bearing angles, which may underweight off-axis points where sampling anisotropy is highest. Tuning of all these and other parameters through grid or Bayesian optimisation would improve localisation, without requiring structural changes to the model.

7.7 Real time feasibility

The time taken under this setup is 3.0-3.6 s for t=3 or dynamic stopping with Δt=0.3 s. This does suit ecological monitoring. Improvements must primarily target association and point count reduction to reduce latency.

Chapter 8

Conclusions

8.1 Summary of Work

This project set out to determine whether wildlife species, such as cheetahs, caracals, and bat-eared foxes, can be identified solely from their point cloud signatures recorded using the Livox Avia. Due to logistical and time constraints, the dataset was collected at a public park where the interactions and behaviour of dogs and their owners were used as a proxy for testing the system under realistic, dynamic conditions.

Chapter 1 established the motivation for LiDAR-based ecological monitoring, while Chapter 2 reviewed current nocturnal animal recognition methods, and LiDAR applications and its machine learning approaches, identifying key gaps with it in wildlife classification. Chapter 3 described the hardware and software used throughout the course of the thesis, and Chapter 4 covered the theoretical framework and introduced the cluster–track–classify pipeline designed to keep an object's identity across frames for fine-grained classification. Preliminary experiments in Chapter 5 developed and validated each stage of the pipeline, and Chapter 6 integrated them into a full end-to-end system. Finally, Chapter 7 critically evaluated performance, identifying the main sources of error and outlining strategies for improvement.

Across all experiments, the classifier achieved **96–97%** accuracy on ground-truth crops and **82–88%** on the end-to-end crops, showing that point cloud signatures are discriminative enough to separate classes effectively. Processing the point cloud signatures over three to five frames improved prediction stability and achieved near-real-time feasibility, with a latency of about 3–4 s , suitable for ecological monitoring.

8.2 Key Findings

The main findings are:

- A unified clustering–tracking framework that maintains object identity through occlusion and partial merges.
- A fine-grained DGCNN-based temporal classifier that distinguishes morphologically similar classes even under sparse and noisy sampling.
- A latency and complexity analysis showing that most computational cost occurs in the data association stage rather than classification.
- Validation that point cloud signatures are sufficient for species classification without the reliance on RGB imagery or other sensor fusion.

8.3 Limitations

Performance was constrained primarily by hardware, dataset scale, and time available for experimentation:

- The Livox Avia was the only LiDAR available, and its low, parallel FOV during data collection accentuated its off-centre sparsity and caused more occlusion.
- The dataset was relatively small (859 frames, ∼1.8k clusters) with limited class diversity (only `dog`, `atlas`, and `human`) and an underrepresented `atlas` class, which further restricted generalisation.
- Time constraints prevented extensive hyperparameter tuning and ablation studies, so reported performance reflects a single stable configuration rather than an optimised one.

8.4 Contributions and Impact

This thesis contributes a reproducible, modular framework that unifies clustering, tracking and temporal classification for LiDAR-based animal recognition. It demonstrates that affordable LiDAR sensors can capture meaningful structural and motion information without colour imagery. Even independently of classification, the generated sequences of ROI clusters for each animal provide ecologists with valuable spatio-temporal data to analyse an animal's health, movement patterns, social interactions, and habitat use. By extending LiDAR's application from terrain mapping to animal classification, this work offers a foundation for scalable, non-intrusive wildlife monitoring systems and transferable insights for related domains such as robotics and autonomous driving.

8.5 Future Outlook

Future research should broaden the dataset to include more species, longer behavioural sequences, and a greater diversity of environments. The LiDAR should be elevated and angled downward to provide more uniform coverage and reduce occlusion. Progress in both hardware and algorithmic design, from experimenting with different types of LiDAR and scanning patterns to optimizing the pipeline, will further enhance robustness, increase accuracy and computational efficiency. In the longer term, an automated LiDAR-trap could enable continuous, non-intrusive way of capturing data of wildlife in their natural habitats. Together, these developments help move this framework toward a real-time, autonomous ecological monitoring system capable of classifying nocturnal species across complex natural landscapes.

8.6 Final Perspective

This work shows that LiDAR can reveal not only the structure of landscapes but directly the species that inhabit them. It leads to a broader class of perception systems capable of interpreting dynamic environments, both natural and engineered, through spatial structure and motion rather than two-dimensional appearance.

Bibliography

[1] C. R. Qi, H. Su, K. Mo, and L. 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 (CVPR)*, 2017, pp. 652–660.

[2] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4490–4499.

[3] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," in *Sensors*, vol. 18, no. 10. MDPI, 2018, p. 3337.

[4] F. n. Qiu *et al.*, "Gfnet: Geometric flow network for 3d point cloud semantic segmentation," in *[Conference]*, 2023.

[5] M. S. Norouzzadeh, A. Nguyen, M. Kosmala, A. Swanson, M. S. Palmer, C. Packer, and J. Clune, "Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning," *Proceedings of the National Academy of Sciences*, vol. 115, no. 25, pp. 5716–5725, 2018.

[6] S. Schneider, G. W. Taylor, S. Linquist, and S. C. Kremer, "Deep learning object detection methods for ecological camera trap data," *Ecology and Evolution*, vol. 8, no. 13, pp. 12 823–12 835, 2018.

[7] W. Lyu, C. Liu, X. Liu, Y. Luo, and Y. He, "Nighttime wildlife object detection based on yolov8-night," *Sensors*, vol. 24, no. 2, p. 381, 2024.

[8] A. Krishnan, S. Pant, M. Meijer, J. A. Eikelboom, S. A. Wich, G. Olthuis, and L. P. Koh, "Fusion of visible and thermal images improves automated detection and classification of animals for drone surveys," *Scientific Reports*, vol. 13, no. 1, p. 11655, 2023.

