A Comparative Study of Sequential and End-to-End LiDAR Perception Frameworks in Autonomous Driving

Scientific work for obtaining the academic degree
Master of Science (M.Sc.)
at the Department Mobility Systems Engineering
of the TUM School of Engineering and Design
at the Technical University of Munich

Supervised by Prof. Dr.-Ing. Markus Lienkamp
Max Musterbetreuer, M.Sc.
Chair of Automotive Technology

Submitted by Parim Suka
Am Mitterfeld 78
81829 München

Submitted on May 26, 2025

Lehrstuhl für Fahrzeugtechnik
Fakultät für Maschinenwesen
Technische Universität München

MA

A Comparative Study of Sequential and End-to-End LiDAR Perception Frameworks in Autonomous Driving

Autonomous driving (AD) is a highly sought-after technology in which strong advances have been made – as demonstrated by multiple companies and research institutions that already test autonomous vehicles on public roads. However, their reliability, specifically in perception, must still be significantly increased to offer fully autonomous vehicles to the open market. AD perception software stacks are typically divided into cascaded detection, tracking and prediction (DTP) modules with high performance and generalization requirements. Today, literature shows that unified perception architectures, sharing information and trained end-to-end, can outperform sequential architectures.

This work aims to analyze the performance of sequential and unified (end-to-end) perception. Specifically, an evaluation framework should be implemented, allowing for the quantification and in-depth comparison of sequential approaches. Furthermore, a unified tracking-prediction architecture should be developed and compared against sequential alternatives.

Milestones: Firstly, a thorough literature review on existing DTP modules should be conducted. Subsequently, representative DTP modules should be implemented into a common framework, together with the evaluation pipeline. Furthermore, a joint tracking and prediction architecture should be developed. Lastly, a critical analysis of the results should be conducted.

Work packages:

- Literature review: Sequential and unified (end-to-end) perception.
- Framework implementation: Sequential DTP architectures + evaluation.
- Model implementation: Joint tracking and prediction network.
- Critical analysis of the results.

Requirements:

- Very good programming skills in Python or C++.
- High personal motivation and independent working style.
- Very good language proficiency in German, English or French.

The thesis should clearly document the individual work steps. The candidate undertakes to complete the term paper independently and to indicate all scientific aids used.

The submitted work remains the property of the chair as an examination document.

Prof. Dr.-Ing. M. Lienkamp

Betreuer: Loïc Stratil, M. Sc.

Ausgabe:_____________

Abgabe: _____________

Chair of Automotive Technology
TUM School of Engineering and Design
Technical University of Munich

Geheimhaltungsverpflichtung

Herr: **Suka, Parim**

Gegenstand der Geheimhaltungsverpflichtung sind alle mündlichen, schriftlichen und digitalen Informationen und Materialien die der Unterzeichner vom Lehrstuhl oder von Dritten im Rahmen seiner Tätigkeit am Lehrstuhl erhält. Dazu zählen vor allem Daten, Simulationswerkzeuge und Programmcode sowie Informationen zu Projekten, Prototypen und Produkten.

Der Unterzeichner verpflichtet sich, alle derartigen Informationen und Unterlagen, die ihm während seiner Tätigkeit am Lehrstuhl für Fahrzeugtechnik zugänglich werden, strikt vertraulich zu behandeln.

Er verpflichtet sich insbesondere:

- derartige Informationen betriebsintern zum Zwecke der Diskussion nur dann zu verwenden, wenn ein ihm erteilter Auftrag dies erfordert,
- keine derartigen Informationen ohne die vorherige schriftliche Zustimmung des Betreuers an Dritte weiterzuleiten,
- ohne Zustimmung eines Mitarbeiters keine Fotografien, Zeichnungen oder sonstige Darstellungen von Prototypen oder technischen Unterlagen hierzu anzufertigen,
- auf Anforderung des Lehrstuhls für Fahrzeugtechnik oder unaufgefordert spätestens bei seinem Ausscheiden aus dem Lehrstuhl für Fahrzeugtechnik alle Dokumente und Datenträger, die derartige Informationen enthalten, an den Lehrstuhl für Fahrzeugtechnik zurückzugeben.

Eine besondere Sorgfalt gilt im Umgang mit digitalen Daten:

- Für den Dateiaustausch dürfen keine Dienste verwendet werden, bei denen die Daten über einen Server im Ausland geleitet oder gespeichert werden (Es dürfen nur Dienste des LRZ genutzt werden (Lehrstuhllaufwerke, Sync&Share, GigaMove).
- Vertrauliche Informationen dürfen nur in verschlüsselter Form per E-Mail versendet werden.
- Nachrichten des geschäftlichen E-Mail Kontos, die vertrauliche Informationen enthalten, dürfen nicht an einen externen E-Mail Anbieter weitergeleitet werden.
- Die Kommunikation sollte nach Möglichkeit über die (my)TUM-Mailadresse erfolgen.

Die Verpflichtung zur Geheimhaltung endet nicht mit dem Ausscheiden aus dem Lehrstuhl für Fahrzeugtechnik, sondern bleibt 5 Jahre nach dem Zeitpunkt des Ausscheidens in vollem Umfang bestehen. Die eingereichte schriftliche Ausarbeitung darf der Unterzeichner nach Bekanntgabe der Note frei veröffentlichen.

Der Unterzeichner willigt ein, dass die Inhalte seiner Studienarbeit in darauf aufbauenden Studienarbeiten und Dissertationen mit der nötigen Kennzeichnung verwendet werden dürfen.

Datum: 26. Mai 2025

Unterschrift: ______________________

Chair of Automotive Technology
TUM School of Engineering and Design
Technical University of Munich

Erklärung

Ich versichere hiermit, dass ich die von mir eingereichte Abschlussarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Garching, den 26. Mai 2025

P. Suka

Parim Suka, B.Sc.

Chair of Automotive Technology
TUM School of Engineering and Design
Technical University of Munich

Declaration of Consent, Open Source

Hereby I, Suka, Parim, born on June 17, 2000, make the software I developed during my Semester Thesis available to the Institute of Automotive Technology under the terms of the license below.

Garching, May 26, 2025

P. Suka

Parim Suka, B.Sc.

Copyright 2025 Suka, Parim

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Contents

Formula Symbols .... **V**

1 Introduction **1**

  1.1 Motivation and Problem Statement **1**

  1.2 Research Objectives and Contributions **2**

2 Background and Literature Review **4**

  2.1 LiDAR-Based Autonomous Driving Perception Datasets **4**

  2.2 Evaluation Metrics for LiDAR-based Perception on nuScenes **6**

    2.2.1 Stage 1: Detection 7

    2.2.2 Stage 2: Tracking 8

    2.2.3 Stage 3: Prediction 11

  2.3 Object Detection in Autonomous Systems **12**

    2.3.1 Voxel-Based Approaches 13

    2.3.2 Point-Based Approaches 13

    2.3.3 BEV-Based Fusion and Multi-Modal Approaches 13

    2.3.4 Benchmark Performance on nuScenes (2022–2025) 14

  2.4 Multi-Object Tracking (MOT) **15**

    2.4.1 Traditional Tracking-by-Detection Methods (Non-Learnable) 15

    2.4.2 Learnable and End-to-End MOT Models 16

    2.4.3 NuScenes Tracking Benchmark: Performance Comparison 18

  2.5 Trajectory Prediction **18**

    2.5.1 RNN and Graph-Based Models (Trajectron++) 19

    2.5.2 Anchor and Goal-Based Models (CoverNet, TNT, Goal-LBP) 19

    2.5.3 Attention and Transformer-Based Models (WIMP, Transformer nets) 21

    2.5.4 Occupancy Grid and Joint Prediction Methods (CASPNet++) 22

    2.5.5 Performance on nuScenes: Benchmark Metrics and Model Comparison 22

  2.6 End to end - joint training **24**

    2.6.1 Joint Detection and Prediction Models (LiDAR-Only) 24

    2.6.2 Integrated Tracking and Prediction Models 25

    2.6.3 Unified Detection–Tracking–Prediction Systems 25

  2.7 Modular vs. End-to-End LiDAR Perception Pipelines: A Literature Review **26**

    2.7.1 Research Gaps and Thesis Positioning 28

3 Methodology & System Architecture **30**

  3.1 Overview of the Proposed Pipeline **30**

  3.2 Implementation Details **32**

  3.3 Joint Tracking and Prediction via Differentiable 3DMOTFormer and Trajectron++ **44**

    3.3.1 System Overview 44

    3.3.2 Differentiable Data Association: Sinkhorn Matching 44

    3.3.3 Sinkhorn Algorithm for Soft Matching 46

    3.3.4 Soft Track State Updates 48

    3.3.5 Gradient Flow Through the Tracker 50

3.3.6 Joint Training Pipeline 51
4 Results 52
4.1 Detection Results 52
4.2 Tracking Results 54
4.3 Learning-Based Tracking Results 55
4.4 Prediction Results 57
4.4.1 Trajectron++ Baseline Performance 57
4.4.2 Prediction Performance by Tracking Input and Ablations 58
4.4.3 Full Perception Pipeline Overview 58
4.4.4 Joint model various training configurations 60
5 Discussion of Results 61
5.1 Detection Results 61
5.2 Tracking Results Discussion 62
5.2.1 Analysis of PolyMOT Performance 63
5.2.2 Analysis of 3DMOTFormer Performance Across Scenarios 63
5.2.3 Overall Tracking Performance Comparison 67
5.3 Prediction Results Discussion 68
5.3.1 Analysis of Trajectron++ Baseline Performance 68
5.3.2 Impact of Upstream Modules on Prediction Accuracy 69
6 Conclusion 75
7 Outlook 77
List of Figures i
List of Tables iii
Bibliography v
Appendix xi

Formula Symbols

Formula Symbols	Unit	Description
$\mathbf{x}$	m	Position vector of an object in 2D or 3D space
$\mathbf{v}$	$\mathrm{m\,s^{-1}}$	Velocity vector of an object
$\mathbf{a}$	$\mathrm{m\,s^{-2}}$	Acceleration vector of an object
θ	rad	Orientation angle of a bounding box or object heading
$\mathbf{p}_i^t$	m	Position of object i at time t
$\mathbf{v}_i^t$	$\mathrm{m\,s^{-1}}$	Velocity of object i at time t
ADE	m	Average Displacement Error over all time steps
FDE	m	Final Displacement Error at the last prediction step
MR	%	Miss Rate: percentage of predictions beyond a distance threshold
AMOTA	-	Average Multi-Object Tracking Accuracy
AMOTP	-	Average Multi-Object Tracking Precision
IDS	-	Number of Identity Switches
mAP	%	Mean Average Precision for object detection
NDS	%	nuScenes Detection Score (holistic detection metric)
P_{ij}	-	Soft assignment weight between track i and detection j (Sinkhorn output)
C_{ij}	-	Matching cost between track i and detection j
$\mathcal{L}_{\text{pred}}$	-	Loss function for trajectory prediction
$\mathcal{L}_{\text{track}}$	-	Loss function for tracking module
$\mathcal{L}_{\text{joint}}$	-	Combined loss for joint training
a_{ij}	-	Affinity score between track i and detection j
$h_{A,ij}$	-	Edge feature representation between track i and detection j in the attention graph
T	s	Prediction horizon in seconds
Δt	s	Time step interval between frames
λ	-	Entropy regularization parameter in the Sinkhorn algorithm
$H(P)$	-	Entropy of the assignment matrix P

α, β	-	Weighting coefficients for multi-task loss components
$\hat{\mathbf{y}}_i^t$	m	Predicted future position of object i at time t
$\mathbf{y}_i^t$	m	Ground truth future position of object i at time t
$\dot{\mathbf{x}}$	$\mathrm{m\,s^{-1}}$	Time derivative of position (velocity)
$\ddot{\mathbf{x}}$	$\mathrm{m\,s^{-2}}$	Time derivative of velocity (acceleration)

1 Introduction

Autonomous driving is expected to significantly change how we travel by making transportation safer, more efficient, and more accessible. At the heart of any reliable self-driving system is the perception module, which helps the vehicle understand its surroundings. This involves several key tasks: detecting objects, following their movement over time, and predicting where they will go next. How well these perception tasks perform has a direct effect on the safety and reliability of the vehicle, which is why they are a major focus in current research and development.

1.1 Motivation and Problem Statement

The traditional approach to environmental perception in autonomous driving systems often follows a sequential pipeline: Detection, Tracking, and then Prediction (DTP). In this paradigm, dedicated modules perform each task independently. A detector first identifies objects in sensor data (e.g., LiDAR point clouds)[1–3]. These detections are then fed into a tracking module, which associates detections across consecutive frames to form coherent object trajectories. Finally, a prediction module consumes these historical tracks to forecast the future behavior of dynamic agents.

This sequential architecture offers several practical advantages. Its modularity allows for individual components to be developed, optimized, and debugged in isolation, often leveraging highly specialized algorithms and datasets. This separation simplifies system complexity and facilitates incremental improvements. However, this inherent modularity also gives rise to significant limitations. Errors generated at an earlier stage, particularly false positives, false negatives, or inaccurate localization from the detection module, inevitably propagate downstream to the tracking and prediction modules. These errors can compound, leading to degraded overall system performance and potentially unsafe decision-making. Furthermore, each module in the sequential pipeline is typically optimized independently for its specific sub-task, often without explicit consideration for the impact on the subsequent stages or the final end-to-end performance of the entire perception stack. This setup can limit the overall performance of the system, because each part is optimized separately and might not fully support the main goal of accurately predicting future states. These observations raise an important research question:

> *Can trajectory prediction quality be improved by tightly coupling tracking and prediction into a single, differentiable module—rather than treating them as isolated stages?*

In response to these challenges, there has been a growing interest in integrated, end-to-end learning paradigms for perception. These approaches aim to optimize multiple perception tasks jointly, allowing for potential information flow and feedback across stages, and mitigating the issue of error propagation by design. While promising, a comprehensive and rigorous comparative analysis between the established sequential DTP pipeline and these emerging joint, end-to-end differentiable frameworks for tracking and prediction remains a critical gap in the literature. This kind of analysis needs to go deeper than basic comparisons, by testing how the systems perform in realistic conditions and understanding how different factors affect the results.

The central problem addressed by this thesis is therefore: to thoroughly compare the performance, robustness, and error characteristics of a state-of-the-art sequential Detection-Tracking-Prediction pipeline against a joint, end-to-end differentiable Tracking-Prediction framework within the context of autonomous driving. This comparison is vital in determining which paradigm offers superior performance and is more suitable for real-world deployment, especially considering the complexities of sensor data and dynamic environments.

1.2 Research Objectives and Contributions

Building upon the identified problem, this thesis aims to provide a detailed, quantitative comparison between sequential and joint perception paradigms. Our primary objective is to evaluate the trade-offs inherent in each approach, specifically focusing on their impact on multi-object tracking and trajectory prediction performances.

To achieve this overall goal, the following specific research objectives have been defined:

- **Implementation of a State-of-the-Art Sequential Perception Pipeline:** To set up and integrate a robust sequential DTP pipeline, leveraging a high-performing 3D object detector and a widely recognized rule-based multi-object tracker, followed by a trajectory prediction model.
- **Implementation of a Joint Differentiable Tracking and Prediction Framework:** To implement and integrate an end-to-end framework where tracking and prediction are jointly optimized and fully differentiable, allowing for direct learning of associations and future trajectories.
- **Comprehensive Comparative Evaluation under Real-World Input Conditions:** To quantitatively compare the performance of both the sequential and joint paradigms using real detection outputs generated by the detector, assessing their accuracy, robustness, and failure cases on standard benchmarks (as discussed in Chapter 2).
- **Impact of Input Accuracy:** To test how both sequential and joint systems perform when using perfect ground truth detections. This helps separate the actual weaknesses of the tracking and prediction models from the extra errors caused by inaccurate detections.
- **Comparative Study of Tracker Paradigms:** To investigate the performance differences between a non-learnable tracker and a learnable tracker component in the sequential pipeline, analyzing their respective strengths and weaknesses in different scenarios.
- **Error Propagation Analysis:** To provide insights into how errors propagate and compound in the sequential pipeline versus how a joint, differentiable optimization approach might mitigate these issues by enabling end-to-end learning.

Based on the objectives outlined above, the following points summarize what this thesis has achieved and contributed to the field of autonomous driving perception:

- **A Unified Benchmarking Framework:** A reproducible experimental framework was developed to enable direct comparison between modular (sequential) and integrated (joint) perception systems, ensuring consistent evaluation of tracking and prediction components.

- **Extensive Experimental Evaluation:** Large-scale quantitative experiments were conducted across multiple configurations, generating data that reveals how system design choices influence tracking and prediction performance under realistic conditions.
- **Analysis of Detection Quality Effects:** The impact of input quality—by comparing real detections and ground-truth annotations—was systematically investigated to understand how error accumulation differs between pipeline designs.
- **Comparison of Tracking Strategies:** Learnable and rule-based trackers were evaluated within the sequential pipeline to identify their respective strengths and weaknesses, highlighting trade-offs in accuracy, stability, and implementation complexity.
- **Design Insights for End-to-End Architectures:** A detailed analysis of error propagation and optimization flow was performed, offering practical guidance for the development of future joint tracking and prediction systems using end-to-end training approaches.

The subsequent chapters of this thesis will elaborate on the methodology employed, present the detailed experimental results, and discuss the implications of our findings for advancing autonomous driving perception.

2 Background and Literature Review

This chapter provides an overview of the current state of the art in autonomous driving perception, with a focus on the core components of the detection-tracking-prediction pipeline. It begins by presenting the most widely used datasets in this domain, highlighting their structure, sensor setups, and relevance for multi-task evaluation. The chapter then outlines the evaluation metrics used to assess the performance of each pipeline stage, including metrics for object detection, multi-object tracking, and trajectory prediction. Together, these sections provide the technical foundation necessary for understanding the models and experimental results presented in later chapters.