[9] L. B. Kekule. Bat-eared fox photographed with a white-flash camera trap. Photo: L. Bruce Kekule. [Online]. Available: https://brucekekule.com/tag/bat-eared-fox/

[10] M. Chynoweth and Q. F. Team. (2023) Infrared camera trap image capturing the first cheetah sighted in djibouti in over three decades. Quinney College of Natural Resources (QCNR), Utah State University. Photo credit: Mark Chynoweth, QCNR camera-trap survey, Djibouti. [Online]. Available: https://qcnr.usu.edu/news/cheetah-djibouti-camera-trap

[11] NatureSpy. Seeing in the dark: thermal imaging cameras are transforming nocturnal wildlife surveys. Still image of a hedgehog from article. [Online]. Available: https://naturespy.org/blogs/journal/seeing-in-the-dark-thermal-imaging-cameras-are-transforming-nocturnal-wildlife-surveys

[12] A. B. Davies and G. P. Asner, "Advances in animal ecology from 3d-lidar ecosystem mapping," *Trends in Ecology & Evolution*, vol. 29, no. 12, pp. 681–691, 2014.

[13] W. D. Simonson, H. D. Allen, and D. A. Coomes, "Applications of airborne lidar for the assessment of animal species diversity," *Methods in Ecology and Evolution*, vol. 5, no. 8, pp. 719–729, 2014.

[14] D. J. McNeil, D. A. Buehler, T. R. Evans, E. Luttermoser, and L. P. Bulluck, "Using aerial lidar to assess regional availability of potential habitat for a conservation-dependent forest bird," *Forest Ecology and Management*, vol. 541, p. 121022, 2023.

[15] S. Ciuti, L. Pipia, I. Gherghel, S. Davini, M. Apollonio, L. Pedrotti, and F. Cagnacci, "An efficient method to exploit lidar data in animal ecology," *Methods in Ecology and Evolution*, vol. 8, no. 9, pp. 1276–1285, 2017.

[16] D. Gartner, M. Merrick, A. Sanchez-Azofeifa, and S. E. Franklin, "Into the third dimension: Benefits of incorporating lidar data in wildlife habitat models," in *Proceedings of the 2013 Forest Service Remote Sensing Applications Conference*. USDA Forest Service, 2013, pp. 49–56.

[17] J. Yang, Y. Wang, D. Chou, C. Kim, D. Wang, D. Foster, and Y. Yang, "Synchronizing 3d lidar-based reconstruction of animals and environment for in situ ecosystem characterization," *Ecological Informatics*, vol. 80, p. 102393, 2025.

[18] V. Haucke and V. Steinhage, "Exploiting depth information for wildlife monitoring," *arXiv preprint arXiv:2102.05607*, 2021.

[19] Y. Li, L. Ma, Z. Zhong, F. Liu, D. Cao, J. Li, and M. A. Chapman, "Deep learning for lidar point clouds in autonomous driving: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 9, pp. 1–19, 2020.

[20] H. Zhao, Y. Guo, Y. Wang, G. Wang, and H. Liu, "A comprehensive overview of deep learning techniques for 3d point cloud classification and semantic segmentation," *Pattern Recognition*, vol. 107, p. 107446, 2021.

[21] D. Griffiths and J. Boehm, "A review on deep learning techniques for 3d sensed data classification," *arXiv preprint arXiv:1907.04444*, 2019.

[22] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

[23] T. Liu, F. Zhang, X. Yang, W. Xu, Y. Wang, H. Chen, and Y. Liu, "Pcb-randnet: Rethinking random sampling for lidar semantic segmentation in autonomous driving scene," *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[24] Z. Jing, H. Guan, P. Zhao, D. Li, Y. Yu, Y. Zang, H. Wang, and J. Li, "Multispectral lidar point cloud classification using se-pointnet++," *Remote Sensing*, vol. 13, no. 13, p. 2516, 2021.

[25] Y. Chen, G. Liu, Y. Xu, P. Pan, and Y. Xing, "Pointnet++ network architecture with individual point level and global features on centroid for als point cloud classification," *Remote Sensing*, vol. 13, no. 3, p. 472, 2021.

[26] I. Lang, A. Manor, and R. Giryes, "Interpolated convolutional networks for 3d point cloud understanding," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 1324–1334.

[27] X. He, Y. Wu, D. Zhang, Y. Xu, X. Xu, Y. Zhou, and Y. Zhao, "Restreenet: A height-aware lidar tree classification model with explainable ai for forestry applications," *Remote Sensing*, vol. 15, no. 15, p. 3755, 2023.

[28] W. Li, Q. Guo, M. K. Jakubowski, and M. Kelly, "Tree species classification using ground-based lidar data by various point cloud deep learning methods," *Remote Sensing*, vol. 13, no. 6, p. 1154, 2021.

[29] H. Kuang, Y. He, Y. Li, J. Liu, W. Ma, and J. Yang, "Voxel-fpn: Multi-scale voxel feature aggregation for 3d object detection from lidar point clouds," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 3460–3466.

[30] J. Shi and P. Wonka, "Voxelkp: A voxel-based network architecture for human keypoint estimation in lidar data," *arXiv preprint arXiv:2303.09962*, 2023.

[31] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 12 697–12 705.

[32] S. Zhang, S. Zhou, Q. Wu, and L. Shen, "Yolo4d: A spatio-temporal approach for real-time multi-object detection and classification from lidar point clouds," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 2022, pp. 1825–1834.

[33] Y. Li, Z. Ou, G. Liu, Z. Yang, Y. Chen, R. Shang, and L. Jiao, "Three-dimensional point cloud object detection based on feature fusion and enhancement," *Remote Sensing*, vol. 16, no. 6, p. 1045, 2024.

[34] F. n. Xu, "Rethinking range view representation for lidar segmentation," *[Journal/Conference]*, 2023.

[35] D. Wu, Z. Liang, and G. Chen, "Deep learning for lidar-only and lidar-fusion 3d perception: a survey," *Intelligence & Robotics*, vol. 2, no. 2, pp. 105–129, 2022.

[36] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," in *ACM Transactions on Graphics (TOG)*, vol. 38, no. 5, 2019, pp. 1–12.

[37] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas, "Kpconv: Flexible and deformable convolution for point clouds," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 6411–6420.

[38] A. Camuffo, A. Cenedese, and L. Ballan, "Graph neural networks for intelligent transportation systems: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 9, pp. 14 254–14 273, 2022.

[39] F. Drobnitzky and D. Cremers, "Recent advancements in learning algorithms for point clouds: An updated overview," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. V-2-2022, pp. 73–80, 2022.

[40] H. Fan, X. Yu, Y. Ding, Y. Yang, and M. Kankanhalli, "Pstnet: Point spatio-temporal convolution on point cloud sequences," in *International Conference on Learning Representations (ICLR)*, 2021. [Online]. Available: https://openreview.net/forum?id=Fq2F7gGJqAw

[41] Z. Wang, J. Zhang, Y. Yang, and Z. Zhu, "Pointmotionnet: Point-wise motion learning for large-scale lidar point clouds sequences," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2022, pp. 1825–1834.

[42] A. El Sallab *et al.*, "Yolo4d: From perception to prediction for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018.

[43] Y. Li *et al.*, "Streammos: Streaming moving object segmentation with multi-view perception and dual-span memory," *IEEE Transactions on Intelligent Vehicles*, 2023.

[44] Z. Feng *et al.*, "Spatio-temporal bi-directional cross-frame memory for distractor filtering in point cloud single object tracking," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

[45] B. Yang, R. Wang, W. Wang, G. Li, T. Chen, C. Wen, and J. Li, "A survey on deep-learning-based lidar 3d object detection for autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 8, pp. 8447–8468, 2023.

Appendix A

Multilayer Perceptron (MLP)

A **multilayer perceptron (MLP)** is a feed-forward neural network that transforms input features through a series of learnable linear layers and nonlinear activations. It is the foundational building block used in many point-based LiDAR networks such as PointNet.

Structure and Operation Each MLP consists of:

- An **input layer** that receives the input features (e.g., coordinates (x, y, z) or (x, y, z, i)).
- One or more **hidden layers**, each performing a linear transformation followed by a nonlinear activation.
- An **output layer** that produces the transformed features or classification logits.

Each layer performs the operation:

$$\mathbf{h}_{k+1} = \sigma(\mathbf{W}_k \mathbf{h}_k + \mathbf{b}_k)$$

where $\mathbf{W}_k$ and $\mathbf{b}_k$ are the learnable weights and biases, and $\sigma(\cdot)$ is a nonlinear activation function such as ReLU or tanh.

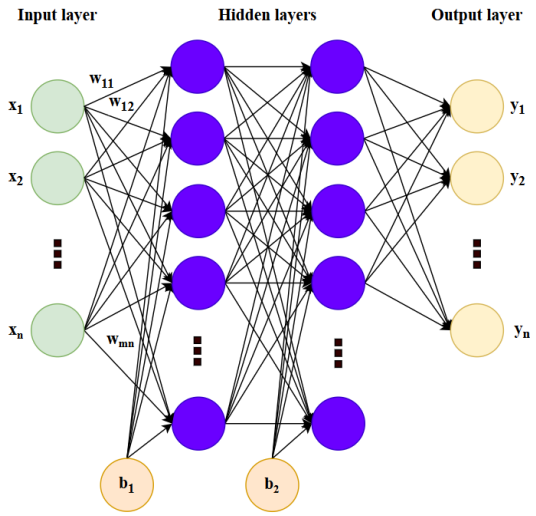


Figure A.1: Schematic of a multilayer perceptron (MLP). Each input feature (e.g., $x_1, x_2, ..., x_n$) is connected to neurons in the hidden layers through weighted connections (w_{ij}). Bias terms ($b_1, b_2, ...$) shift the activation, and the resulting outputs ($y_1, y_2, ..., y_n$) form the network's predictions or feature encodings.

Relevance to Point-Cloud Processing In point-based LiDAR architectures, the same MLP (with shared weights) is applied independently to every point in the cloud. This ensures that the transformation is identical

for all points and that the final output is invariant to the input order. After this per-point transformation, a symmetric aggregation function (such as max or mean pooling) combines all point features into a single, permutation-invariant global descriptor.

- MLPs can approximate complex nonlinear functions between raw points and higher-level geometric features.
- Shared weights enforce consistent learning across points, reducing parameters and improving generalisation.
- The combination of per-point MLPs and symmetric pooling forms the foundation of permutation-invariant point-cloud learning.

Appendix B

Kalman Filter from Preliminary Experiments 2 and 3

This appendix provides the complete mathematical formulation of the Kalman Filter (KF) used in Experiment 2 for centroid-level tracking on the BEV plane. The main text (**??**) summarises its role and implementation, while the full derivation and parameterisation are presented here for completeness.

State and prediction. The tracker models each tracked object's motion on the ground plane using a constant-velocity state:

$$\mathbf{x}_k = [x \; y \; \dot{x} \; \dot{y}]^\top,$$

where (x, y) are the estimated horizontal coordinates of the object's centroid in the BEV map and $(\dot{x}, \dot{y})$ are its planar velocities. The goal is to predict the next position based on its current location and estimated velocity.

State propagation between frames follows a simple linear motion model[1]:

$$\mathbf{x}_{k|k-1} = \mathbf{F}\mathbf{x}_{k-1|k-1}, \quad \mathbf{F} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where $\mathbf{F}$ is the state-transition matrix that advances the previous state by one time step Δt. In this constant-velocity model, the next position equals the previous position plus velocity multiplied by Δt, while the velocities themselves remain unchanged between frames.

Process noise. The predicted uncertainty in this motion is captured by the process-noise covariance:

$$\mathbf{Q} = \mathrm{diag}(q_x, q_y, q_{\dot{x}}, q_{\dot{y}}),$$

which represents unknown accelerations, small deviations from straight-line motion, or sensor jitter. Larger values of $\mathbf{Q}$ make the filter more responsive to sudden changes in motion, while smaller values make it

[1] $\mathbf{x}_{k|k-1}$ denotes the one-step-ahead prediction: the estimate of the state at time k given all measurements up to and including time $k-1$. In general, $\mathbf{x}_{t|s}$ means "state at t conditioned on data through s". After incorporating the measurement $\mathbf{z}_k$, the filtered (posterior) estimate is $\mathbf{x}_{k|k}$.

smoother and more inertial. The propagated uncertainty is then obtained as

$$\mathbf{P}_{k|k-1} = \mathbf{F}\mathbf{P}_{k-1|k-1}\mathbf{F}^\top + \mathbf{Q},$$

where $\mathbf{P}$ denotes the covariance matrix of the state estimate. This prediction $(\mathbf{x}_{k|k-1}, \mathbf{P}_{k|k-1})$ provides both the expected position and its uncertainty ellipse before the next detection arrives.