2.1 LiDAR-Based Autonomous Driving Perception Datasets

A number of public datasets have played a key role in advancing 3D object detection, multi-object tracking, and motion prediction in autonomous driving. These datasets differ in terms of LiDAR sensor setup, annotation detail, geographic and temporal coverage, supported tasks, and official benchmarks.

The nuScenes dataset[1], released by Aptiv in 2019, was the first large-scale dataset to include a full sensor suite with six cameras, five radars, and one 360° LiDAR operating at 20 Hz. It contains 1000 driving scenes, each lasting 20 seconds, collected in Boston and Singapore. Around 40,000 keyframes are annotated at 2 Hz with 3D bounding boxes for 23 object classes and eight attributes. Each object has a global tracking ID, allowing tracking throughout a scene. High-definition semantic maps are also available. NuScenes supports detection, tracking, and prediction tasks. Its benchmark introduced new metrics like mean Average Precision (mAP) and the nuScenes Detection Score (NDS), and later included tracking metrics such as AMOTA and AMOTP. The dataset offers consistent 3D annotations, rich sensor data, and diverse driving conditions, making it widely used for research in detection, tracking, and motion forecasting.

The Waymo Open Dataset (WOD)[3], released between 2019 and 2021, is one of the largest available datasets. It includes five LiDAR scanners — one long-range top LiDAR and four short-range side LiDARs — as well as five cameras covering 360°. The original release had 1000 20-second segments sampled at 10 Hz, containing about 200,000 LiDAR frames and over 12 million labeled 3D boxes. A later version increased the number of segments to over 2000. Each object has a tracking ID, and the original release includes around 113,000 unique tracks. Four object types are labeled, and objects are annotated up to 75 meters. The data was collected across several US cities including Phoenix, San Francisco, Detroit, and Seattle, under various weather and lighting conditions. The dataset includes benchmarks for 3D detection and tracking. In 2021, a separate motion dataset was introduced for prediction research, containing over 100,000 vehicle trajectories along with map information. The perception dataset focuses on raw sensor data, while the motion dataset emphasizes longer-term trajectory forecasting. With dense point clouds and a wide variety of scenes, WOD remains a leading resource for detection, tracking, and forecasting studies.

The KITTI dataset[2, 4, 5], released in 2012, was one of the earliest datasets for autonomous driving. It includes LiDAR data from a roof-mounted 64-beam scanner and imagery from four cameras. The 3D detection benchmark provides 7481 training and 7518 test frames with annotations for up to eight object classes, mainly Car, Pedestrian, and Cyclist. Around 80,000 labeled instances are available. The tracking benchmark contains 21 training and 29 test sequences, each lasting between 15 and 40 seconds. Tracking annotations include consistent object IDs, but official evaluation focuses on Cars and Pedestrians. KITTI data was collected at 10 Hz in daylight and good weather. Although influential, KITTI is limited in scope, with a small geographic area, fewer scenes, and less temporal context. It lacks map annotations and long trajectories for prediction. The detection benchmark does not include tracking IDs, and the tracking set is relatively small. Despite this, KITTI became a widely used benchmark for 3D object detection, especially in earlier research.

The Argoverse datasets[6, 7] were released by Argo AI and focus on both perception and trajectory prediction tasks. Argoverse 1 includes a 3D tracking dataset with 113 driving scenes and a separate motion forecasting set with over 327,000 tracked scenarios. The tracking scenes are 15 to 30 seconds long and were collected using two roof-mounted LiDARs and multiple cameras, with annotations for vehicles, pedestrians, and other dynamic objects. High-definition maps of Miami and Pittsburgh are provided, including lane centerlines and traffic directions. Argoverse 1 was limited in terms of the number of driving scenes, which affected its suitability for training detection and tracking models.

Argoverse 2 addresses these limitations by increasing the number of annotated 3D scenes to 1000 and expanding coverage to six cities, including Austin, Detroit, and Washington D.C. The sensor setup remains similar, but annotation quality and object diversity are improved. The updated motion forecasting set contains 250,000 scenarios, including richer labels such as vehicle behaviors. The HD maps are also enhanced, providing more detailed semantic information. Argoverse is especially strong in tasks that combine perception with map-based reasoning, though its adoption for detection and tracking has been less widespread than nuScenes or Waymo.

Several other datasets have also contributed to 3D perception research. The Lyft Level 5 dataset[8] includes multimodal data with consistent annotations and HD maps, mainly for 3D detection. ApolloScape[9] offers urban driving data from Beijing with dense traffic and annotations for detection and tracking. PandaSet[10], created by Hesai and Scale AI, includes data from mechanical and solid-state LiDARs and is used for research on sensor fusion and fine-grained labels. ONCE[11] is a large-scale dataset with over one million LiDAR frames, designed for semi-supervised learning, though only part of the data is labeled. A*3D[12] focuses on challenging conditions such as night and adverse weather, supporting robust 3D detection. H3D[13] captures crowded urban scenes with heavy occlusions and provides dense 3D tracking annotations, making it useful for studying tracking in complex environments.

These datasets collectively support a wide range of perception tasks in autonomous driving, each offering unique strengths based on sensor setup, annotation quality, and task support.

Table 2.1 provides a compact comparison of major public 3D perception datasets used in autonomous driving. It summarizes their key differences in sensor configurations, dataset scale, annotated tasks, and unique features.

From the comparison in Table 2.1, nuScenes emerges as the most suitable dataset for this thesis. Several factors motivate this choice.

First, nuScenes supports all three perception tasks within a single dataset, with annotations for 3D bounding boxes, consistent object IDs for tracking, and velocity and orientation attributes that enable short-term trajectory prediction. This makes it possible to train and evaluate both modular pipelines (e.g., detection $\rightarrow$ tracking $\rightarrow$ prediction) and fully integrated end-to-end systems using a unified data format and evaluation protocol.

Table 2.1: Compact Comparison of Major 3D Perception Datasets

Dataset (Year)	**Sensors**	**Size / Scale**	**Core Tasks**	**Key Highlights**
KITTI (2012)	64-beam LiDAR, 4 cams	22 sequences	Detection, Tracking	First 3D benchmark; limited scenes (single city, daylight)
nuScenes (2019)	32-beam LiDAR, 6 cams, 5 radars	1k scenes, 1.4M boxes	Detection, Tracking, Prediction	Full sensor suite; rich metadata; Boston & Singapore
Waymo (2020)	5 LiDARs, 5 cams	390k frames	Detection, Tracking, Prediction	Dense data; multiple US cities; high diversity
Argoverse 1 (2019)	2×32-beam LiDAR, 9 cams	11k frames	Detection, Tracking, Forecasting	HD maps; map-aware prediction
Argoverse 2 (2021)	64-beam LiDAR, 12 cams	1k scenarios	Detection, Tracking, Prediction	Multi-city; updated sensor setup; diverse agents
Lyft L5 (2019)	3 LiDARs, 7 cams	55k frames	Detection (+Track IDs)	Kaggle challenge; limited geography
ApolloScape (2018)	64-beam LiDAR, cams	12k labeled frames	Detection, Tracking	Dense Beijing traffic; some trajectory labels
PandaSet (2020)	Mech + solid LiDARs, 6 cams	20k LiDAR sweeps	Detection, Segmentation	Sensor fusion focus; point-wise labels
H3D (2019)	64-beam LiDAR	27k frames	Detection, Tracking	Dense urban scenes; high crowd density
ONCE (2021)	32-beam LiDAR, 7 cams	1M frames	Detection (semi-supervised)	Massive scale; partial labels; self-supervised potential
A*3D (2020)	64-beam LiDAR, cams	39k frames	Detection (7 classes)	Night & rain conditions focus

Second, the sensor setup reflects real-world AV constraints. It uses a single 32-beam LiDAR providing full 360° horizontal field-of-view at 20 Hz. While this results in sparser point clouds compared to multi-LiDAR rigs like Waymo's, it represents a more realistic, cost-effective AV configuration. The sparser data also encourages model robustness and efficiency, particularly important in tracking and prediction tasks.

Third, nuScenes offers rich and reliable annotations. Every keyframe (at 2 Hz) is labeled with 3D bounding boxes for 10 object classes, including track IDs and motion attributes. These labels are cross-verified using LiDAR, radar, and camera inputs, resulting in high annotation quality. The annotated object velocities and motion status are especially valuable for training predictive models that learn kinematic behavior, not just spatial positions.

Additionally, the dataset spans diverse driving conditions—urban and highway, day and night, dry and rainy—across two cities (Boston and Singapore). This variation improves model generalization and reflects real-world deployment conditions, which is crucial when evaluating perception performance in both typical and challenging scenarios.

NuScenes further provides a comprehensive devkit with standardized metrics (mAP, NDS, AMOTA, etc.), which is essential for reproducibility and fair benchmarking. Its established detection challenge[1] and growing community use for tracking and short-term forecasting make it a strong foundation for perception research.

Finally, compared to older datasets like KITTI (limited in diversity and sequence length) and other datasets like Lyft or ApolloScape (which either lack map data, sufficient scale, or active benchmarks), nuScenes provides a well-rounded, modern, and actively supported resource. It enables the development and fair comparison of both modular and joint perception models, making it the ideal choice for this study.

2.2 Evaluation Metrics for LiDAR-based Perception on nuScenes

In this section, the evaluation metrics used to assess each stage of the LiDAR-based perception pipeline on the nuScenes dataset are described. The metrics cover 3D object detection, multi-object tracking, and trajectory prediction, following the official nuScenes evaluation protocol and common practices in autonomous driving research. Definitions, mathematical formulations, and the rationale for each metric are provided, along with an explanation of why these metrics are widely accepted. Where relevant, alternative metrics are also mentioned, including reasons for their exclusion from this study.

2.2.1 Stage 1: Detection

Detection stage evaluates how well the system detects objects in 3D. NuScenes defines detection metrics that emphasize both detection accuracy and the quality of the estimated object properties (location, size, orientation, etc.). Two primary metrics are used:

Mean Average Precision (mAP) in 3D Detection

Mean Average Precision (mAP) is the principal metric for object detection, extended to 3D. It measures detection accuracy by computing the average precision over recall for each class, and then averaging across classes. In nuScenes, a detection is considered a True Positive if the predicted 3D bounding box is sufficiently close to a ground-truth box in the horizontal plane, using a 2D center distance threshold on the ground plane rather than the typical IoU (intersection-over-union) criterion. This choice decouples the metric from object size and orientation, which is important because for small objects (like pedestrians or bicycles) even a small localization error can yield zero IoU, unfairly penalizing methods that localize correctly but slightly off in size/orientation. By using center distance (e.g. within 0.5, 1, 2, or 4 meters) to define matches, nuScenes mAP focuses on position accuracy and is more robust to small size/orientation errors.

The mAP is calculated by first computing the Average Precision (AP) for each class at multiple distance thresholds, then averaging. Specifically, AP is the area under the precision-recall curve (considering only the region where precision and recall are above 10% to avoid noise). NuScenes evaluates AP at four distance thresholds $D = 0.5, 1, 2, 4$ meters, and for each class c it obtains $\mathrm{AP}_{c,d}$ at each threshold d. The mean Average Precision is then obtained by averaging over all classes C and thresholds D:

$$\mathrm{mAP} = \frac{1}{|\mathcal{C}| \cdot |\mathcal{D}|} \sum_{c \in \mathcal{C}} \sum_{d \in \mathcal{D}} \mathrm{AP}_{c,d},$$

as defined in the nuScenes documentation[1]. This yields a single number (between 0 and 1, often reported as a percentage) summarizing detection performance across object categories and a range of matching tolerances. mAP is widely used in vision benchmarks because it provides a balanced evaluation of detection performance across different recall levels and object classes, without being dominated by a particular operating point. In nuScenes, using distance-based matching in mAP makes the metric better suited for 3D detection in autonomous driving contexts, where localization is paramount. (For comparison, the KITTI benchmark used IoU-based mAP with fixed IoU=0.7 for cars, etc., but nuScenes' center-distance mAP is more suitable given the dataset's diversity in object sizes and the inclusion of attributes like velocity)

nuScenes Detection Score (NDS)

While mAP captures how many objects are detected correctly, it does not capture how well certain properties are estimated (e.g. precise localization, orientation, velocity, attribute classification). The nuScenes Detection Score (NDS) is a composite metric introduced by nuscenes official paper[1] to provide a more holistic evaluation of 3D detection quality on their dataset. NDS is essentially a weighted average of the mAP and several True Positive (TP) error metrics that evaluate the quality of each detection. It condenses six metrics into a single score:

- mAP – the mean Average Precision as defined above.
- Mean Translation Error (mATE) – average 3D localization error, measured as the Euclidean distance between the predicted and ground-truth box centers (in meters).

- Mean Scale Error (mASE) – average bounding box scale error, defined via 3D IoU after aligning orientation and position (expressed as $1-\text{IoU}$, a value in $[0,1]$).
- Mean Orientation Error (mAOE) – average orientation error, measured as the smallest yaw angle difference between prediction and truth (in radians).
- Mean Velocity Error (mAVE) – average velocity magnitude error, computed as the L2 norm of the velocity difference in the ground plane (m/s).
- Mean Attribute Error (mAAE) – average attribute classification error, defined as $1-\text{accuracy}$ for predicting object attributes (e.g. whether a vehicle is parked or moving).

Each of these five error metrics (excluding mAP) is computed only on true positive detections (predictions matched to a ground truth). For instance, mATE is the mean translation error over all matched detections (averaged over classes), mAOE is the mean orientation error, etc. They are first computed per class and then averaged over classes to get a mean TP error (notated as mTP) for each metric. NDS combines mAP with these error metrics by converting each error into a score component. In particular, each error metric is normalized to a [0,1] scale by 1 - min(1, error) (so that lower errors yield higher scores, capped at 1.0). The official formulation of NDS is given by nuscenes documentation[1] as:

$$\text{NDS} = \frac{1}{10}\left(5\cdot\text{mAP} + \sum_{\text{mTP}\in TP}(1-\min(1,\text{mTP}))\right)$$

where *TP* the set of the five mean True Positive metrics.

This formula assigns **50% of the weight to mAP** and **50% to the combined TP quality metrics** (each of the five error metrics effectively contributing 10% of the score). In other words, half of NDS reflects how many objects are detected (coverage), and the other half reflects how accurate each detection is in terms of its box parameters, velocity, and attributes[1]. NDS is widely used in the nuScenes benchmark[1] as the primary ranking metric because it provides a single scalar that rewards not just detection but also precise state estimation. This addresses the limitation that “mAP alone cannot capture all aspects of the nuScenes detection task, like velocity and attribute estimation”. For example, a detector that finds many objects but with poor localization or velocity estimates will have a lower NDS, even if its mAP is high. Conversely, a method with slightly lower mAP but very accurate boxes and velocities can achieve a high NDS. This encourages balanced performance. In summary, mAP and NDS together give a comprehensive view: mAP focuses on object presence and NDS on overall detection quality. Other detection benchmarks use IoU-based AP or class-specific AP (e.g. KITTI[2]) and sometimes track orientation/attribute errors separately. The evaluation adhered to nuScenes standards to ensure consistency. Metrics such as Precision and Recall at fixed thresholds were not used, as mean Average Precision (mAP) already provides a summary of performance across all thresholds. Additionally, while IoU is common in 2D/3D vision, nuScenes found that IoU-based matching would penalize certain classes unfairly[1], so the distance-based approach was preferred in our evaluation.

2.2.2 Stage 2: Tracking

In the tracking stage, the goal is to link detections over time into trajectories (tracks) and maintain consistent identities for objects. Multi-object tracking (MOT) performance is evaluated using metrics that extend the classic CLEAR MOT[14] metrics to account for varying detection confidence. All tracking metrics are computed for each object class and then averaged over classes (nuScenes defines tracking metrics per class, then reports the macro-average)[1]. The primary metrics are Average Multi-Object Tracking Accuracy (AMOTA) and Average Multi-Object Tracking Precision (AMOTP), along with the count of Identity Switches

(IDS). These metrics are derived from the MOT metrics MOTA, MOTP, and ID Switch counts, but are averaged over different recall thresholds to provide a more robust evaluation.

Average Multi-Object Tracking Accuracy (AMOTA)

AMOTA is the main tracking metric on nuScenes, summarizing overall tracking accuracy. It is an extension of the classic CLEAR MOTA (Multi-Object Tracking Accuracy)[14], which is defined (for a given set of tracker outputs and ground truth) as:

$$\text{MOTA} = 1 - \frac{\text{FN} + \text{FP} + \text{IDS}}{\text{GT}}.$$

This formula essentially measures the fraction of object instances correctly tracked, by penalizing any errors: false negatives (missed ground-truth objects), false positives (spurious tracker outputs), and identity switches (ID reassignments), normalized by the total number of ground-truth objects (GT). A perfect tracker with no misses, no false alarms, and no ID errors would have MOTA = 1.0 (100%), whereas any error reduces the score. However, a known issue with MOTA is that it does not consider confidence scores of detections – it is computed at a particular threshold for deciding matches, so it can be overly harsh (e.g., if a tracker is tuned to be high-recall, it may incur many false positives, hurting MOTA). In nuScenes, due to the difficulty of the dataset, a single operating point can yield a low or zero MOTA (especially if recall is low). Average MOTA (AMOTA) addresses this by evaluating tracking over a range of recall levels, analogous to how Average Precision works for detection. Instead of reporting a single MOTA at a fixed threshold, nuScenes computes MOTA at various recall thresholds and averages them. At each recall level r, a modified MOTA score is computed, sometimes denoted sMOTAR (scaled MOTA at recall r). The formulation (from Weng et al. 2020[15], adopted in nuScenes) is:

$$\text{sMOTAR}(r) = \max\left(0,\ 1 - \frac{\text{IDS}_r + \text{FP}_r + \text{FN}_r - (1-r)\cdot\text{GT}}{r\cdot\text{GT}}\right)$$

This scaled formulation ensures that sMOTAR ranges from 0 to 1 for all recall values (preventing negative values in cases of very low recall). The Average MOTA (AMOTA) is then computed by averaging sMOTA(r) over a set of discrete recall thresholds $R = \{r_1, r_2, \ldots, r_n\}$ covering the range from 0.1 to 1.0 (nuScenes uses 40 recall points). Formally:

$$\text{AMOTA} = \frac{1}{|R|}\sum_{r\in R}\text{sMOTA}_r$$

In practice, this corresponds to integrating the MOTA-versus-recall curve and reporting the average value. AMOTA yields a more stable ranking of trackers than a single MOTA value, as it accounts for the tracker's performance across operating points. NuScenes emphasizes AMOTA as the primary tracking metric [1], using it to rank the official test set leaderboard. Although traditional MOTA is still reported, it can be highly sensitive to detection quality, especially in challenging scenarios with sparse sensors and dense scenes which may lead to lower values in weaker submissions. By averaging over multiple recall thresholds, AMOTA provides a more robust evaluation of tracking performance, rewarding methods that maintain a good trade-off between false positives, false negatives, and ID switches. Since sMOTA includes identity switches in its error term, AMOTA is sensitive to both detection accuracy and identity consistency, making it a comprehensive single-number summary of multi-object tracking performance.

Average Multi-Object Tracking Precision (AMOTP)

Whereas AMOTA focuses on accuracy (accounting for misses and ID mistakes), AMOTP evaluates the localization precision of the tracking. It is also derived from the CLEAR MOTP[14], which measures how close the tracker's reported object positions are to the true positions, on average, for the matches that are made. In classic terms, MOTP is defined as the total positioning error for matched objects over all frames, divided by the number of matches. For example, MOTP can be computed as the average IoU between matched predicted and ground-truth boxes (higher is better), or equivalently as an average distance error (lower is better) if using a distance-based matching criterion. NuScenes uses a distance-based matching, so MOTP here can be interpreted as the average 2D center distance between matched trajectories and ground truth (lower MOTP error means higher precision in localization). AMOTP extends this by averaging across recall thresholds similarly to AMOTA. The official nuScenes tracking evaluation provides AMOTP as the average MOTP over the same recall sample set used for AMOTA. In nuScenes tracking results, AMOTP is often reported alongside AMOTA to diagnose tracker performance: AMOTA tells how many objects were tracked correctly (accounting for errors), whereas AMOTP tells how well the tracker localized the objects it did track. By averaging over recall, AMOTP ensures that the precision is not measured at just one operating threshold but across the spectrum. This prevents a tracker from optimizing a single threshold for best precision while ignoring recall. Why these metrics: Both AMOTA and AMOTP were proposed by Weng et al. (2020)[15] and adopted by nuScenes to create a fair evaluation that accounts for detection confidence. They are now widely accepted in 3D MOT benchmarks (nuScenes, Waymo, etc.) as they provide a more comprehensive picture than the single-threshold MOTA/MOTP. In particular, AMOTA is the primary ranking metric for nuScenes tracking, because it correlates better with overall tracking success in challenging scenarios. The nuScenes tracking evaluation also includes traditional tracking metrics for reference, such as raw MOTA, MOTP, and raw counts of errors including false positives (FP), false negatives (FN), and identity switches (IDS). However, the analysis in this work focuses on AMOTA and AMOTP, as these provide a more comprehensive and robust evaluation.

Identity Switches (IDS)

An Identity Switch (ID switch) occurs when a tracker mistakenly assigns a new identity to an object that was previously being tracked under a different ID, or merges two tracks into one, essentially, any event where the persistent ID of a ground-truth object changes during tracking. In practical terms, if object A is being tracked as ID 5 and later the same object is suddenly labeled as ID 8, that is an ID switch. Likewise, if two objects swap IDs, two ID switch events are counted. ID switches indicate failures in maintaining consistent trajectories and are detrimental because they break the continuity of object tracks. The total number of ID switches (often denoted 'IDS') is counted over the evaluation. This is a raw count metric (with lower being better). It directly reflects the tracker's ability to keep consistent labels for objects over time. Fewer ID switches means the tracker is better at re-associating detections frame-to-frame, even through occlusions or close interactions. The evaluation reports the total IDS for each class and overall, although IDS is also implicitly accounted for in AMOTA (as discussed, each ID switch contributes to the MOTA error count at the moment it occurs). Why is IDS important? In an autonomous driving context, maintaining identity is critical for behavior understanding (e.g., knowing that the car observed 5 seconds ago is the same one about to cross our path now). A tracker with many ID switches might detect all objects but constantly confuse which is which, this would degrade the usefulness of the tracking output for prediction or planning. Therefore, IDS aree highlighted as a key consistency metric too. Reporting the number of ID switches is standard in MOT evaluations (CLEAR MOT metrics[14]) and is part of the nuScenes tracking metrics output[1]. A low IDS count usually correlates with a high-quality data association algorithm in the tracker. By using these 3 nuScenes metrics (AMOTA, AMOTP, IDS), it's ensured that evaluation for tracking is comparable to published results on that dataset.

2.2.3 Stage 3: Prediction

The prediction stage in autonomous driving perception involves forecasting the future trajectories of dynamic agents such as vehicles and pedestrians over a given time horizon. To evaluate the accuracy of these predicted trajectories, several standard metrics are commonly used in the literature[1, 3, 6, 7, 16]. Among the most widely adopted are the Average Displacement Error (ADE), Final Displacement Error (FDE), and Miss Rate (MR), which are also the metrics used in this thesis for evaluating prediction performance.

These metrics are typically computed over prediction horizons such as 4 seconds (8 time steps) or 6 seconds (12 time steps), especially in datasets like nuScenes, which uses a 0.5-second interval between frames. Evaluating prediction performance at multiple horizons helps distinguish between short-term and longer-term forecasting capabilities, as prediction uncertainty generally increases with time. The metrics are often reported separately for each horizon using notations such as ADE@4s, FDE@6s, or equivalently $\text{ADE}_{4\text{s}}$, $\text{FDE}_{6\text{s}}$, etc.

Average Displacement Error (ADE)

Average Displacement Error (ADE) is a common metric that measures the average distance between the predicted trajectory and the ground truth trajectory over all time steps. Mathematically, for a given object, if we denote the ground truth future positions as (x_t, y_t) and the predicted positions as $(\hat{x}_t, \hat{y}_t)$ for time steps $t = 1, 2, \ldots, T$ (where $T = 8$ for 4s or $T = 12$ for 6s), the ADE is:

$$\text{ADE} = \frac{1}{T} \sum_{t=1}^{T} \sqrt{(\hat{x}_t - x_t)^2 + (\hat{y}_t - y_t)^2}$$

ADE, it's simply the mean Euclidean distance between the predicted and true positions, averaged over the whole forecast horizon for that object. ADE gives a sense of the overall quality of the trajectory prediction, it penalizes prediction drift at any point in the sequence, since all time steps contribute equally to the error. A lower ADE (in meters) is better. For instance, ADE = 0.5 m (over 4s) would indicate that on average the predicted position is 0.5 m away from the true position at each time step, which is quite accurate for short-term prediction. ADE is widely used in the literature as a basic measure of prediction accuracy. Although ADE provides a strong measure of overall trajectory quality, it can miss important cases, such as when a prediction follows the ground truth closely but ends at the wrong location, or when it deviates early and only aligns near the end. FDE addresses these by specifically evaluating the final position, making it a necessary complement to ADE for assessing endpoint accuracy.

Final Displacement Error (FDE)

Final Displacement Error (FDE) measures the distance between the predicted endpoint of the trajectory and the ground-truth endpoint after the full horizon. It is essentially the error at the last time step of the prediction. Using the same notation as above, if $(\hat{x}_T, \hat{y}_T)$ is the predicted position at the final time step ($T = 8$ or 12) and (x_T, y_T) is the true position, then:

$$\text{FDE} = \sqrt{(\hat{x}_T - x_T)^2 + (\hat{y}_T - y_T)^2}$$

where lower values indicate better predictions, for both ADE and FDE. This is computed per object, and then averaged across all objects to report an overall FDE (for different time horizons separately). FDE directly evaluates how close the prediction was at the end of the trajectory, which is often critical for planning (e.g., did we correctly predict where the pedestrian will be 6 seconds from now?). While ADE gives the average error over time, FDE highlights the final position accuracy. It is possible for a predictor to have a low ADE but a relatively high FDE (if it was generally accurate but drifted toward the end), or vice versa. Therefore, both are reported. Both ADE and FDE are widely accepted metrics in trajectory forecasting benchmarks (nuScenes prediction challenge, Argoverse, etc.)[1, 3, 6, 7]. They are easy to interpret and directly measure accuracy in meters. For this reason, these metrics were selected for use in this work, to quantify prediction performance at 4s and 6s.

Miss Rate (MR)

Miss Rate (MR) is a metric that captures the frequency of large errors in prediction. Miss Rate is defined as the percentage of trajectories for which the final prediction is significantly off from the ground truth. Specifically, a displacement threshold (commonly 2.0 meters) is chosen and consider a prediction a "miss" if the FDE (final error) exceeds this threshold. Conversely, if the final point is within 2 m of the true endpoint, it is a hit (or successful prediction). Miss Rate is then calculated as:

$$\text{MR} = \frac{\text{Number of trajectories with FDE} > 2.0\,\text{m}}{\text{Total number of trajectories}} \times 100\%$$

This is equivalent to saying “if an endpoint deviation exceeds d (e.g. 2 m), that prediction is a miss”[17], and MR is the fraction of all predictions that are misses[17]. MR is typically reported as a percentage, e.g., MR = 20% means 20% of predicted trajectories missed the target by more than 2 m. Miss Rate (MR) generally increases with longer prediction horizons, as forecasting further into the future becomes more uncertain. For example, if 10% of predicted trajectories deviate by more than 2 meters at 4 seconds ($\text{MR}_{4\text{s}} = 10\%$), this may rise to 25% at 6 seconds ($\text{MR}_{6\text{s}} = 25\%$). This illustrates the growing difficulty of maintaining accurate predictions over extended time horizons.

ADE/FDE give average magnitudes of error, but MR gives a sense of how often the prediction is unacceptably wrong by a certain standard. In autonomous driving, a predictor might have a low average error but still occasionally make a very large error, MR captures this by focusing on the tails of the error distribution. mWhile alternative metrics exist, such as minADE/minFDE for multi-modal predictions or task-specific metrics like collision rate and goal attainment, we focus on standard single-trajectory metrics: ADE, FDE, and Miss Rate. These are widely used in major benchmarks (e.g., nuScenes, Argoverse, Waymo)[1, 3, 6, 7, 16], and it’s found that they are sufficient to provide a clear, representative evaluation of prediction accuracy. Using well-established metrics also ensures easier comparison and interpretability of results.

In summary, the evaluation methodology covers detection, tracking, and prediction with appropriate metrics for each. Widely-used metrics from the autonomous driving field are adhered to in this evaluation: mAP and NDS for 3D detection quality, AMOTA and AMOTP (with IDS) for multi-object tracking performance, and ADE, FDE, Miss Rate for trajectory prediction accuracy. These metrics collectively give a comprehensive view of the perception pipeline’s performance and align with the nuScenes benchmark, ensuring that results are meaningful and comparable to existing state-of-the-art results in the literature.

2.3 Object Detection in Autonomous Systems

Recent years have seen rapid progress in LiDAR-based 3D object detection for autonomous driving. From 2021 to 2025, numerous models have pushed the state of the art on benchmarks like the nuScenes dataset[1]. This review focuses on key architectural innovations, including voxel-based, point-based, and Bird’s-Eye View

(BEV) approaches, as well as benchmark performances (mAP and NDS), training strategies (modular vs. end-to-end), and the evolving role of the CenterPoint model as a reference baseline.

2.3.1 Voxel-Based Approaches

Voxel-based methods are widely used in 3D object detection due to their balance of accuracy and efficiency. Early models such as VoxelNet[18] and SECOND[19] divided the 3D space into voxels and applied sparse 3D convolutional backbones. These models still used dense anchor-based heads, which required unnecessary computation in empty regions. A major improvement came with CenterPoint[20], which introduced a center-based and anchor-free detection head working on the Bird's-Eye View (BEV) plane. Its backbone produced BEV feature maps, and object centers were detected using a keypoint head. This simplified the pipeline and achieved strong performance on the nuScenes benchmark, making CenterPoint a widely used baseline.

Further improvements were made by methods like PillarNet[21], which used pillar features similar to PointPillars[22], combined with a more powerful backbone and detection head. PillarNet outperformed CenterPoint on nuScenes with higher mAP and NDS while still running in real time. VoxelNeXt[23] removed the need for dense BEV heatmaps and anchors entirely. Instead, it performed detection directly from sparse voxel features using only non-empty locations. This made the model much faster while still matching or exceeding the accuracy of CenterPoint.

DSVT[24] replaced the convolutional backbone with a sparse transformer architecture. It introduced rotated window-based sparse self-attention to better capture long-range dependencies in sparse point clouds. DSVT ran in real time and achieved state-of-the-art performance on the nuScenes test set. More recently, LION[25] proposed a linear RNN-based backbone that grouped voxels and applied efficient operations across them. LION achieved the highest mAP and NDS scores reported on nuScenes while being more efficient than transformer-based models.

These voxel-based methods all share a focus on simplifying the detection pipeline, using sparse operations and stronger backbones like CNNs, transformers, or RNNs, to improve performance and efficiency.

2.3.2 Point-Based Approaches

Point-based detectors process raw 3D points directly instead of converting them into voxels. Models like PointRCNN[26] and 3DSSD[27] used point clusters to regress bounding boxes in a single stage. These models avoid quantization errors and better preserve detailed geometry. However, they often struggled on large-scale outdoor datasets due to higher computational cost. For example, 3DSSD achieved a lower NDS than voxel-based models on nuScenes. One main limitation is the high cost of point-based operations such as set abstraction when applied to large numbers of points.

A hybrid approach was introduced by PV-RCNN++[28], which combined voxel backbones for region proposal with point-based refinement. While more accurate than early point-based models, this pipeline was more complex and less efficient. In recent years, research focus has moved toward voxelized methods for large-scale benchmarks like nuScenes. Although fully point-based models are less common today, their design ideas still influence newer voxel-based models, particularly in sparse feature handling and sampling strategies.

2.3.3 BEV-Based Fusion and Multi-Modal Approaches

Another important direction is the use of BEV (Bird's-Eye View) as a common space for combining features from LiDAR and other sensors. BEVFusion[29] is one of the most effective methods in this category. It extracts BEV features separately from LiDAR and from each camera using view transformers, then combines them into a shared BEV space. This approach improves detection, especially for object types where image appearance helps, such as motorcycles or bicycles. BEVFusion reached top results on the nuScenes leaderboard for multi-modal detection, with significant gains in both mAP and NDS.

TransFusion[30] also combined LiDAR and camera data using a transformer-based image branch while keeping a CenterPoint-style LiDAR backbone. It improved results slightly over LiDAR-only models. Some BEV-based models use only LiDAR input. For example, CenterPoint[20] and PillarNet[21] use BEV for their detection heads. Newer approaches such as EA-BEV[31] and UniAD[32] go even further by unifying detection, tracking, and prediction in a single BEV space. These end-to-end systems process sensor inputs and produce multi-task outputs within one architecture. Although they do not always outperform specialized detectors in detection-only tasks, they demonstrate a shift toward more integrated systems that cover multiple perception tasks.

Overall, BEV-based architectures have enabled stronger performance by offering a shared view of the scene that combines geometry from LiDAR and semantics from camera data. These models show that camera information can help LiDAR-based detection, particularly for small or visually distinct objects, and they continue to influence both single-task and multi-task model designs.

2.3.4 Benchmark Performance on nuScenes (2022–2025)

Table 2.2 compares several representative models on the nuScenes 3D detection benchmark. Both the mAP and NDS are listed as reported on the nuScenes test set in their respective papers. These results illustrate the performance leap from the CenterPoint era (2021) to the latest methods:

Table 2.2: Comparison of 3D object detection models on the nuScenes test set. All results (mAP and NDS) are reported from the respective original papers or official leaderboard entries[1].

Model (Year)	**Input**	**mAP (%)**	**NDS (%)**	**Notes**
CenterPoint (2021)	LiDAR-only	58.0	65.5	Baseline detector
PillarNet-34 (2022)	LiDAR-only	66.0	71.4	Sparse CNN, one-stage
VoxelNeXt (2023)	LiDAR-only	66.2	71.4	Fully sparse, no dense head
DSVT (2023)	LiDAR-only	68.4	72.7	Sparse Transformer
LION (2024)	LiDAR-only	69.8	73.9	RNN backbone, SOTA
BEVFusion (2022)	LiDAR+Camera	70.2	72.9	BEV fusion (multi-modal)
TransFusion (2022)	LiDAR+Camera	65.5	70.2	CenterPoint + camera input

CenterPoint serves as the yardstick for progress. While newer methods surpass it by up to 8 NDS points, it remains widely adopted and modular enough for extension. CenterPoint deserves special discussion as the reference model that many later works build upon or compare against. Introduced by Yin et al. in 2021[20], CenterPoint took the ideas of 2D center-based detection (CenterNet)[33] into 3D. It used a BEV heatmap to represent object centers, doing away with the complicated anchor generation of previous methods. CenterPoint also integrated velocity estimation and a simple tracking-by-detection scheme: each detected object is given a tracking ID by matching centers frame-to-frame, enabling 3D multi-object tracking. Upon release, CenterPoint topped the nuScenes leaderboard (with 65% NDS) and became open-source, leading to wide adoption. It was quickly incorporated into popular codebases like OpenPCDet[34] and MMDetection3D[35], making it a go-to baseline for academia and industry. The role of CenterPoint in subsequent research has been two-fold. First, it serves as a strong baseline to beat, virtually every new LiDAR detector paper evaluates against CenterPoint. For example, TransFusion (2022)[30] reported that adding camera data to CenterPoint modestly improved mAP/NDS (showing CenterPoint is already strong). PillarNet (2022)[21] explicitly compares multiple variants to CenterPoint, highlighting gains in both accuracy and speed.