Measurement model. At each time step, the BEV heatmap supplies a noisy 2D position measurement:

$$\mathbf{z}_k = [x\ y]^\top.$$

This observation is linked to the state through the measurement matrix

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix},$$

which extracts only the (x, y) position components for comparison with the detected centroid. The measurement covariance $\mathbf{R}$ represents the expected localisation noise from the BEV heatmap; broader peaks or lower-confidence detections yield larger $\mathbf{R}$ values.

Kalman update. The Kalman update then combines the predicted and measured states to compute the posterior estimate $(\mathbf{x}_{k|k}, \mathbf{P}_{k|k})$, with standard update equations:[2]

$$\underbrace{\mathbf{v}_k}_{\text{innovation}} = \mathbf{z}_k - \mathbf{H}\mathbf{x}_{k|k-1}, \qquad \underbrace{\mathbf{S}_k}_{\text{innovation cov.}} = \mathbf{H}\mathbf{P}_{k|k-1}\mathbf{H}^\top + \mathbf{R}, \qquad \underbrace{\mathbf{K}_k}_{\text{Kalman gain}} = \mathbf{P}_{k|k-1}\mathbf{H}^\top\mathbf{S}_k^{-1}.$$

$$\mathbf{x}_{k|k} = \mathbf{x}_{k|k-1} + \mathbf{K}_k\,\mathbf{v}_k, \qquad \mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H})\,\mathbf{P}_{k|k-1}\,(\mathbf{I} - \mathbf{K}_k\mathbf{H})^\top + \mathbf{K}_k\,\mathbf{R}\,\mathbf{K}_k^\top.$$

This update step weights the two sources of information according to their confidence: if the detection is precise (small $\mathbf{R}$), the estimate shifts strongly toward the measurement; if it is noisy (large $\mathbf{R}$), the prediction is trusted more. The updated covariance $\mathbf{P}_{k|k}$ then shrinks or expands accordingly, reflecting the filter's new confidence in the estimate.

Summary. In practice, this Kalman Filter stabilised short-term tracks, smoothed jittery centroid motion, and maintained object identity through brief occlusions. Parameter tuning was empirical: $\mathbf{Q}$ was set proportionally to Δt^2 and $\mathbf{R}$ was derived from the width of the BEV detection peak. The algorithm's compact state representation and low computational cost made it suitable for real-time tracking within the clustering–tracking pipeline described in Chapter 6.

[2]The covariance update above is the Joseph form. One may also write $\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H})\,\mathbf{P}_{k|k-1}$.