Many researchers reuse CenterPoint's backbone or head and focus on innovating elsewhere. The CenterPoint framework is modular enough that one can plug in a new backbone (VoxelNeXt[23] did this), a new head (CenterFormer[36] did this), or a new fusion mechanism (TransFusion[30] did this), and still benefit from the robust center-based representation and training pipeline. Despite its strong legacy, more recent methods clearly outperform CenterPoint on nuScenes.

As seen in Table 2.2, CenterPoint's 58.0% mAP, 65.5 NDS is now surpassed by a lot of newer models, which reach into the 70s NDS. These improvements come from addressing CenterPoint's limitations, e.g. sparse feature utilization (CenterPoint's dense head wastes computation on empty cells), backbone receptive field (CenterPoint's CNN struggles with long range, which DSVT's global attention handles), and multi-task learning (CenterPoint did detection then separate tracking, whereas newer models jointly reason about objects over time).

Nonetheless, CenterPoint remains highly relevant. It is often the starting point for deploying a LiDAR detector due to its simplicity, efficiency, and well established performance across benchmarks. For these reasons, CenterPoint is adopted as the detector in this work, as it provides a strong and reliable baseline for evaluating downstream tracking and prediction modules.

2.4 Multi-Object Tracking (MOT)

LiDAR-based 3D multi-object tracking (MOT) is a critical component of autonomous driving perception. Given a sequence of LiDAR point clouds (often combined with camera data), the goal is to detect objects in 3D and maintain consistent identities over time. A good tracker maximizes AMOTA (higher is better) while minimizing AMOTP (position error; lower is better). Additionally, identity switches (IDS) are tracked as a measure of track consistency (fewer IDS is better).

Traditional 3D MOT follows a tracking-by-detection (TBD) paradigm: first run a 3D object detector on each frame, then associate detections across frames using a separate tracking module. This tracking module often relies on a motion model (e.g. constant velocity via a Kalman Filter[37]) to predict positions and a matching algorithm (e.g. Hungarian algorithm[38] solving a cost matrix of distances or IoU) for data association. Recent advances have explored learnable tracking models and even end-to-end approaches that jointly perform detection and tracking. This review covers the state-of-the-art in LiDAR-based 3D MOT on nuScenes, contrasting classic non-learnable pipelines with modern learnable models. Finally, PolyMOT and 3DMOTFormer are highlighted as two archetypes, one a strong modular baseline and the other a learanble tracker, to illustrate the current trade-offs between them in 3D MOT frameworks.

2.4.1 Traditional Tracking-by-Detection Methods (Non-Learnable)

Early 3D multi-object tracking methods used simple tracking-by-detection pipelines without learning. A common approach was to track each detection using a Kalman Filter and solve frame-to-frame associations with the Hungarian algorithm based on distance or IoU costs[38]. This baseline, often called AB3DMOT[41], became a standard for KITTI and nuScenes. However, performance was limited by the simplicity of the models and the quality of input detections.

Later systems improved these pipelines using better heuristics. Some models included appearance features from camera images to improve association, such as EagerMOT[42], while others refined the motion modeling and data association step. For example, Probabilistic 3D MOT[43] introduced Mahalanobis distance gating to account for uncertainty in tracking, which improved robustness. These improvements, although still using classical components, helped reduce identity switches and false matches.

PolyMOT[39] became a leading traditional tracker by combining these ideas into a well-designed framework. It used different motion models depending on object type and matched objects using different similarity metrics based on class, such as IoU for large vehicles or center distance for pedestrians. It included detection filtering, category-specific tuning, and a two-stage matching strategy. All parts were hand-tuned and no neural networks were used. With strong LiDAR detections as input, PolyMOT achieved 75.4 percent AMOTA on the nuScenes test set and kept identity switches very low[45]. It outperformed many learning-based models, including camera-based ones like CAMO-MOT[46]. Its performance remained stable across different detectors, showing that its modular design could generalize well.

Other strong traditional methods include Fast-Poly[40], which optimized PolyMOT for speed, and SimpleTrack[49], which reduced complexity by using Euclidean distance instead of IoU. SimpleTrack achieved about 65 to 66 percent AMOTA with a much simpler design. These results showed that well-designed tracking-by-detection pipelines can remain competitive even without learning. By the end of 2023, most top-performing trackers on the nuScenes leaderboard still followed the traditional tracking-by-detection approach.

2.4.2 Learnable and End-to-End MOT Models

To overcome the limits of hand-tuned trackers, recent work has introduced learning into the tracking process. In some models, tracking is still done by associating detections, but the association step is learned. In other cases, detection and tracking are done together in a single end-to-end model. The latter approach is sometimes called tracking-by-attention.

3DMOTFormer[48] is a leading example of learned association. It starts from input detections and replaces the matching step with a graph Transformer that learns to link detections with existing tracks. The model builds a bipartite graph between current detections and past tracks, encodes features like distance and direction, and classifies links using attention. To address the gap between training and inference, where real predictions contain errors, 3DMOTFormer uses online training. It steps through sequences using its own past predictions to simulate real use, teaching the model to handle mistakes. With CenterPoint detections as input[20], 3DMOTFormer reached 68.2 percent AMOTA on the nuScenes test set and showed strong generalization across different detectors. An overview of the 3DMOTFormer architecture is shown in Figure 2.1, which is recreated based on the original paper to support later visualization of a joint tracking-prediction pipeline.

Another learnable model, OGR3MOT[47], used graph neural networks to perform message passing between detections and tracks. It improved identity consistency by lowering ID switches, but the overall AMOTA remained close to traditional methods, showing that early learning-based trackers were not always better than hand-engineered ones.

In end-to-end approaches, the tracking is built into the detector. One early idea was CenterTrack[50], which predicted tracking offsets along with detections in 2D. This idea carried over to LiDAR-based methods like CenterPoint[20], where each detected object is given a velocity vector. Tracks are formed by matching predicted centers with detections in the next frame. This simple matching outperformed Kalman-based methods, showing that learned motion is more accurate than a generic constant-velocity model. CenterPoint tracking reached about 63.8 percent AMOTA on nuScenes test and showed that even simple learned motion helps.

More recent end-to-end models include Minkowski Tracker[51] and JDT3D[52]. Minkowski Tracker applied 4D sparse convolutions over both space and time to extract joint spatiotemporal features. It processed multiple point cloud frames together and produced detections with associated track IDs. It reached 69.8 percent AMOTA using only LiDAR input, a strong result for an end-to-end model, though it required heavy computation. JDT3D used a transformer model to represent each track as a query that persists over time. This allowed it to handle occlusions, but performance lagged behind tracking-by-detection models. It reached

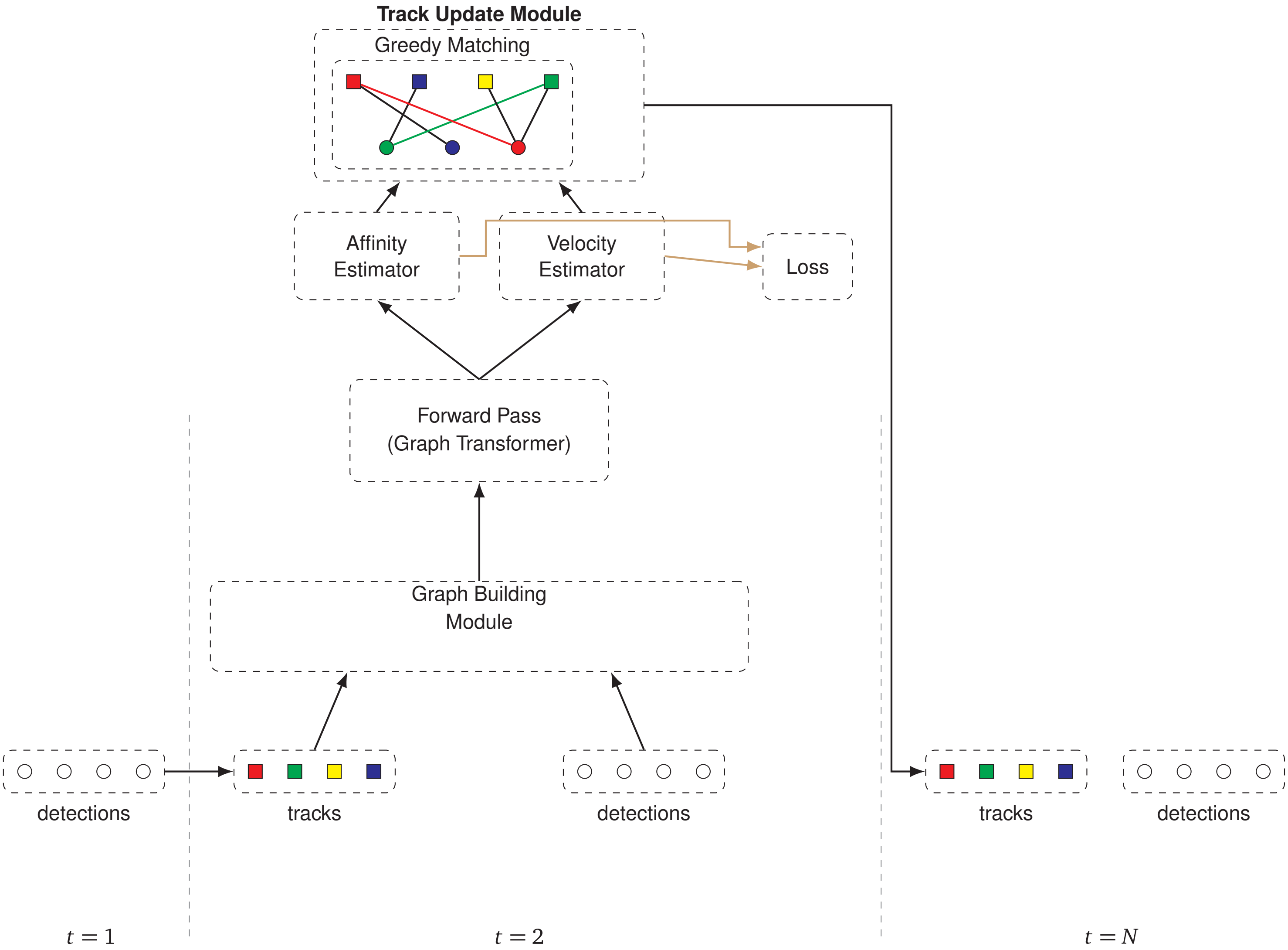


Figure 2.1: Recreated overview of the 3DMOTFormer tracking architecture from [48]. The original structure is redrawn here to support later extensions for illustrating joint tracking and prediction integration in our study.

57.4 percent AMOTA, which was below PolyMOT and other traditional methods. The main challenges were the low temporal resolution of nuScenes LiDAR and difficulty balancing detection and tracking loss during training.

In summary, learnable models like 3DMOTFormer show that with proper design and training, learned trackers can approach or match traditional ones. End-to-end tracking is promising but still struggles with sparse data and training complexity. As of now, the best results on nuScenes are still achieved by well-designed tracking-by-detection methods like PolyMOT and Fast-Poly.

2.4.3 NuScenes Tracking Benchmark: Performance Comparison

To concretely illustrate the landscape, Table 2.3 compares several representative 3D MOT methods on the nuScenes test set. A classic baseline is listed, top traditional methods (learnable or not), and recent learning-based models. The metrics shown are the official nuScenes AMOTA, AMOTP, and total IDS on the test split. All methods use the LiDAR sensor; some use additional camera inputs as noted. These numbers reflect the state-of-the-art progression on the nuScenes 3D MOT benchmark.

Table 2.3: Comparison of 3D multi-object tracking (MOT) methods on the nuScenes test set. All results (AMOTA, AMOTP, and ID switches) are reported from their respective papers or official benchmark entries.

Method (Year)	**Approach**	**Sensors**	**AMOTA ↑**	**AMOTP ↓**	**IDS ↓**
Tracking-by-Detection (TBD)					
CenterPoint Tracker (2021)	Det. + velocity match	LiDAR-only	63.8	0.637	760
SimpleTrack (2022)	Det. + simple matching	LiDAR-only	66.8	0.550	575
EagerMOT (2021)	Det. + multi-modal fusion	LiDAR+Camera	68.0	0.569	1156
OGR3MOT (2022)	Learned GNN assoc.	LiDAR-only	65.6	0.620	288
Poly-MOT (2023)	Modular TBD (no learning)	LiDAR-only	75.4	0.521	292
Fast-Poly (2023)	Modular TBD (no learning)	LiDAR-only	75.8	n/a	326
CAMO-MOT (2023)	Multi-modal optim.	LiDAR+Camera	75.3	0.527	324
3DMOTFormer (2023)	Learned Transformer	LiDAR-only	68.2	0.496	438
End-to-End Detection and Tracking					
Minkowski Tracker (2022)	End-to-end 4D CNN	LiDAR-only	69.8	n/a	325
JDT3D (2024)	End-to-end Transformer	LiDAR-only	57.4	0.837	254

For this study, PolyMOT is selected as the non-learnable tracker and 3DMOTFormer as the learnable counterpart to enable a clear comparison of how classical versus learned tracking approaches perform within a sequential detection–tracking–prediction (DTP) pipeline. PolyMOT offers strong, benchmark-leading performance using manually designed logic, making it a solid baseline for modular systems. On the other hand, 3DMOTFormer not only achieves competitive results through learned association but its learnable model architecture, allows to extend it for joint training with a prediction model. Both trackers are open source, well-documented, and compatible with the nuScenes dataset, which further supports reproducibility and integration into our pipeline.

2.5 Trajectory Prediction

The goal of trajectory prediction is to forecast the future positions of dynamic agents—such as vehicles or pedestrians—over a given time horizon based on past observations. Deep learning approaches have dramatically improved trajectory forecasting by leveraging data to learn complex agent behaviors, multi-agent interactions, and road context. Modern models take into account scene context (maps or LiDAR), social interactions between vehicles and other agents, and produce probabilistic, multi-modal predictions (multiple possible future trajectories with confidences). Below key families of state-of-the-art models are reviewed, from recurrent graph models to anchor-based and transformer-based approaches, highlighting how they handle LiDAR input, interactions, and uncertainty. Their performance is also compared on the large-scale nuScenes benchmark, and discuss training strategies and integration into planning.

2.5.1 RNN and Graph-Based Models (Trajectron++)

One of the early deep learning methods for motion forecasting used recurrent neural networks to model agent trajectories over time. In the context of autonomous driving, Trajectron[54] and its improved version Trajectron++[55] became widely used graph-based forecasting models. Trajectron++ represents the scene as a dynamic directed graph, where each node corresponds to an agent such as a vehicle or pedestrian, and edges represent their interactions. Each agent's motion history is encoded with an RNN, while a graph neural network handles interactions between agents. Trajectron++ can also include semantic map information, such as road layout, to provide context.

A key feature of Trajectron++ is its use of a conditional variational autoencoder to model future uncertainty. During training, the model learns a latent variable that captures future modes, such as turning or going straight. At inference, the model samples from this latent space to produce a set of diverse trajectories, each with an associated probability. This allows the model to estimate uncertainty and provide multiple future predictions.

Trajectron++ was designed to be modular. It takes as input the tracked states of agents and optionally a map, and outputs multi-modal future predictions. It does not operate on raw LiDAR, but integrates easily with detection and tracking systems that do. The model also supports conditional forecasting, where the future behavior of agents can be predicted based on a planned trajectory of the ego vehicle. This makes it useful for planning applications.

On the nuScenes dataset, Trajectron++ outperformed prior deterministic models such as CoverNet[56]. It achieved around 1.9 meters ADE and 4 to 5 meters FDE on the validation set with five predicted modes. The constant-velocity baseline, by comparison, had about 3.7 meters ADE and 9.1 meters FDE. The Miss Rate at 2 meters was around 70 percent, meaning that 30 percent of the predictions were within 2 meters of the ground truth. Although newer models have surpassed it, Trajectron++ remains a foundational approach for multi-agent, multi-modal forecasting. It introduced several important design ideas, including the use of latent variable modeling and graph-based interaction, which were adopted by later models such as WIMP[57].

Figure 2.2 provides an overview of the Trajectron++ architecture. It was recreated based on the original paper[55] and its open-source code, and illustrates the main components including agent-wise RNN encoders, a graph interaction module, the latent sampling process, and the decoder for generating multiple predicted trajectories.

2.5.2 Anchor and Goal-Based Models (CoverNet, TNT, Goal-LBP)

Some forecasting models avoid generating trajectories directly from scratch and instead use predefined trajectory sets or predicted goals to guide the output. This includes anchor-based and goal-conditioned methods, which aim to reduce errors and ensure realistic motion.

CoverNet[56] introduced an anchor-based approach where a large set of trajectories is precomputed from training data. The model then performs classification over this set, selecting the most likely future paths. This ensures that each predicted trajectory is physically feasible. Despite having over 2000 possible anchors, CoverNet performed well. On nuScenes, it reached around 3.9 meters ADE and 9.3 meters FDE using the single most probable trajectory. When allowing up to five predictions, the minADE improved to about 1.96

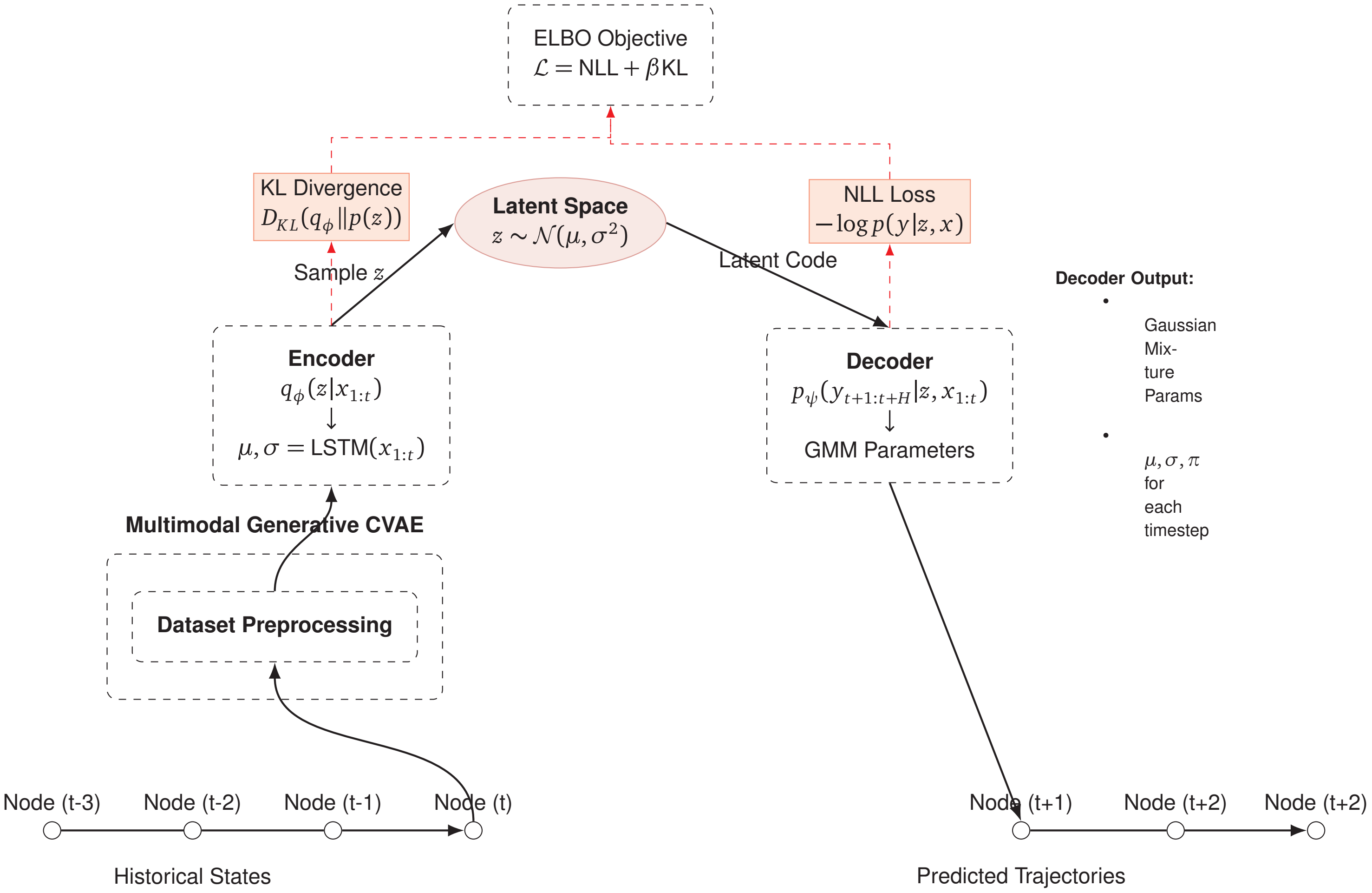


Figure 2.2: Recreated schematic of the Trajectron++ prediction architecture from [55]. This version was redrawn for clarity based on the original paper and open-source code, and is used in this thesis to support analysis and comparison with other forecasting frameworks.

meters, and the Miss Rate at 2 meters dropped to around 67 percent. The main limitation is the need to construct and store a large anchor set.

Other models such as MultiPath[58] and MTP[59] follow similar ideas but introduce additional flexibility. These models use a fixed set of endpoints or anchor trajectories, then predict small offsets from these to improve accuracy. Each predicted mode is scored, allowing the model to prioritize the most likely futures. On nuScenes, MultiPath achieved around 3.45 meters ADE and 3.6 meters FDE with five modes, which was better than earlier LSTM models. The use of anchors or fixed endpoints makes the model interpretable and efficient.

Goal-conditioned prediction focuses on forecasting destination points. One such method is TNT[60], which first predicts a set of likely goal points, then generates complete trajectories toward those goals. On the Argoverse dataset, TNT reached about 0.73 meters minADE using six trajectory predictions and correctly predicted the final goal location 73 percent of the time within a 2-meter radius. Although TNT was not evaluated on nuScenes, it remains a strong goal-based model.

Goal-LBP[61] is a more recent goal-conditioned model that improves prediction by first identifying short-term behaviors, then refining trajectories toward high-quality goals. It combines intermediate maneuver understanding with destination prediction. On nuScenes, it achieved around 1.02 meters ADE and 9.2 meters FDE using five predicted trajectories. Its Miss Rate at 2 meters was about 32 percent, making it one of the top performers on the benchmark as of 2023. By using both goal candidates and local behavior patterns, Goal-LBP reduced prediction errors significantly compared to earlier methods.

These anchor and goal-based models show that using structured prediction spaces, such as a set of anchors or forecasted goals, helps generate accurate and realistic multi-modal trajectories. They also tend to be more interpretable and efficient than free-form generative models.

2.5.3 Attention and Transformer-Based Models (WIMP, Transformer nets)

Models using attention and transformer-based architectures have been developed to better capture complex interactions among agents and to make use of contextual information such as road layout. These models do not rely on fixed graph or convolution structures, and instead use attention mechanisms to dynamically focus on relevant elements in the scene.

WIMP (What-If Motion Prediction)[57] introduced a graph-attention architecture that represents agents and map elements like lanes as nodes, and uses attention layers to model interactions. WIMP can also condition predictions on hypothetical future paths of the ego vehicle, which allows it to answer counterfactual queries. It achieved strong performance on nuScenes with about 1.11 meters minADE using 10 predicted modes, and was particularly effective in complex intersections by combining social and map context. Compared to earlier models such as Trajectron++[55] and CoverNet[56], WIMP achieved lower displacement error and miss rate. Architecturally, WIMP still uses RNNs but its attention layers set the stage for later transformer-based models.

Transformer networks use self-attention to jointly encode all agents and map elements. This makes them well suited for multi-agent forecasting, especially in dense urban scenes. Models like LaneGCN[62], VectorNet[63], AgentFormer[64], SceneTransformer[65], and THOMAS[66] applied various combinations of vectorized map encoding and transformer layers. AgentFormer, for instance, treats each agent as a token and models their state over time with a variational transformer. THOMAS achieved around 1.04 meters minADE on nuScenes with 10 modes, showing the value of jointly modeling agents and maps.

Among the most advanced is MTR (Multi-Modal Motion Transformer)[67], which uses a large transformer model to predict multiple trajectories with probabilities. MTR uses two decoding stages, starting with coarse proposals and refining them. Trained on large-scale datasets through UniTraj[68], it reached top results on nuScenes with around 0.96 meters minADE using five modes and about 5.5 meters minFDE. These results show the benefit of scaling both model size and data in combination with attention mechanisms.

SemanticFormer[69] further builds a semantic scene graph including relationships like yielding or occupancy and processes this graph using a hybrid GNN-transformer model. It achieved strong accuracy on nuScenes and highlighted the benefit of understanding interactions beyond simple geometry. Other models like LaPred and LaFormer also adopted transformer-based designs and achieved competitive results. LaFormer reached approximately 1.19 meters minADE and 48 percent miss rate, showing how large-scale attention models can provide accurate and diverse multi-modal predictions.

Attention-based models are highly effective in capturing multi-agent behavior and integrating map features. These models often use vectorized inputs derived from LiDAR detections and map data, and represent the scene in a way that attention layers can reason about it globally. Models like SemanticFormer go further by explicitly modeling relationships between agents. Overall, these models have substantially improved performance on benchmarks like nuScenes from 2021 to 2024.

2.5.4 Occupancy Grid and Joint Prediction Methods (CASPNet++)

Another approach to prediction is based on forecasting future occupancy in the scene, rather than predicting separate trajectories for each agent. These methods predict spatiotemporal grids that represent how the space is likely to be occupied over time. This allows modeling uncertainty jointly for all agents and is especially useful in crowded or occluded situations.

CASPNet++[72] is a model designed for nuScenes that uses an occupancy grid representation together with an instance-level trajectory head. It builds a voxel grid of the scene and predicts how the occupancy changes in time. From this, it can extract trajectories for specific agents of interest. The model combines data from multiple sources including HD maps, radar detections, and LiDAR segmentation. In particular, point-wise semantic segmentation of LiDAR provides detailed shape and motion cues, which help forecast motion, even for partially observed agents.

CASPNet++ reached strong performance on nuScenes, reporting around 1.16 meters ADE and ranking among the top entries on the leaderboard. It is also efficient enough for real-time use. Occupancy-based prediction naturally supports multi-modal outputs, since different possible futures appear as separate regions in the occupancy map. This makes it easier to plan around all likely outcomes, especially in scenes with many agents, compared to working with a fixed list of trajectory hypotheses.

This approach treats prediction as a form of dynamic scene understanding. It scales well with the number of agents and integrates naturally with downstream planning, since planners can avoid regions of high predicted occupancy. The main challenge is that training these models can be complex and explicit control over individual agent paths may be limited, though models like CASPNet++ address this with instance-level outputs. Overall, occupancy-based models offer a powerful and unified way to reason about future motion and uncertainty in dense traffic scenarios.

2.5.5 Performance on nuScenes: Benchmark Metrics and Model Comparison

The nuScenes prediction benchmark evaluates models on a 6-second horizon (with 2 seconds of context). Models can output up to 25 trajectories (modes) with confidences for each agent, though most methods use far fewer (5 or 10). Key metrics include:

Average Displacement Error (ADE): the mean Euclidean distance between predicted and ground-truth positions, averaged over the whole forecast horizon. MinADE_K is often used, the ADE of the single best (closest) out of K predicted trajectories. This rewards multi-modal predictors that put one trajectory close to the truth. Final Displacement Error (FDE): the distance between the predicted end-point after 6s and the true end-point. MinFDE_K is likewise used for the best of K. Miss Rate (MR)@2m: the percentage of cases where no predicted trajectory comes within 2.0 meters of the true trajectory at every timestep. (Equivalently, if the best trajectory still strays >2 m at some point, it's a miss.) In practice, many use a simpler definition: whether the final point is within 2 m, but nuScenes' official metric considers the whole trajectory tolerance. Lower Miss Rate is better (0% means every scenario had an adequate prediction).

Table 2.4 summarizes the performance of representative trajectory prediction models on the nuScenes test set, comparing both classical and state-of-the-art deep learning approaches. All metrics are for a 6-second prediction horizon and are reported as stated in their respective papers.

Table 2.4: Performance comparison of selected trajectory prediction models on the nuScenes validation set (6-second horizon). Metrics include minimum Average Displacement Error (minADE_5), minimum Final Displacement Error (minFDE_1), and Miss Rate@2m with top-5 predicted trajectories. All values are reported from the models' original publications.

Model (Year)	**minADE_5**	**minFDE_1**	**MissRateTopK_2_5**
LaPred (2021)	1.47	8.37	53%
LaFormer (2023)	1.19	6.95	48%
CoverNet (2019)	1.96	9.26	67%
MultiPath (2019)	3.45	9.47	87%
MTP (2018)	3.24	9.59	80%
Trajectron++ (2020)	1.87	9.517	70%
Agentformer (2021)	1.86	-	-
THOMAS (2021)	1.33	6.71	55
WIMP (2020)	1.84	8.49	55%
Goal-LBP (2023)	1.02	9.2	32%
CASPNet++ (2023)	1.16	6.18	50%
SemanticFormer (2024)	0.87	3.88	26%
MTR (UniTraj) (2022) – 5 modes	**0.96**	5.40	43

A clear progression is seen, older models have high error, while modern deep models like Goal-LBP and MTR are under 1.0 m ADE, a four-fold accuracy improvement. Miss Rate drops from nearly 80% to 25%, meaning current models often cover the true future in their set of predictions. It's worth noting that achieving low miss rates sometimes requires outputting many trajectories (nuScenes allows up to 25), but the top methods above manage with around 5–10 modes. Models like Goal-LBP and MTR especially show that incorporating map-based goals and using large-capacity transformers can significantly reduce prediction uncertainty (hitting the correct outcome more frequently).

Trajectron++[55] was selected as the prediction module in the Detection–Tracking–Prediction (DTP) pipeline due to its practical design and strong suitability for modular architectures. It is specifically built to work with tracked agents as input, which makes it compatible with pipelines where detection and tracking are handled separately. This allows trajectory outputs from different tracking models, whether learned or traditional, to be passed directly to the prediction stage without modification.

Although newer models have achieved better scores on recent benchmarks, Trajectron++ remains a strong and widely used baseline in the research community. It consistently outperforms earlier recurrent models and continues to appear in comparative studies on trajectory forecasting[73, 74]. Its performance, while not state-of-the-art, is reliable and representative of modern prediction systems.

Another important reason for choosing Trajectron++ is its open-source and well-documented codebase[55]. The code is clean, actively maintained, and easy to extend. This supports reproducibility and integration into larger systems. It also enables custom modifications such as joint training, which are important for the experimental goals of this work.

In summary, Trajectron++ offers a good balance between performance, flexibility, and ease of use. Its modular structure and public availability make it a practical and effective choice for the prediction component in both sequential and jointly trained pipelines.

2.6 End to end - joint training

In autonomous driving, perception and prediction are traditionally handled by a modular pipeline: separate models for 3D object detection, object tracking, and motion forecasting feed sequentially into one another. This modular approach simplifies development, but it has notable shortcomings. Errors compound as each stage assumes perfect input from the previous stage. Moreover, computation is not shared across tasks, leading to redundant processing (e.g. repeatedly extracting features for detection and tracking). Recently, researchers have explored joint or end-to-end approaches that integrate detection, tracking, and trajectory prediction into a single fully differentiable system. By training multiple tasks together, such models aim to share features and jointly optimize the perception stack, mitigating error propagation and potentially improving overall accuracy. This section focuses on the state-of-the-art in fully LiDAR-based joint perception models for autonomous driving, examining how detection, multi-object tracking, and motion prediction are combined. Both academic literature and industry approaches are reviewed, with a comparison of end-to-end and modular pipelines in terms of training strategies, implementation complexity, performance, and deployment considerations.

2.6.1 Joint Detection and Prediction Models (LiDAR-Only)

Some of the earliest end-to-end LiDAR perception models focused on combining object detection and motion forecasting in a single pipeline. Fast and Furious (FaF)[74] introduced a convolutional model that used sequential LiDAR data to detect objects, estimate velocities, and predict future positions. Forecasting was treated as velocity offset prediction, and the model allowed backpropagation from trajectory loss to improve detection. IntentNet[75] extended this idea by including HD maps and predicting high-level intent along with future motion. Both models showed that detection and prediction tasks could benefit from shared features and joint optimization. For example, IntentNet improved detection using map information and improved forecasting using features learned for detection.

Later models added more advanced architectures and better handling of multi-agent behavior. SpAGNN[76] applied a spatial graph network to model interactions between detected agents, improving predictions without needing a separate tracking step. LaserFlow[77] and RV-FuseNet[78] used range-view LiDAR detectors extended to multiple frames and predicted short-term motion alongside detections. MotionNet[80] introduced a different design that avoided bounding boxes and instead predicted category and motion in each cell of a bird's-eye-view occupancy grid. This allowed the model to detect and forecast motion for both known and unknown object types using a unified representation. MotionNet achieved strong results on nuScenes, highlighting the benefits of this scene-centric formulation.

Recent approaches used multi-task heads and attention mechanisms to further improve joint learning. MultiXNet[81] combined multi-class 3D detection with multi-modal trajectory prediction. It predicted several possible future paths per detected object, using a two-stage approach with initial detection followed by trajectory refinement. FutureDet[73] proposed viewing prediction as a future detection task. It processed sequences of LiDAR frames and output detections not only for the present but also for future frames. Each future detection was linked to a current object, allowing multiple possible futures. FutureDet improved over modular pipelines that separate detection, tracking, and prediction, especially in complex scenes with non-linear motion. These works showed the value of moving from deterministic forecasts to richer multi-modal predictions within a single end-to-end LiDAR-only system.

2.6.2 Integrated Tracking and Prediction Models

While much of the research focused on detection and prediction, some recent models also integrated multi-object tracking into the same system. Traditional tracking uses non-differentiable methods to link detections over time. New models instead learn associations jointly with forecasting, allowing the tracking and prediction steps to share features and be trained together.

StreamMOTP[82] is one such model that combines tracking and prediction in a unified streaming framework. It processes LiDAR frames sequentially and keeps a memory of past object features. This memory supports both association and prediction at each new frame. A spatio-temporal encoding helps align object positions over time, and a learned affinity matrix with a differentiable Sinkhorn algorithm computes associations. Future trajectories are predicted by a dual-branch network, and the entire model is trained end-to-end. StreamMOTP significantly improved long-term consistency and accuracy over separate tracking and prediction models on nuScenes. The memory helps maintain stable predictions by providing a coherent object history.

Another relevant idea is using learnable tracking modules inside larger pipelines. For example, 3DMOT-Former[48] replaced Kalman filter-based tracking with a transformer-based association module. While it does not forecast motion by itself, it can be combined with a prediction module such as Trajectron++[55]. This approach enables joint training of both tracking and prediction, which is the setup used in this thesis. Early results from models like StreamMOTP suggest that joint optimization provides more accurate and stable predictions than independent components. Having a learned tracker in the loop ensures that prediction modules work with a consistent and robust object history, which improves future trajectory accuracy.

2.6.3 Unified Detection–Tracking–Prediction Systems

Some recent frameworks aim to perform object detection, tracking, and motion forecasting together in a single end-to-end model. One of the most complete examples is PnPNet[84], which takes a sequence of LiDAR sweeps and an HD map as input and outputs detections with persistent track IDs and predicted trajectories. The architecture includes three connected modules: a 3D object detection backbone that operates on voxelized BEV inputs, a tracking module that assigns detections to tracks and refines their states using both discrete matching and continuous fitting, and a prediction module that uses the accumulated history of each track to forecast its future motion. These components share features and are trained together, with losses applied to all three tasks.