Appendix C

Code

C.1 Cluster and Track - Key Functions

```python
from dataclasses import dataclass
from typing import List, Dict, Optional, Tuple
import numpy as np, math, torch, torch.nn as nn, torch.nn.functional as F

# ---- Minimal configs (key fields only) ----
@dataclass
class GeoMask:
    range_max_m: float = 95.0
    hfov_deg: float = 70.4
    r_full_m: float = 30.0
    center_at_far_deg: float = 5.0
    taper_power: float = 5.0
    ground_z_margin: float = 0.12
    use_planar_range: bool = True
    use_simple_ground: bool = True

@dataclass
class EvalCfg:
    assign_max_dist: float = 3.0
    adaptive_gate_k: float = 0.04
    mahal_xy_thr: float = 2.8
    low_density_factor: float = 0.40
    r_split_near: float = 25.0
    alpha_near: float = 0.60
    alpha_far: float = 0.40
    motion_smooth: float = 0.30
    emb_batch: int = 32768
    use_amp: bool = True

GMASK = GeoMask()
ECFG  = EvalCfg()
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"

# ---- Geometry masks ----
def _width_allowed_monotone(r, r_full, r_max, hfov_half, center_far, power):
    t = np.clip((r - r_full) / max(1e-6, (r_max - r_full)), 0.0, 1.0)
    ease = np.power(1.0 - t, power)
    theta = center_far + (hfov_half - center_far) * ease
    theta[r <= r_full] = hfov_half
    return r * np.tan(theta)

def fov_mask(xyz: np.ndarray, cfg: GeoMask = GMASK) -> np.ndarray:
    if xyz.size == 0: return np.zeros((0,), bool)
    x, y = xyz[:,0], xyz[:,1]
```

```
    r = np.sqrt(x*x + y*y) if cfg.use_planar_range else np.linalg.norm(xyz, axis=1)
    m_range = (r <= float(cfg.range_max_m)) & np.isfinite(r)
    hf = math.radians(cfg.hfov_deg * 0.5)
    cf = math.radians(cfg.center_at_far_deg)
    w = _width_allowed_monotone(r, cfg.r_full_m, cfg.range_max_m, hf, cf, cfg.taper_power)
    return m_range & (np.abs(y) <= w)

def simple_ground_mask(xyz: np.ndarray, z_margin: float = GMASK.ground_z_margin) -> np.ndarray:
    if xyz.size == 0: return np.zeros((0,), bool)
    x, y, z = xyz[:,0], xyz[:,1], xyz[:,2]
    r = np.sqrt(x*x + y*y)
    bins = np.linspace(0, 95.0, 20)
    keep = np.zeros(len(z), bool)
    for i in range(len(bins)-1):
        m = (r >= bins[i]) & (r < bins[i+1])
        if not m.any(): continue
        zg = np.percentile(z[m], 5)
        keep[m] = z[m] > (zg + z_margin)
    return keep

def pca_tighten_eval(pts: np.ndarray, low=5.0, high=95.0, min_span=0.25) -> np.ndarray:
    if pts.shape[0] < 5: return pts
    xy = pts[:, :2] - np.median(pts[:, :2], axis=0, keepdims=True)
    C = np.cov(xy.T) + 1e-6 * np.eye(2, dtype=np.float32)
    _, V = np.linalg.eigh(C)
    xy_r = xy @ V
    lo, hi = np.percentile(xy_r, low, axis=0), np.percentile(xy_r, high, axis=0)
    span = np.maximum(hi - lo, min_span)
    keep_xy = ((xy_r[:,0] >= lo[0]) & (xy_r[:,0] <= lo[0]+span[0]) &
               (xy_r[:,1] >= lo[1]) & (xy_r[:,1] <= lo[1]+span[1]))
    z = pts[:,2]; zlo, zhi = np.percentile(z, low), np.percentile(z, high)
    keep = keep_xy & ((z >= zlo) & (z <= zhi))
    return pts if not keep.any() else pts[keep]

# ---- Per-point / seed embeddings ----
class SetEncoder(nn.Module):
    """Per-point MLP -> max-pool -> embedding head (L2-normalized)."""
    def __init__(self, in_dim=7, hidden=256, emb_dim=128):
        super().__init__()
        self.point_mlp = nn.Sequential(
            nn.Linear(in_dim, hidden), nn.ReLU(True),
            nn.Linear(hidden, hidden), nn.ReLU(True),
            nn.Linear(hidden, 256), nn.ReLU(True),
        )
        self.head = nn.Sequential(nn.Linear(256, emb_dim), nn.LayerNorm(emb_dim))
    def point_feats(self, X):  # (B,N,7)->(B,N,256)
        return self.point_mlp(X)
    def pool_set(self, H):     # (B,N,256)->(B,256)
        return torch.max(H, dim=1).values
    def forward(self, X):      # (B,N,7)->(B,emb)
        H = self.point_feats(X); Z = self.head(self.pool_set(H))
        return F.normalize(Z, dim=1)

def _time_code_scalar(t_idx: int, denom: float = 100.0) -> Tuple[np.float32, np.float32]:
    tau = float(t_idx) / float(denom)
    return np.float32(np.cos(tau)), np.float32(np.sin(tau))

@torch.no_grad()
```

```python
def embed_points_pointwise(model: SetEncoder, P: np.ndarray, t_idx: int = 0, batch: int = None) -> np.ndarray:
    if P.size == 0: return np.zeros((0, 128), np.float32)
    batch = batch or ECFG.emb_batch
    out = []; use_amp = ECFG.use_amp and torch.cuda.is_available()
    for s in range(0, P.shape[0], batch):
        A = P[s:min(P.shape[0], s+batch)].astype(np.float32)
        x,y,z = A[:,0],A[:,1],A[:,2]
        r = np.sqrt(x*x + y*y); th = np.arctan2(y, x)
        tcos, tsin = _time_code_scalar(int(t_idx))
        X = torch.from_numpy(np.stack([x,y,z,r,th,
                                       np.full_like(r, tcos, np.float32),
                                       np.full_like(r, tsin, np.float32)], 1)
                             ).unsqueeze(0).to(DEVICE, non_blocking=True)
        if use_amp:
            with torch.amp.autocast(device_type="cuda", dtype=torch.float16):
                H = model.point_feats(X); Zp = model.head(H.squeeze(0))
        else:
            H = model.point_feats(X); Zp = model.head(H.squeeze(0))
        out.append(F.normalize(Zp, dim=1).float().cpu().numpy())
    return np.concatenate(out, 0)

@torch.no_grad()
def seed_proto(model: SetEncoder, S: np.ndarray, t_idx: int) -> np.ndarray:
    if S.shape[0] == 0: return np.zeros((128,), np.float32)
    K = min(512, max(32, S.shape[0]))
    A = S[np.random.choice(S.shape[0], size=K, replace=True)]
    x,y,z = A[:,0],A[:,1],A[:,2]
    r = np.sqrt(x*x + y*y); th = np.arctan2(y, x)
    tcos, tsin = _time_code_scalar(int(t_idx))
    X = torch.from_numpy(np.stack([x,y,z,r,th,
                                   np.full_like(r, tcos, np.float32),
                                   np.full_like(r, tsin, np.float32)], 1)
                         ).unsqueeze(0).to(DEVICE, non_blocking=True)
    with torch.amp.autocast(device_type="cuda", dtype=torch.float16, enabled=(ECFG.use_amp and torch.cuda.is_available())):
        H = model.point_feats(X); Z = model.head(model.pool_set(H))
    return F.normalize(Z.squeeze(0), dim=0).float().cpu().numpy()

# ---- Two-stage assignment (CORE + GROW) ----
def _alpha_for_range(seeds_xy: np.ndarray) -> float:
    if seeds_xy.size == 0: return ECFG.alpha_far
    r = np.linalg.norm(seeds_xy, axis=1)
    near_frac = (r < ECFG.r_split_near).mean() if r.size else 0.0
    return float(near_frac*ECFG.alpha_near + (1.0-near_frac)*ECFG.alpha_far)

def knn_density_xy(Pxy: np.ndarray, k: int = 16) -> np.ndarray:
    if Pxy.size == 0: return np.zeros((0,), np.float32)
    try:
        from scipy.spatial import cKDTree
        tree = cKDTree(Pxy.astype(np.float64), leafsize=64)
        k_eff = int(min(k + 1, max(2, len(Pxy))))
        dists, _ = tree.query(Pxy, k=k_eff, workers=-1)
        if dists.ndim == 1: dists = dists[:, None]
        rk = np.maximum(dists[:, 1:].max(axis=1).astype(np.float32), 1e-3)
        return (k / (np.pi * rk * rk)).astype(np.float32)
    except Exception:
        nb = 128
        H, xe, ye = np.histogram2d(Pxy[:,0], Pxy[:,1], bins=nb)
        xi = np.clip(np.searchsorted(xe, Pxy[:,0], side='right')-1, 0, nb-1)
```

```
        yi = np.clip(np.searchsorted(ye, Pxy[:,1], side='right')-1, 0, nb-1)
        area = (xe[1]-xe[0]) * (ye[1]-ye[0]) or 1.0
        return H[xi, yi].astype(np.float32) / np.float32(area)

def assign_two_stage(model: SetEncoder, P: np.ndarray, t_idx: int,
                     seeds_xy: np.ndarray, seeds_emb: np.ndarray,
                     mahal_state: Optional[Dict] = None) -> np.ndarray:
    """CORE: tight gate + margin; GROW: looser + density + Mahalanobis."""
    if P.shape[0] == 0 or seeds_xy.shape[0] == 0:
        return np.full((P.shape[0],), -1, int)

    Zp = embed_points_pointwise(model, P, t_idx)
    Zp_n = Zp / (np.linalg.norm(Zp, axis=1, keepdims=True) + 1e-8)
    Ze_n = seeds_emb / (np.linalg.norm(seeds_emb, axis=1, keepdims=True) + 1e-8)
    XY = P[:, :2].astype(np.float32)
    d_xy = np.linalg.norm(XY[:, None, :] - seeds_xy[None, :, :], axis=2)
    sim  = np.clip(Zp_n @ Ze_n.T, -1.0, 1.0)
    d_emb = 1.0 - sim

    alpha = _alpha_for_range(seeds_xy)
    seed_r_med = np.median(np.linalg.norm(seeds_xy, axis=1)) if seeds_xy.size else 0.0
    gate_xy = ECFG.assign_max_dist * (1.0 + ECFG.adaptive_gate_k * max(0.0, seed_r_med - 20.0))
    d_blend = alpha*d_xy + (1.0 - alpha)*d_emb

    # ---- CORE ----
    j1 = np.argmin(d_blend, axis=1); best = np.min(d_blend, axis=1)
    if d_blend.shape[1] >= 2:
        part2 = np.partition(d_blend, 1, axis=1)[:, :2]; second = part2.max(axis=1)
        margin = second - best
    else:
        margin = np.full_like(best, 1.0, np.float32)
    core_gate_xy = 0.75 * gate_xy
    thr_far_core = (1.4 + 0.02 * max(0.0, seed_r_med - 25.0))
    ok_xy  = d_xy[np.arange(d_xy.shape[0]), j1] <= core_gate_xy
    ok_mix = best <= (alpha*core_gate_xy + (1.0 - alpha)*thr_far_core)
    ok_mg  = margin >= 0.15
    asg = np.full((P.shape[0],), -1, int)
    core_mask = ok_xy & ok_mix & ok_mg
    asg[core_mask] = j1[core_mask]

    # ---- Update seed stats from cores (EMA + Mahalanobis) ----
    K = seeds_xy.shape[0]; mu_list, inv_list = [], []
    for k in range(K):
        hit = (asg == k)
        if not np.any(hit):
            mu_list.append(seeds_xy[k]); inv_list.append(np.eye(2, dtype=np.float32)); continue
        cl = pca_tighten_eval(P[hit])
        if cl.shape[0] == 0:
            mu_list.append(seeds_xy[k]); inv_list.append(np.eye(2, dtype=np.float32)); continue
        new_xy = np.median(cl[:, :2], 0).astype(np.float32)
        seeds_xy[k] = (1.0-ECFG.motion_smooth)*seeds_xy[k] + ECFG.motion_smooth*new_xy
        seeds_emb[k] = seed_proto(model, cl, t_idx)
        xy = cl[:, :2].astype(np.float32); mu = np.median(xy, axis=0).astype(np.float32)
        ctr = xy - mu
        if ctr.shape[0] < 8:
            var = np.maximum(ctr.var(axis=0), 1e-4)
            C = np.diag(var + 1e-4).astype(np.float32)
        else:
```

```
            C = np.cov(ctr.T, bias=False) + 1e-4*np.eye(2, dtype=np.float32)
        try: Cinv = np.linalg.inv(C)
        except np.linalg.LinAlgError: Cinv = np.linalg.pinv(C)
        mu_list.append(mu); inv_list.append(Cinv)
    if mahal_state is not None:
        mahal_state["mu"] = np.stack(mu_list, 0).astype(np.float32)
        mahal_state["covinv"] = np.stack(inv_list, 0).astype(np.float32)

    # ---- GROW ----
    remain = (asg < 0)
    if not np.any(remain): return asg
    rho = knn_density_xy(XY); med_rho = float(np.median(rho)) if rho.size else 0.0
    mu = np.stack(mu_list, 0).astype(np.float32); Cinv = np.stack(inv_list, 0).astype(np.float32)
    grow_gate_xy = 1.15 * gate_xy
    thr_far_grow = (1.7 + 0.02 * max(0.0, seed_r_med - 25.0))
    low_den_thr = ECFG.low_density_factor * (med_rho + 1e-6)

    for ii in np.where(remain)[0]:
        d = XY[ii][None, :] - mu
        q = np.einsum('ki,kij,kj->k', d, Cinv, d)
        mxy = np.sqrt(np.maximum(0.0, q))
        s_xy = d_xy[ii]
        s_blend = alpha*s_xy + (1.0 - alpha)*(1.0 - sim[ii])
        feas = (s_xy <= grow_gate_xy) & (mxy <= ECFG.mahal_xy_thr)
        if not np.any(feas): continue
        kbest = int(np.argmin(np.where(feas, s_blend, np.inf)))
        if (rho[ii] < low_den_thr) and (s_xy[kbest] > 0.9*grow_gate_xy): continue
        if s_blend[kbest] > (alpha*grow_gate_xy + (1.0 - alpha)*thr_far_grow): continue
        asg[ii] = kbest
    return asg

# ---- Minimal eval driver (signature only; full on GitHub) ----
def eval_recording(model: SetEncoder, frames: List[Dict]) -> Dict[str, float]:
    """Returns {'PointAcc': ..., 'B3F1': ...} for a contiguous sequence."""
    ...
```