A key strength of PnPNet is the use of a real-time tracking loop that maintains object history. This allows the model to use longer temporal context for motion prediction. Earlier models without tracking often relied on only one second of past data, while PnPNet benefits from using two to three seconds. On the nuScenes dataset, this led to better performance through occlusions and more accurate motion forecasts. For example, it achieved a final displacement error of 0.93 meters at 3 seconds for vehicles, which was significantly lower than earlier models such as SpAGNN[76]. In addition, PnPNet maintained strong detection accuracy, outperforming previous top models by a margin of about 0.8 percent average precision. These results suggest that joint training helps both detection and forecasting, as the detection head benefits from richer temporal context and the prediction head benefits from improved consistency.

An important detail is the training strategy used by PnPNet. Instead of using ground-truth tracks during training, the model mimics test-time behavior by using its own predicted tracks. This reduces exposure bias and leads to more stable sequential predictions.

Other models have explored similar integration. Fast and Furious[74], while simpler, also combined detection with motion forecasting, although its tracking relied only on propagating detections without maintaining identities. STINet[86] focused on pedestrians, using a spatio-temporal graph to detect, identify, and forecast their trajectories in one model. Although limited to one class, STINet showed that end-to-end learning can handle crowded urban environments.

Overall, unified systems that perform detection, tracking, and prediction together remain difficult to build, especially for diverse traffic scenes with many object types. In the LiDAR-only domain, PnPNet is one of the clearest examples of a fully integrated system that successfully improves performance across all three tasks through joint learning.

Comparison of Representative Models and Frameworks

The following table 2.4 summarizes notable LiDAR-centric models for joint perception and prediction. The specific tasks that are integrated within each model are listed, whether the approach is fully end-to-end (single model) or still modular, the input modality, key innovations, and their performance on the nuScenes benchmark (if reported), along with any limitations.

Table 2.5: Overview of notable LiDAR-based joint perception and prediction models. Each entry lists the integrated tasks (Detection, Tracking, Prediction), a summary of key innovations, and highlights end-to-end learning strategies or architecture features. All models operate primarily on LiDAR data and aim to unify components of the autonomous driving perception stack.

Model (Year)	**Tasks**	**Key innovations**
Fast and Furious (2018)	Det + Track + Pred	First end-to-end 3D CNN in BEV space; fast
IntentNet (2018)	Det + Pred	Predicts intent using HD map + BEV voxel net
SpAGNN (2019)	Det + Pred	Uses spatial attention GNN for agent interaction
MultiXNet (2020)	Det + Pred	Multi-class det + refined multi-future prediction
MotionNet (2020)	Det + Pred	Outputs motion-aware BEV occupancy grids
PnPNet (2020)	Det + Track + Pred	Learns joint association + motion with track memory
FutureDet (2022)	Det + Pred (implicit track)	Forecasts future objects then back-links to now
StreamMOTP (2024)	Track + Pred	Streaming tracking + pred with memory + Sinkhorn match

2.7 Modular vs. End-to-End LiDAR Perception Pipelines: A Literature Review

As mentioned before, autonomous driving perception typically has been approached as a modular pipeline, sequentially handling 3D object detection, multi-object tracking, and trajectory prediction (motion forecasting) as separate components. In a classical stack, the detector finds objects in each LiDAR sweep, a tracker links detections over time into trajectories, and a predictor forecasts each trajectory's future evolution. Each module is trained independently, and only minimalist outputs (object states like positions and velocities) are passed between stages. This decoupling simplifies development but propagates errors: mistakes early in the pipeline cannot be corrected downstream. For example, a missed detection means the tracker and predictor never consider that agent, and any tracking identity switch or localization error can degrade forecast accuracy.

In contrast, end-to-end (E2E) approaches aim to jointly learn or tightly integrate these perception tasks in a single model or training loop. By sharing spatio-temporal features and optimizing all tasks together, an end-to-end pipeline can potentially reduce error compounding and use upstream context to improve downstream predictions. As discussed in the previous section, this holistic approach was shown to be very promising compared to separate modules.

Many trajectory prediction models (e.g. Trajectron++[55], AgentFormer[64], LaPred[70]) assume perfect upstream perception, they are trained and evaluated on ground-truth past trajectories and maps, sidestepping any detection or tracking errors. This “curated input” setting yields clean benchmarks for forecasting algorithms, but does not reflect the realities of a deployed system. In practice, an AV’s forecasting module must consume imperfect data from real-time perception: missed or false detections, identity switches, and coarse maps can all degrade prediction quality. As Xu et al. (2023)[87] note, it has been “not known” how well these modular forecasting models perform when inserted into a real perception pipeline, because they’re usually not evaluated jointly with upstream modules. On the other hand, emerging end-to-end methods that integrate perception and prediction have used disparate evaluation protocols, making direct comparison to modular approaches difficult. Until recently, the field lacked a unified framework to compare modular vs. joint training paradigms under consistent conditions.

Researchers have begun to explicitly compare the modular and end-to-end paradigms for LiDAR-based perception on common benchmarks (notably nuScenes). Below, a key study that evaluate detection, tracking, and prediction in either separate or integrated fashions is reviewed, highlighting their methodology and findings.

Towards Motion Forecasting with Real-World Perception Inputs: Are End-to-End Approaches Competitive?

Xu et al. conduct one of the first comprehensive comparisons between a traditional modular forecasting pipeline and new end-to-end approaches on the nuScenes dataset. A major contribution of this work is a unified evaluation pipeline that inserts learned detection, tracking, and online mapping modules in front of forecasting models, thereby allowing apples-to-apples evaluation of conventional vs. end-to-end forecasting methods. They compare two state-of-the-art “conventional” predictors (AgentFormer[64] and LaPred[70], both originally trained on ground-truth trajectories) against two recent end-to-end networks (ViP3D[88] and UniAD[32]) using the same upstream perception outputs. Notably, UniAD (Hu et al., 2023) is a “Unified Autonomous Driving” model that jointly addresses detection, mapping, motion forecasting and planning in one architecture, while ViP3D (Gu et al., 2023) is an end-to-end camera-based 3D trajectory predictor using query-based vision transformers. By evaluating these under a common protocol, Xu et al. effectively ask: given the same sensor inputs, can an end-to-end learned pipeline match or outperform the traditional separated pipeline in detection, tracking, and prediction performance?

Findings: Perhaps surprisingly, the study concludes that current end-to-end approaches are not yet competitive with modular pipelines in forecasting accuracy. When fed identical detection and tracking outputs, the classic forecasting models (AgentFormer[64], LaPred[70]) yielded equal or lower prediction error than the joint models (UniAD[32], ViP3D[88]) in the nuScenes evaluation. In fact, the “end-to-end forecasting paradigm is so far not better than the conventional one”. One observed issue was that the end-to-end models struggled to effectively utilize map information in nuScenes, for example, UniAD and ViP3D did not significantly benefit from HD maps, whereas LaPred (a map-aware predictor) excelled. The authors suggest that better multi-task learning strategies and map integration designs are needed to realize the potential of end-to-end pipelines.

Crucially, Xu et al. also quantify the performance gap between the idealized “clean” setting and the real-world perception setting. Using their benchmark, they show that plugging ground-truth tracks into a forecasting model versus using detector+tracker outputs can cause a large drop in prediction metrics. For example, AgentFormer’s nuScenes forecasting mAP score plummeted from 0.343 (with perfect data) to 0.112 when using realistic perception inputs. They trace this degradation to more than just precision loss: the nature of the input errors (missed detections, noisy trajectories, etc.) fundamentally hurts the predictor in ways that simple fine-tuning cannot fix. Finetuning the forecasting models on imperfect perception outputs provided only minor improvements, indicating that new robustness techniques are needed. The authors recommend two avenues: (1) Improve upstream perception – for instance, increase detection range and accuracy (especially in localization, not just classification score) to narrow the gap. (2) Make forecasting models aware of perception imperfections – e.g. design trajectory predictors that can handle missing or noisy past data, perhaps by representing uncertainty or leveraging raw sensor features. They also highlight the influence of target distance: errors worsen for far-away agents, a factor typically hidden since forecasting metrics are computed in the target's frame rather than the ego-vehicle frame. Future benchmarks could stratify results by distance to ensure models perform robustly even on distant objects.

Overall, Xu et al. provide a timely reality check: integrated perception-forecasting models have not yet surpassed carefully tuned modular pipelines on nuScenes, and transitioning from ideal inputs to real sensor data remains a major challenge. Their work solidifies the evaluation protocol needed to track progress on this issue, and it motivates research into closing the "simulation-to-real" gap for motion forecasting. Unfortunately, the authors do not release their evaluation framework publicly, so a comparable modular pipeline is implemented as part of this work, to enable a fair and consistent comparison between sequential and joint perception-prediction paradigms.

2.7.1 Research Gaps and Thesis Positioning

Despite the progress, there are notable gaps in the existing research that motivate further investigation. One gap is the limited exploration of learning-based 3D tracking within an end-to-end perception pipeline. Many studies that integrate perception and prediction have actually bypassed the tracking step: e.g. FutureDet[73] effectively skip explicit multi-object tracking by having the network predict future states directly. Others use relatively rudimentary tracking methods. For instance, Xu et al.'s benchmark employed a standard tracker (CenterPoint's built-in tracker) to supply inputs to forecasting models; their comparison did not test any advanced trainable tracking mechanism in between detection and forecasting. This is an open opportunity, intuitively, a powerful learned tracker (e.g. one that uses graph neural networks or transformers to associate detections, like 3DMOTFormer by Ding et al. 2023) could provide better trajectory histories to the predictor than a hand-tuned IoU matcher. 3DMOTFormer, for example, uses a graph-transformer architecture to learn affinities between detections and existing tracks, achieving state-of-the-art MOT accuracy on nuScenes by refining data association and velocity estimation via attention. Moreover, 3DMOTFormer can be extended to enable end-to-end training with a prediction model such as Trajectron++. However, to date, no specific work has implemented this combination or systematically compared it within a unified framework. While other models have explored joint detection and prediction or integrated tracking and forecasting, a comprehensive evaluation that contrasts sequential pipelines, using either classical or learnable trackers, with a jointly trained tracking-prediction model under consistent inputs is still lacking. This gap is precisely one focus of this thesis.

This thesis research is designed to fill these gaps. Three pipeline variants on nuScenes are compared:

1. a fully modular chain using a state-of-the-art LiDAR detector (CenterPoint[20]), a classic but high-performing 3D tracker (PolyMOT[39], a polyhedral optimization-based tracker), and a graph-based trajectory predictor (Trajectron++[55]). This represents the conventional approach where each component is optimized individually.

2. a full modular pipeline using the same detector and predictor, but replacing the tracker with 3DMOTFormer[48], a learned transformer-based 3D MOT algorithm. This will illuminate the impact of a stronger, learnable tracking module on the end-task (forecasting) when everything else is kept constant.

3. an integrated end-to-end model that fuses the tracking and prediction stages, essentially, combining 3DMOTFormer and Trajectron++ into one joint model, trained on the final prediction

objective. In this joint model, the tracker and predictor share information and gradients (detection is still provided by CenterPoint's output).

In all three setups, each stage is additionally evaluated by substituting upstream inputs with ground truth data, such as ground truth tracks or detections, to quantify the extent to which prediction performance degrades due to upstream errors.

By evaluating all three scenarios on the same dataset and metrics, this enables answering :

> *How much does an advanced learned tracker improve forecasting over a traditional tracker?*
>
> *Can joint training of tracking and prediction outperform the traditional pipeline where each stage is trained independently?*

These questions have been hinted at but not directly tested in prior literature. By positioning this work against the studies reviewed, it is aimed to contribute a nuanced understanding of modular vs. joint training paradigms for LiDAR-based perception. If the findings echo Xu et al. (that end-to-end is still lagging), the analysis will detail where the shortcomings arise within the pipeline. If instead the end-to-end tracker+predictor shows an advantage, it will provide one of the first evidence that integrating a modern learned tracker can push forecasting performance beyond the modular state-of-the-art. In either case, this research will extend the comparative analysis to a facet that has seen little attention: the role of differentiable tracking within the perception-to-prediction pipeline. Ultimately, closing these gaps will guide the design of next-generation autonomous driving systems that can reliably perceive and anticipate the future in complex environments.

3 Methodology & System Architecture

This chapter details the methodology and system architecture developed for the comparative study of sequential and end-to-end LiDAR perception frameworks in autonomous driving. It outlines the design of the proposed pipeline, emphasizing its modular nature, the specific components utilized, and the technical implementation details that enable a direct comparison between the two paradigms using state-of-the-art models. Additionally, the chapter provides a detailed explanation of the joint architecture approach, including the integration and training of the tracking and prediction modules in a unified end-to-end setup.

3.1 Overview of the Proposed Pipeline

The fundamental goal of this research is to rigorously evaluate and compare the performance and characteristics of two principal architectural paradigms for LiDAR-based perception in autonomous driving: the traditional sequential pipeline and a more integrated, end-to-end approach. To achieve this, a flexible framework capable of instantiating and executing both configurations within a controlled environment was engineered.

Sequential Perception Pipeline

The sequential pipeline represents a common and well-established architecture in autonomous driving systems. It consists of three primary stages that operate in a linear fashion: Object Detection → Object Tracking → Trajectory Prediction. In this configuration, each stage functions as an independent module, typically trained and optimized in isolation. The output of one stage directly serves as the input for the subsequent stage, enabling a modular and interpretable flow of information.

The first stage, object detection, processes raw 3D LiDAR point cloud data to identify, classify, and localize dynamic objects in the environment. The output generally includes 3D bounding boxes and confidence scores associated with each detection. For this task, CenterPoint [20] is utilized, a high-performance, anchor-free detection model known for its speed and accuracy. The model is implemented using the OpenPCDet framework [34], which provides a flexible and modular platform for LiDAR-based 3D object detection.

Following detection, the object tracking stage is responsible for associating detections across consecutive frames to maintain consistent object identities over time. This stage also estimates motion history features such as velocity and acceleration. In this study, two tracking approaches are evaluated: a classical assignment-based method, PolyMOT [39], which relies on data association algorithms such as the Hungarian algorithm [38], and a learning-based tracker, 3DMOTFormer [48]. While the sequential pipeline primarily relies on PolyMOT to represent a standard discrete tracking stage, 3DMOTFormer is also evaluated within the sequential framework for comparison, and further leveraged in the joint training setup discussed later.

The final stage of the pipeline, trajectory prediction, takes the tracked object histories as input and forecasts their likely future trajectories over a defined time horizon. For this task, Trajectron++ [55] is employed, a graph-structured probabilistic model designed to generate multi-modal future trajectory distributions based on past motion and interaction context.

A defining characteristic of the sequential pipeline is the decoupling of the individual modules during training. Each model, CenterPoint for detection, PolyMOT or 3DMOTFormer for tracking, and Trajectron++ for prediction, is trained independently using task specific ground truth annotations. Communication across modules is achieved by passing the discrete outputs from one stage as direct input to the next. This design choice closely reflects many real-world perception systems, where modules are often developed, validated, and deployed independently by different teams or organizations.

End-to-End Perception Pipeline

The end-to-end pipeline investigates the potential benefits of integrating the tracking and prediction stages into a single, jointly trainable system. While the initial detection stage (using CenterPoint) remains separate, the output detections are fed into a combined Tracking + Prediction module. This module, primarily built around the 3DMOTFormer[48] tracker integrated with components of Trajectron++[55] prediction, is designed to be trained end-to-end.

The key feature of this configuration is the introduction of differentiability between the tracking and prediction processes by replacing the traditional non-differentiable Hungarian matching[38] section with a differentiable Sinkhorn-based[83] soft assignment over the affinity matrix. This allows the prediction loss to propagate gradients back through the tracking mechanism during training. The hypothesis is that by optimizing tracking parameters based on their impact on downstream prediction accuracy, the system can learn tracking behaviors that are more beneficial for generating accurate future trajectories. This setup explores the potential for performance gains by relaxing strict modularity in favor of a more integrated learning objective.

Comparative Framework Design

The overall framework is carefully designed to enable a fair and direct comparison between the sequential and end-to-end perception paradigms. To ensure the validity of this comparison, both configurations are trained and evaluated using the same underlying dataset (NuScenes [1]) which provides diverse and high-quality annotations suitable for detection, tracking, and prediction tasks.

To isolate the impact of the pipeline architecture itself, the same state-of-the-art components are used consistently across both setups. For example, CenterPoint[20] is employed for object detection and Trajectron++[55] for trajectory prediction, regardless of whether the modules are trained independently or as part of a joint system. This consistency ensures that any observed differences in performance can be attributed to the architectural structure, sequential versus end-to-end, rather than differences in model design or capability.

In addition, a standardized software infrastructure is adopted using Docker [89], which guarantees consistent execution environments and dependency management across all experiments. This eliminates variability that arises from platform discrepancies or package versioning.

Finally, both configurations are evaluated using a shared protocol and a common set of metrics. This unified evaluation approach further supports an objective comparison by ensuring that the results are generated and interpreted under identical conditions. Overall, this carefully controlled setup ensures that any differences in performance are mainly caused by the way the pipeline is structured, whether sequential or end-to-end, and not by other unrelated factors.

3.2 Implementation Details

The implementation of a flexible framework capable of supporting and comparing distinct perception pipeline architectures, each relying on complex deep learning models with potentially conflicting software dependencies, presents significant engineering challenges. The approach used in this work, leverages a highly modular and containerized system based on Docker[89] to effectively manage these complexities.

Modular Containerized Architecture using Docker

A fundamental decision was to put each main part or set of parts into its own separate Docker container. This meant creating individual containers for Object Detection, which held the CenterPoint model and OpenPCDet necessary files. There was a container for Sequential Object Tracking, housing the PolyMOT tracker. Furthermore, a container was made for Trajectory Prediction, with the Trajectron++ model, and finally, a container for Joint Differentiable Tracking and Prediction, containing the 3DMOTFormer [48] model for sequential evaluation and integrating both the 3DMOTFormer and Trajectron++ for combined training.

The main reason for using Docker containers for each key module, Detection, Tracking, Prediction, and the combined Tracking/Prediction part, comes from several important benefits. Modern deep learning models often need specific versions of software like PyTorch, TensorFlow, CUDA, cuDNN, and other specialized packages. If all these models were put together in one system, it would cause problems with conflicting software versions and make installation very difficult. By keeping each module in its own container, it gets a clean, dedicated space where its specific software can be installed and managed without affecting other modules. This greatly improves how stable the system is and reduces "only works on one machine" issues.

Also, Docker containers package the application code with its exact operating environment. This helps ensure that experiments can be repeated precisely on different computers or at a later time because the environment for running the code is consistent and saved within the container image. This is very important for scientific research. The design with separate containers is also key for comparing different methods. It allows for easily swapping out different tracking methods, such as evaluating the 3DMOTFormer within the sequential process by simply replacing the PolyMOT container setup. Similarly, different prediction models can be swapped out by building a new container image for that module and updating the settings to use it. Doing this would be much harder in a system where everything is tightly connected or combined into one large piece.

Furthermore, developers can focus on building and fixing problems in one module within its separate container without worrying about breaking other parts of the system. If a problem occurs, it can often be traced back to a specific container. While managed by external scripts, the container approach also allows for easier dedicated allocation of resources, such as assigning specific GPUs to individual containers using Docker's –gpus flag. This is important for training or running calculations on models that require a lot of computing power.

Unlike a system where all parts are closely linked in one set of code and environment, the container-based approach emphasizes separation and flexibility. While one could theoretically save everything locally and run different programs, managing the separate, potentially conflicting software environments for CenterPoint, PolyMOT, 3DMOTFormer, and Trajectron++ directly on the host machine would be a huge burden and a fragile setup. Docker simplifies this complexity by providing clean, portable environments for each part to run.

Centralized Configuration via `config.env`

Managing the many settings needed to build, run, and manage several Docker containers is done using a central settings file called `config.env`. Each container might need specific computer resources and interact through shared resources. This file sets up key environment variables that control important parts of how the system is set up and run.

Using a single `config.env` file, which is read by different scripts that manage containers and run the system, offers big benefits. All important global settings are defined in one place. This avoids having repeated information and potential problems that happen when settings are spread across many scripts or files. Changing the system setup, such as altering the dataset path, picking which GPU to use, updating container names, or pointing to a different shared storage space, only requires editing this one file. This makes trying out new things and adapting to different computer hardware or environments much simpler. By saving the `config.env` file along with the code, the exact settings used for a specific experiment are recorded, which further helps in getting the same results again.

The environment variables that control how experiments are set up, like data paths, names for model containers and images, and GPU input parameters, are all specified in the `config.env` file (see Listing 1).

```
# config.env

GPU=0
DATASET_PATH=/home/supa/dataset/nuscenes
SHARED_VOLUME=shared-data

# images
BASE_IMAGE=docker-base-cuda116
DETECTION_IMAGE=openpcdet-image
TRACKING_IMAGE=object-tracking-image
PREDICTION_IMAGE=object-prediction-image

DIFFERENTIABLE_TRACKING_IMAGE=differentiable-tracking

# containers
DETECTION_CONTAINER=object_detection_container
TRACKING_CONTAINER=object_tracking_container
PREDICTION_CONTAINER=prediction_container

DIFFERENTIABLE_TRACKING_CONTAINER=differentiable-tracking_container
```

Listing 1: `config.env` file used to define environment-wide variables for experiments. These variables control paths, GPU set parameter, container names etc.

Therefore, the `config.env` (Listing 1) file acts as the main control center for setting up the entire experiment, making it easy for users to adjust the framework to their specific environment and experimental needs.

Inter-Container Data Exchange via Shared Docker Volume

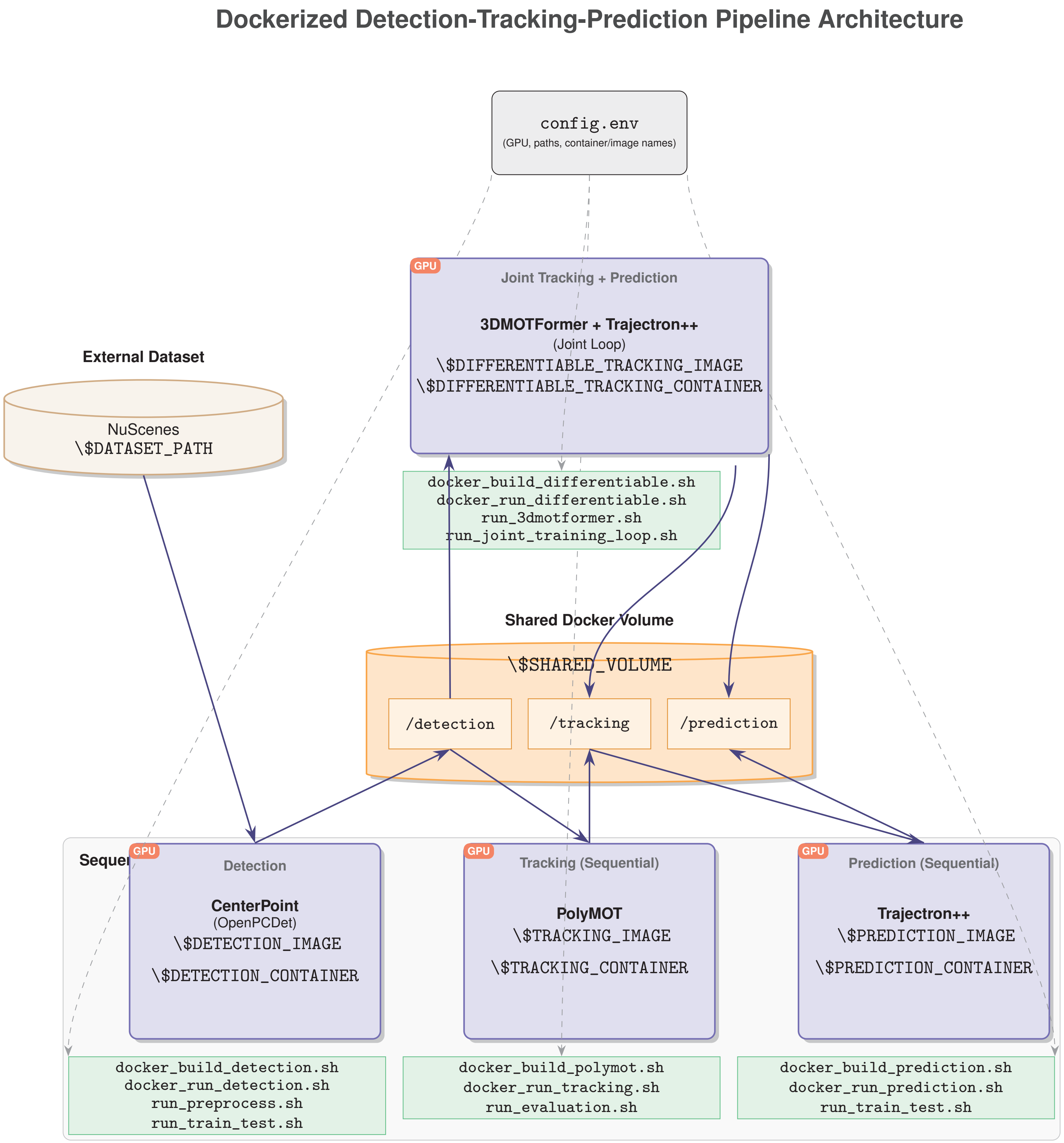
Since each module runs in its own separate container, a way is needed for them to share data. This includes raw input, processed results, trained models, and output from evaluations. A shared Docker volume was chosen as the main method for containers to communicate with each other. A lasting volume, named `$SHARED_VOLUME` (as set in the configuration), is created and attached at the same location within each relevant container.

The choice to use a shared volume instead of just saving data to the computer's hard drive and linking folders was made for several reasons. Docker volumes are automatically created and managed by the Docker engine, removing the need for users to manually set up shared file paths on every computer. This makes it easier to use across different systems and ensures consistency without extra setup. Also, Docker volumes work smoothly across different operating systems like Linux, macOS, and Windows, avoiding common problems with file permissions. Unlike bind mounts, they do not require manual permission adjustments such as chmod, which simplifies deployment and usage considerably

Volumes also support clear actions for their entire lifespan, such as cleaning up or deleting, through Docker. This prevents old or leftover files from building up in computer folders, reducing clutter or accidental file exposure on the user's system. Using named Docker volumes greatly improves how fast data is read and written, especially on macOS and Windows. Unlike linked folders that depend on Docker Desktop's file-sharing system, which adds extra work, named volumes are managed directly by the Docker engine or virtual machine, resulting in speeds almost as fast as native input/output. This is important for tasks that involve moving many files quickly, like logs, notes, or parsing JSON. Finally, volumes reduce the risk of accidentally showing sensitive computer files or overwriting important local data. By not directly linking local paths, the container environment remains more controlled and secure, lessening the impact if a container does not behave as expected.

A dedicated Docker volume is created and mounted into each relevant container. This shared volume is set up with specific subfolders for each step of the process, for example, `$SHARED_VOLUME/detection`, `$SHARED_VOLUME/tracking`, `$SHARED_VOLUME/prediction` and `$SHARED_VOLUME/differentiable_tracking`. This volume acts as a central storage place for all data inputs, outputs, and temporary results shared between the modules. The process involves one container writing its output, like detection results in a specific file format, to its assigned subfolder. The next container in the process is set up to read its needed input from another assigned location within the same shared volume. This organized approach manages the flow of data and clearly shows how the modules depend on each other. The data exchanged includes preprocessed dataset parts, which can be accessed by Detection, Tracking, and Prediction. It also includes detection results, such as bounding boxes, scores, and features, which are created by the Detection container and used by the Tracking and Joint Tracking+Prediction containers. Tracking results, like object identities and past paths, are produced by the Tracking container and used by the Prediction container in the sequential process. These are also produced and used internally by the Joint Tracking+Prediction container. Additionally, trained model checkpoints and configuration files for each module are stored permanently for later use.

The overall design, showing the role of the shared volume, the configuration file, the dataset, and the different containerized modules and how they interact, is shown in Figure 3.1. Below, the structure and purpose of each primary subdirectory within the shared volume it's detailed, starting with the contents related to the object detection stage.



Note: `config.env` variables configure container scripts and GPU access (`-gpus "device=$GPU"`).
Scripts inside each container handle training/evaluation. Shared volume provides structured I/O via dedicated subdirectories.

Figure 3.1: Overview of the proposed Dockerized sequential Detection–Tracking–Prediction (DTP) architecture used in this study. Each module (CenterPoint, PolyMOT, Trajectron++) operates within its own container environment and communicates via a structured shared volume. This setup enables modular evaluation and controlled error propagation analysis across stages.

Structure of `/$SHARED_VOLUME/detection`

The `/$SHARED_VOLUME/detection` directory is dedicated to storing the outputs generated by the object detection container, specifically focusing on trained model checkpoints and the results of running inference and evaluation. This directory acts as the bridge, providing the trained detection models and providing the detected object outputs that are consumed by both the sequential tracking container and the joint tracking and prediction container. Based on the object detection framework (OpenPCDet[34] with CenterPoint[20]) and the experimental setup, the structure within this directory is organized to manage outputs from different detection model configurations or training runs. A typical structure is illustrated below:

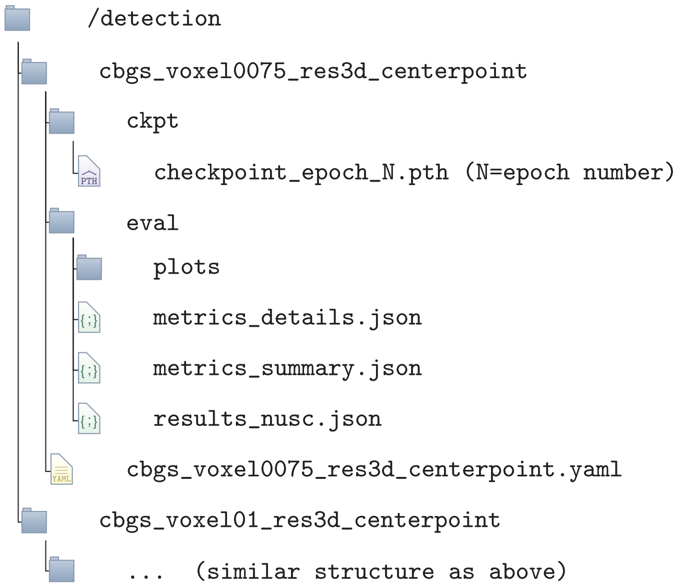

/detection
cbgs_voxel0075_res3d_centerpoint
ckpt
PTH
checkpoint_epoch_N.pth (N=epoch number)
eval
plots
{;}
metrics_details.json
{;}
metrics_summary.json
{;}
results_nusc.json
YAML
cbgs_voxel0075_res3d_centerpoint.yaml
cbgs_voxel01_res3d_centerpoint
... (similar structure as above)

Figure 3.2: Structure of the `/detection` subdirectory within the shared volume.

As Figure 3.2 shows, the top level under `/detection` contains folders named after specific detection model setups, like `cbgs_voxel0075_res3d_centerpoint` and `cbgs_voxel01_res3d_centerpoint`. This arrangement is done on purpose to keep experimental variations organized. Each subfolder holds all the outputs related to a specific training run or setup of the CenterPoint model, for example, using different voxel sizes or training schedules. This clear separation is crucial for knowing which outputs belong to which experiment, preventing results from different detection models from getting mixed up when later stages are evaluated. By organizing results by setup, it becomes easy to find and access the outputs, which include trained models and detection results, from the best performing detection model to use in the following tracking and prediction experiments.

Inside each model setup folder, two main subfolders are found. One is `ckpt/`. This subfolder stores the saved model checkpoints, which are `.pth` files, created during the detection model training process. Each file, such as `checkpoint_epoch_20.pth`, is a snapshot of the model's learned information and state at a certain point during training. Storing checkpoints here serves multiple important purposes. It allows the training process to restart from a saved point, helps in choosing models based on their performance at different training stages, and provides the necessary `.pth` files that the detection container loads when performing inference on validation or test data. The scripts that manage training, for instance, `run_train_<model>.sh`, are set up through the OpenPCDet settings or command line arguments to save these files to the linked shared volume path inside the container. Similarly, evaluation scripts, such as `run_test_<model>.sh`, are set to load a specific checkpoint from this location using arguments like `-ckpt $CKPT`, where `$CKPT` refers to a path within the shared volume.

The other main subfolder is `eval/`. This directory contains the results produced after running a trained detection model on a specific part of the dataset, for example, the validation set, and comparing these detections against the true labels. The contents of this directory are very important for both checking the detection model's performance with numbers and, more importantly, for providing the input data needed by the next tracking stage. It usually contains several JSON files.

One of these is `results_nusc.json`. This file holds the raw output of the detection model for each processed sample or keyframe during inference on the evaluated dataset split. It strictly follows the official NuScenes detection results format, which is specifically designed to be the input for the NuScenes evaluation toolkit and provides a standard way to represent detection outputs. The expected JSON structure for this file is as follows:

```
submission {
    "meta": {
        "use_camera":   <bool>    -- Whether this submission uses camera data.
        "use_lidar":    <bool>    -- Whether this submission uses lidar data.
        "use_radar":    <bool>    -- Whether this submission uses radar data.
        "use_map":      <bool>    -- Whether this submission uses map data.
        "use_external": <bool>    -- Whether this submission uses external data.
    },
    "results": {
        sample_token <str>: List[sample_result] -- Maps each sample_token to list of
        sample_results.
    }
}

sample_result {
    "sample_token":       <str>          -- Identifies the sample/keyframe.
    "translation":        <float> [3]    -- Bounding box location (center_x, center_y,
    center_z) in global frame.
    "size":               <float> [3]    -- Bounding box size (width, length, height) in m.
    "rotation":           <float> [4]    -- Bounding box orientation as quaternion in
    global frame.
    "velocity":           <float> [2]    -- Bounding box velocity (vx, vy) in global frame
    (if estimated).
    "detection_name":     <str>          -- Predicted class label (e.g., car, pedestrian).
    "detection_score":    <float>        -- Object prediction score (0 to 1).
    "attribute_name":     <str>          -- Predicted attribute or empty string.
}
```

The file includes a `"meta"` field that indicates the input types used, for our LiDAR-based detector, `"use_lidar"` is true. The main data is within the `"results"` dictionary, where each key is a `sample_token`, a unique identifier for a keyframe from the dataset. The corresponding value is a list of `sample_result` dictionaries. Each `sample_result` dictionary provides details about a single detected object, including its 3D bounding box position, which includes `translation`, `size`, and `rotation`. It also includes the estimated `velocity`, if the detector supports it like CenterPoint, the predicted class label (`detection_name`), a confidence `detection_score`, and an associated `attribute_name`. This `results_nusc.json` file represents all the raw detection outputs for the evaluated split and directly feeds into the tracking container. The tracking module is specifically designed to read and use the data within this file to match data and create object tracks across sequences. Storing this standard file in the shared volume ensures a clear, compatible connection for the next tracking stage.

Other files are `metrics_summary.json` and `metrics_details.json`. These files contain the performance numbers calculated by comparing the detections in `results_nusc.json` against the true labels using the NuScenes evaluation toolkit. `metrics_summary.json` gives a high level overview, including metrics like class specific Average Precision at various distance limits, the overall Mean Average Precision, True Positive errors for different location and attribute aspects, and the combined NuScenes Detection Score. `metrics_details.json` provides more detailed data. There is also a `plots/` subfolder. This directory is for storing any visual outputs generated during the detection evaluation process, such as PR curves or pictures of detection results overlaid on point clouds or images, which helps in visual analysis. The scripts that manage the evaluation process within the detection container are set up to ensure these outputs are written to the correct location within the `eval/` subfolder on the shared volume, making the detection results and performance metrics centrally available.