Listing C.1: Core functions for joint clustering + tracking.

What each block does (Cluster+Track):

- Configs (GeoMask, EvalCfg): Hyperparameters that affect gating, range/field-of-view, density and smoothing.
- `fov_mask`, `simple_ground_mask`, `pca_tighten_eval`: Preprocessing.
- `SetEncoder`, `embed_points_pointwise`, `seed_proto`: Turns raw points into normalized embeddings; `seed_proto` builds a per-track prototype from its points.
- `assign_two_stage`: Main algorithm. CORE stage assigns only if close in XY, similar in embedding, and with a margin to the 2nd best. Then GROW stage admits harder points using density and a Mahalanobis gate fitted from CORE hits.
- `eval_recording` (part): Runs per-frame masking/assignment and calculates metrics (full loop is in GitHub).

C.2 Fine-grained Classification - Key Functions

```python
import numpy as np, torch, torch.nn as nn
from torch.utils.data import Dataset, DataLoader

# ---- DGCNN-style per-frame classifier (variable N) ----
def _gather_neighbors_flat(x, idx):
    B,N,F = x.shape; k = idx.size(-1)
    base = torch.arange(B, device=x.device).view(B,1,1)*N
    return x.reshape(B*N, F)[(idx + base).reshape(-1), :].reshape(B,N,k,F)

class EdgeConv(nn.Module):
    def __init__(self, in_ch, out_ch):
        super().__init__()
        self.mlp = nn.Sequential(nn.Linear(in_ch*2, out_ch, bias=False),
                                 nn.BatchNorm1d(out_ch),
                                 nn.LeakyReLU(0.2, inplace=False))
    def forward(self, x, mask, idx):
        B,N,F = x.shape; k = idx.size(-1)
        xi = x.unsqueeze(2).expand(-1,-1,k,-1); xj = _gather_neighbors_flat(x, idx)
        e = torch.cat([xi, xj - xi], -1).reshape(B*N*k, 2*F)
        f = self.mlp(e).view(B,N,k,-1).max(2)[0] * mask.unsqueeze(-1).float()
        return f

def knn_graph_masked(x, mask, k):
    B,N,F = x.shape; device = x.device
    idx_out = torch.zeros(B,N,k, dtype=torch.long, device=device)
    vc = mask.sum(1)
    for b in range(B):
        n = int(vc[b]); inds = torch.nonzero(mask[b], as_tuple=False).squeeze(1)
        if n <= 1:
            fill = int(inds[0].item()) if n==1 else 0
            idx_out[b].fill_(fill); continue
        xb = x[b, inds, :]
        try:
            with torch.amp.autocast('cuda', enabled=x.is_cuda): d = torch.cdist(xb, xb)
        except RuntimeError:
            d = torch.cdist(xb.cpu(), xb.cpu()).to(device)
        d.fill_diagonal_(float('inf'))
        k_eff = min(k, max(1, n-1))
        nbr_local = d.topk(k_eff, dim=1, largest=False).indices
        if k_eff < k: nbr_local = torch.cat([nbr_local, nbr_local[:, :1].expand(n, k-k_eff)], 1)
        idx_out[b, inds, :] = inds[nbr_local]
        if (~mask[b]).any(): idx_out[b, ~mask[b], :] = inds[0]
    return idx_out

def masked_max_mean(f, mask):
    maskC = mask.unsqueeze(-1); f_masked = f.masked_fill(~maskC, float('-inf'))
    f_max = torch.where(torch.isfinite(torch.amax(f_masked, dim=1)), torch.amax(f_masked, dim=1), torch.zeros_like(f_masked[:,0,:]))
    s = (f * maskC.float()).sum(1); cnt = mask.float().sum(1, keepdim=True).clamp_min(1.0)
    return f_max, s / cnt

class DGCNNVarN(nn.Module):
    def __init__(self, in_ch=4, k=16, num_classes=3):
        super().__init__()
        self.k = k
        self.ec1 = EdgeConv(in_ch, 64)
```

```
        self.ec2 = EdgeConv(64, 128)
        self.ec3 = EdgeConv(128, 256)
        glob = (64+128+256)*2
        self.head = nn.Sequential(
            nn.Linear(glob, 256, bias=False), nn.BatchNorm1d(256), nn.LeakyReLU(0.2, inplace=False), nn.Dropout(0.3),
            nn.Linear(256, 256, bias=False), nn.BatchNorm1d(256), nn.LeakyReLU(0.2, inplace=False), nn.Dropout(0.3),
            nn.Linear(256, num_classes),
        )
    def forward(self, x, mask):
        idx = knn_graph_masked(x, mask, self.k)
        f1 = self.ec1(x, mask, idx)
        f2 = self.ec2(f1, mask, idx)
        f3 = self.ec3(f2, mask, idx)
        gmax, gmean = masked_max_mean(torch.cat([f1,f2,f3], -1), mask)
        return self.head(torch.cat([gmax, gmean], -1))

# ---- Temporal head over cached logits (TxC) ----
class TinyTemporalHead(nn.Module):
    """Lightweight conv+attention aggregator across time."""
    def __init__(self, C, hid=64, num_classes=3):
        super().__init__()
        self.proj = nn.Linear(C, hid)
        self.conv = nn.Conv1d(hid, hid, 3, padding=1)
        self.attn_v = nn.Linear(hid, 1, bias=False)
        self.fc = nn.Sequential(nn.Linear(hid,128), nn.ReLU(True), nn.Dropout(0.3),
                                nn.Linear(128, num_classes))
    def forward(self, X, M):     # X:(B,T,C)  M:(B,T)
        H = self.proj(X)        # (B,T,H)
        Hc = self.conv(H.transpose(1,2)).transpose(1,2)
        H = (H + Hc) * M.unsqueeze(-1).to(H.dtype)
        s = self.attn_v(torch.tanh(H)).squeeze(-1)            # (B,T)
        s = s.masked_fill(~M, torch.tensor(-1e4, dtype=s.dtype, device=s.device))
        w = torch.softmax(s - s.max(1, keepdim=True).values, 1)
        return self.fc((H * w.unsqueeze(-1)).sum(1))          # (B,C)

# ---- Concise eval hooks ----
@torch.no_grad()
def eval_frame(model, dl, device="cuda" if torch.cuda.is_available() else "cpu"):
    model.eval(); crit = nn.CrossEntropyLoss(); tot=0.0; n=0; acc=0.0
    for X,M,Y in dl:
        X,M,Y = X.to(device), M.to(device), Y.to(device)
        with torch.amp.autocast('cuda', enabled=(device=='cuda')):
            logits = model(X,M); loss = crit(logits, Y)
        tot += loss.item()*Y.size(0); n += Y.size(0); acc += (logits.argmax(1)==Y).sum().item()
    return tot/max(1,n), acc/max(1,n)

@torch.no_grad()
def evaluate_temporal(model, dl, crit):
    model.eval(); tot=0.0; n=0; acc=0.0
    for X,M,Y in dl:
        X,M,Y = X.to(DEVICE), M.to(DEVICE), Y.to(DEVICE)
        with torch.amp.autocast('cuda', enabled=(DEVICE=='cuda')):
            logits = model(X,M); loss = crit(logits, Y)
        tot += loss.item()*Y.size(0); n += Y.size(0); acc += (logits.argmax(1)==Y).sum().item()
    return tot/max(1,n), acc/max(1,n)
```

Listing C.2: Core per-frame classifier and temporal fusion head.

What each block does (Classification):

- `EdgeConv` & `_gather_neighbors_flat`: Builds edge features $(x_j - x_i, x_i)$ from kNN and learns local geometry.
- `knn_graph_masked`: kNN graph that handles variable point counts per crop.
- `masked_max_mean`: Global pooling under the mask (gets both max and mean for stability).
- `DGCNNVarN`: Per-frame classifier: stacks 3 EdgeConv layers, concatenates features, pools, then classifies.
- `TinyTemporalHead`: Fuses per-frame logits over time via linear projection, a small 1D conv, and attention; outputs a sequence label.
- `eval_frame`, `evaluate_temporal`: Minimal evaluation loops for per-frame and temporal heads.

Appendix D

Graduate Attributes

Graduate Attribute	Description (Activities, Skills or Abilities)	How this was achieved in the project
GA 1: Problem Solving	Identify, formulate, research literature and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences with holistic considerations for sustainable development.	Reviewed existing literature on LiDAR-based classification and nocturnal animal detection to ground the problem context. Broke the thesis into sub-problems: extracting clusters, tracking, and fine-grained classification. Designed file structures, data flow, and naming conventions - referring to how intermediate data and results were consistently stored, labelled, and referenced across scripts and experiments. Applied principles from mathematics and machine learning (amongst other fields) to design an original, multi-stage LiDAR classification pipeline addressing a novel, real-world problem.
GA 4: Investigations, Experiments and Data Analysis	Demonstrate competence to conduct investigations of complex engineering problems using research methods, including design of experiments, data analysis and synthesis of information to provide valid conclusions.	Designed and conducted investigations across several datasets and scenarios. Used the KITTI dataset to test clustering algorithms, collected a "laundry basket" dataset for test tracking, and a human movement dataset to test temporal classification. Collected a final dataset at a public dog park to evaluate the end-to-end system under realistic conditions, applying lessons from earlier experiments. Balanced frame size (120k vs 240k points) to trade off resolution versus stability, and experimented with different machine learning approaches, architectures, and feature sets. Analysed results using statistical and visual metrics (accuracy, latency, F1) and engaged critically with literature to interpret and explain the results.
GA 5: Use of Engineering Tools	Demonstrate competence to create, select and apply, and recognise limitations of appropriate techniques, resources and modern engineering and IT tools, including prediction and modelling.	Used Python, PyTorch, NumPy, Pandas, Open3D, and other supporting libraries for development, alongside MATLAB (where its limitations were later identified). Leveraged GPU acceleration (CUDA) for train more quickly. Annotated data using CVAT and visualised plots using Matplotlib and Plotly. Developed Python scripts to convert between formats for point-cloud storage, preprocessing, plots, annotating datasets, etc. . Demonstrated familiarity with tools and awareness of their practical constraints.
GA 6: Professional and Technical Communication	Demonstrate competence to communicate effectively and inclusively on complex engineering activities, both orally and in writing, with appropriate academic and professional discourse.	Produced a comprehensive technical report with clear diagrams, tables, and confusion matrices. Communicated progress and results effectively during supervisor meetings. Presented and articulated experimental requirements to volunteers, including the dog owners. Integrated written, verbal, and visual communication at a professional standard suitable for both academic and engineering audiences.
GA 8: Individual and Collaborative Teamwork	Demonstrate competence to function effectively as an individual and as a member or leader in diverse and multi-disciplinary teams and settings.	Worked independently on all major project components, from problem formulation to implementation and analysis. Maintained steady progress through personal milestones and responsible time management. Collaborated constructively with supervisors and assistants during field data collection, demonstrating initiative, accountability, and leadership.
GA 9: Independent Learning Ability	Demonstrate competence to engage in independent learning through well-developed learning skills, adaptability to new and emerging technologies, and critical thinking in evolving contexts.	Self-taught advanced tools beyond coursework, including point-cloud processing libraries, PyTorch deep-learning frameworks, and data annotation software. Overcame technical challenges (replacing the incompatible MinkowskiEngine with a self-designed voxel-based classifier). Learned from literature and benchmark datasets, reflected on supervisor feedback, and adjusted methods accordingly. Developed a deep understanding of key evaluation metrics for this system and demonstrated resilience, curiosity, and adaptability to new technologies.

Table D.1: Description of each Graduate Attribute (GA) and how it was demonstrated through this project.

Appendix E

Use of AI Tools

AI tools were used in a limited, supporting role. Uses included:

- Improving clarity and consistency when it came to more technical writing.
- Helping with LaTeX structure (tables, figures, captions) and fixing compilation issues.
- Used for some low-level coding, cleanups, and general debugging.
- Suggested different annotation softwares.
- Helping with clearer notation and more consistent presentation of equations and metrics.
- Proposing a layout for some sections to improve readability (e.g., tables).

These tools were not used to produce the core technical contributions or experimental results; all analysis, system design, and conclusions are my own. Sources found using AI-assisted search were independently verified and cited in the bibliography.