It is important to note that, because of the default way OpenPCDet framework handles data, which was difficult to change much without causing potential problems, the initial preprocessed dataset files needed by the detection model for training and inference are not stored within this shared volume subfolder. Instead, these preprocessed files are located in the main dataset path, `$DATASET_PATH`, outside the shared volume. This path is then accessed or mounted by the detection container when it operates. The `/detection` shared volume subfolder is specifically for the outputs of the detection process: the trained models and the inference or evaluation results. This separation of storage responsibility keeps the shared volume focused on exchanging data between stages and logging results, while using the detection library's own data handling for preprocessing.

It was ensured that the scripts managing the detection container correctly set up the OpenPCDet execution to save its checkpoints and evaluation outputs to these specific, organized paths within the `/$SHARED_VOLUME/detection` directory. This makes them easy to find and use by the following tracking and prediction containers.

Shared-Volume Layout: `/tracking`

The `/tracking` subfolder holds all the outputs created by the PolyMOT tracking container. Just like with detection, separate results are kept for both CenterPoint variations, namely `cbgs_voxel0075_res3d_centerpoint` and `cbgs_voxel01_res3d_centerpoint`. There is also a third section that uses perfect detections from true data, which acts as an ideal standard for the best possible performance. A visual summary of this folder structure is provided in Fig 3.3.

Before the tracking process can begin, PolyMOT needs two extra inputs to be created. The first is the `first_token_table/` directory. This folder contains `nusc_first_token.json`, which is a fixed map listing the first frame of every NuScenes scene. This file is necessary for real time processing, where the tracker needs to know when to start a new track. This file is made using the `first_frame.py` script, and it helps by avoiding database calls during operation, which makes the tracking faster and ensures consistent results. The second is the `detector/` directory. PolyMOT expects all detections to be in strict time order. Since the raw outputs from the detector do not guarantee this, the `reorder_detection.py` tool is used to create `val_centerpoint.json`. This file is a time-sorted version of the detection results used during tracking. This reordering is very important to prevent ID changes caused by input frames that are not in the correct time sequence.

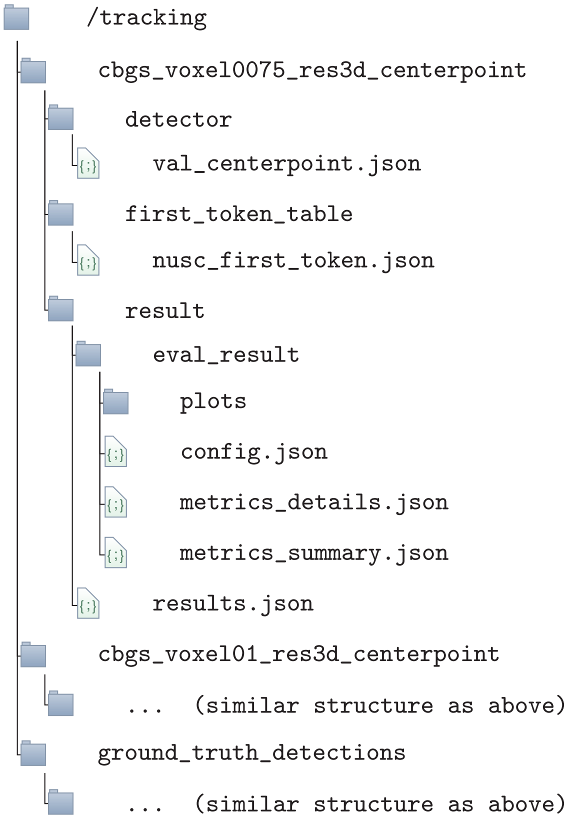


Figure 3.3: Tracking container output under `$SHARED_VOLUME/tracking`. Structure shown here for the `voxel0075` model; the others follow the same layout.

Each experiment folder, for example, `cbgs_voxel0075_res3d_centerpoint`, includes `result/results.json`. This is the main output of the tracker and serves as the input for the prediction stage. Its format is exactly like the detection format, with three added fields specific to tracking:

```
{
        "tracking_id": "<str>",
        "tracking_name": "<str>",
        "tracking_score": "<float>"

        // Note that except for these tracking_* fields the result format
        // is identical to the detection challenge.
}
```

These fields allow trajectories to continue smoothly over time and are used by the prediction module to divide temporal sequences for each object. The rest of the format, including `sample_token`, `translation`, and `rotation`, is unchanged from the detection format and is not described again here.

Another part of the tracking results interface is `result/eval_result/`. This folder contains `metrics_summary.json`, `metrics_details.json`, and visual `plots/`. These reflect the evaluation outputs of detection, but they show metrics specific to tracking, such as AMOTA, AMOTP, and ID switches

Since PolyMOT is a rule based tracker without any parameters that can be trained, no `.pth` or weight files are created. This ensures that the tracking stage remains simple and does not keep any state, making it easier to quickly evaluate under different detector inputs without creating unnecessary files.

The third entry, `ground_truth_detections`, follows the same structure and format but receives 100% accurate 3D boxes as input. This version is used to separate the effect of detector quality from the tracking logic and provides a reference for the best possible performance when evaluating the overall system.

Shared-Volume Layout: `/prediction`

The `/prediction` subdirectory contains all artefacts produced by the Trajectron++ container. This stage of the pipeline consumes temporal tracks generated by the tracker (or oracle ground truth), performs trajectory forecasting, and logs both numerical results and model checkpoints. Unlike detection and tracking, prediction involves multiple combinations of input sources, model architectures, and output configurations, resulting in a deeply nested but strictly structured directory layout (Figure 3.4).

```
/prediction
├── models
│   └── model_name (for vel_ee, int_ee, int_ee_me, robot)
│       ├── config.json
│       └── model_registrar-N.pt (N=epoch number)
├── processed
│   └── subdir (e.g., ground_truth, tracking, tracking_learning_based, ...)
│       ├── nuScenes_train_full.pkl
│       ├── nuScenes_test_full.pkl
│       └── nuScenes_val_full.pkl
└── results
    └── eval_type (e.g., ground_truth, tracking, ...)
        ├── model_name (e.g., int_ee, vel_ee, ...)
        │   └── class_name (PEDESTRIAN, VEHICLE)
        │       └── <model_name>_{8,12}_{ade,fde,mr}_most_likely_z.csv
        └── (structure repeated for other models and classes)
```

Figure 3.4: Directory hierarchy under `$SHARED_VOLUME/prediction`. Each result is organized by input type, model architecture, and semantic class (vehicle or pedestrian).

The prediction sub-tree is organized into three main folders. The first is `models/`. This directory holds all trained Trajectron++ model versions, with each one stored under its own name. The four supported versions show increasing levels of contextual modeling. `vel_ee` is the basic model, using only velocity based movement without map or robot information. `int_ee` adds to the base model by including interaction-aware movement between agents. `int_ee_me` further improves interaction dynamics by adding static scene context through map encoding. Finally, `robot` includes all the previous features, plus future path conditions for the ego vehicle, which allows for robot aware prediction. Each subfolder, for example, `models/int_ee_me`, contains the complete model definition in `config.json` and all the saved checkpoints for each training step. These names are consistent across training and evaluation and are directly referred to in scripts for easy reproduction. This structure allows models to be switched easily during evaluation, while keeping each version's settings and architecture isolated.

The second folder is `processed/`. This contains pre-processed scene data saved as `.pkl` files. These files are the required input format for Trajectron++, holding structured temporary agent data, spatial features, and semantic labels. Each subfolder here shows where the agent tracks came from, such as `ground_truth`, which refers to ideal paths derived from NuScenes annotations, `tracking`, which refers to paths from the PolyMOT tracker results using traditional detection, and `tracking_learning_based`, which refers to paths from the learnable tracker, 3DMOTFormer. Each folder contains three splits: `nuScenes_train_full.pkl`, `nuScenes_val_full.pkl`, and `nuScenes_test_full.pkl`, which contain the full space and time context of the scene. During preprocessing, each NuScenes scene is gone through frame by frame to get agent states, including position, heading, and bounding box sizes, along with object identity and frame index. Agents are filtered by category, either `PEDESTRIAN` or `VEHICLE`, and aligned with a fixed time step sequence. Their global coordinates are then adjusted to a local coordinate frame centered around the scene's origin, typically the ego vehicle. The result is a set of time indexed trajectories structured for Trajectron++'s graph based encoder decoder architecture. This separation by data source ensures that experiments are modular and reproducible across different model versions, and it prevents data from getting mixed between ideal and predicted inputs.

The third folder is `results/`. This contains CSV files with evaluation metrics produced by Trajectron++ after forecasting trajectories for each experiment setup. The directory is organized in a hierarchy: first by input type, such as `ground_truth` or `tracking`, then by model version, for example, `vel_ee` or `int_ee`, and finally by semantic class, like `VEHICLE` or `PEDESTRIAN`. This ensures that results from different sources and architectures remain separate and easy to trace. Each evaluation run creates a set of CSV files named as:

```
<model_tag>_{8,12}_{ade,fde,mr}_most_likely_z.csv
```

Here, the number suffix, `8` or `12`, indicates how far into the future the prediction goes in timesteps, meaning 4 or 6 seconds into the future, assuming 0.5 second intervals. The suffix, `ade`, `fde`, or `mr`, specifies the metric. ADE, or Average Displacement Error, is the average straight line distance between the predicted and true path across all time steps. FDE, or Final Displacement Error, is the straight line distance at the very last prediction step. Miss Rate, or MR, is the percentage of predictions where the final displacement goes beyond a certain limit, usually 2 meters. Trajectron++ performs multimodal prediction using a probabilistic model with hidden variables. For each sample, it creates multiple possibilities, and metrics are calculated using the most likely hidden mode, which is shown by `most_likely_z` in the filename. This method keeps the realism in probabilistic forecasting while still allowing for direct comparison. All result files are automatically created by the evaluation script, which also makes sure the correct paths into the shared volume are used based on the provided input and model tags. These CSV files are directly used by other analysis tools, for example, Jupyter notebooks like `NuScenes_Quantitative.ipynb`, for showing data visually and for comparisons.

All folder creation and data population are handled automatically through a containerized evaluation script, which ensures consistent locations for results and proper model handling. A simpler version is shown in Listing 2. The script goes through each model, including `vel_ee`, `int_ee`, `int_ee_me`, and `robot`, and saves results under the corresponding output root. It also tells the difference between input types, such as `ground_truth` and `tracking`, and makes sure that semantic class folders like `VEHICLE` and `PEDESTRIAN` are created and used correctly. This modular structure was designed to prevent results from being overwritten, to simplify how comparisons are made, and to make it easier to evaluate multiple models from a single starting point.

```
# Loop through each model and evaluate for multiple prediction horizons
for TAG in "${!MODELS[@]}"; do
    MODEL_PATH="$BASE_MODEL_PATH/${MODELS[$TAG]}"
    OUTPUT_PATH="$OUTPUT_ROOT/$TAG/$NODE_TYPE"

    for PREDICTION_HORIZON in 8 12; do

        # Run evaluation for the current model and prediction horizon
        docker exec -it $PREDICTION_CONTAINER bash -c "
            cd experiments/nuScenes &&
            mkdir -p $OUTPUT_PATH && # Create output directory if it doesn't exist
            python evaluate_with_missrate.py \
                --model $MODEL_PATH \
                --checkpoint=$CHECKPOINT \
                --data $DATA_PATH \
                --output_path $OUTPUT_PATH \
                --output_tag ${TAG} \
                --node_type $NODE_TYPE \
                --prediction_horizon $PREDICTION_HORIZON
        "
    done
done
```

Listing 2: Evaluation script that loops over all prediction models and inputs. It programmatically creates result directories and launches model evaluation via Docker.

Shared-Volume Layout: `/differentiable_tracking`

The `/differentiable_tracking` subdirectory stores all artefacts related to the learnable tracking pipeline (3DMOTFormer) and its joint integration with Trajectron++ for end-to-end optimization. This stage supports both standalone tracking and differentiable joint training with trajectory forecasting (Figure 3.5).

3DMOTFormer needs each scene to be represented at the frame level, where both detections and true annotations are lined up, structured, and saved. This format is different from detection or prediction formats and uses one `.pkl` file for each frame, grouped under scene folders, for example, `scene_0000` to `scene_0699`. For every scene, 40 frames are saved, each named `000.pkl`, `001.pkl`, and so on, up to `039.pkl`.

Each `.pkl` file contains processed detections, including class, position, size, rotation, velocity, and confidence. It also holds true annotations, which include ID, category, size, and information about continuous motion. Frame metadata, such as ego vehicle translation, timestamp, and NuScenes token, are also included. Detection post processing involves filtering, projection, and optionally Non-Maximum Suppression (NMS) with an adjustable IoU threshold, which is typically 0.1, as shown in the function `simpletrack_nms()`. This step makes the system more robust against duplicate or overlapping box predictions before they are sent to the model.

The true annotations include not only current state information but also fields that are consistent moving forward in time, for example, `next_translation` and `next_yaw`. These enable the model to learn about the temporal consistency in how objects move. Instance tokens are changed into dense, continuous tracking IDs for each scene using a token to ID remapping process. The pre-processed scenes are grouped by their use, either for `training`, `validation`, or `testing`. Each scenario, such as `ground_truth`, `detection_results`, or `joint_training`, is saved under its own subfolder. This separation allows for flexible experiments without needing to re process shared data.

Since 3DMOTFormer is a learnable architecture, it creates and stores model weights under `results/models`, in the format `checkpoint-epoch_N.pth`, along with its `config.json`. The evaluation output structure under `val/` is exactly the same as the format used by the PolyMOT tracker, because this tracking output is a

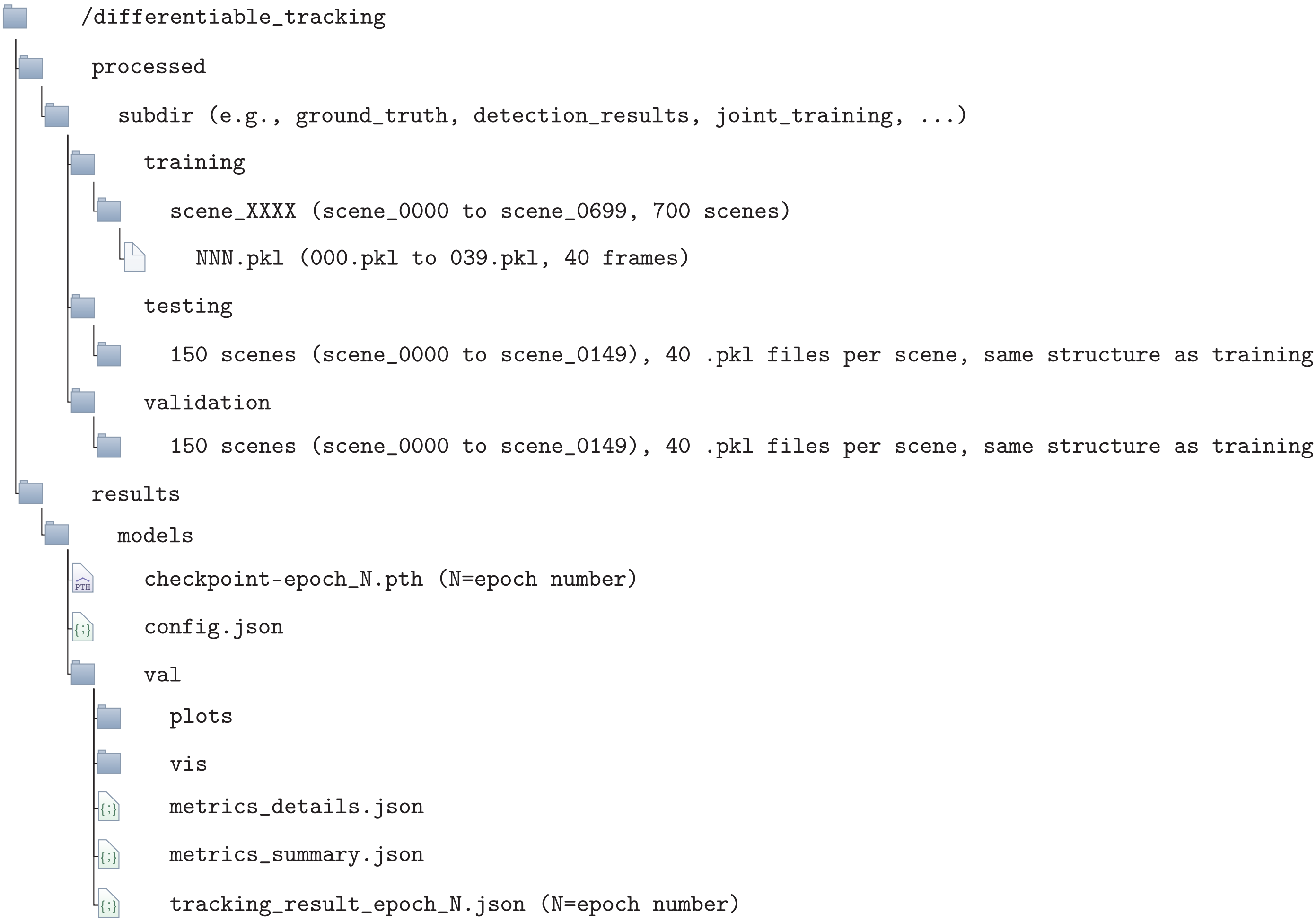

/differentiable_tracking
processed
subdir (e.g., ground_truth, detection_results, joint_training, ...)
training
scene_XXXX (scene_0000 to scene_0699, 700 scenes)
NNN.pkl (000.pkl to 039.pkl, 40 frames)
testing
150 scenes (scene_0000 to scene_0149), 40 .pkl files per scene, same structure as training
validation
150 scenes (scene_0000 to scene_0149), 40 .pkl files per scene, same structure as training
results
models
PTH
checkpoint-epoch_N.pth (N=epoch number)
{;}
config.json
val
plots
vis
{;}
metrics_details.json
{;}
metrics_summary.json
{;}
tracking_result_epoch_N.json (N=epoch number)

Figure 3.5: Directory structure of `$SHARED_VOLUME/differentiable_tracking`. Only a single representative configuration is expanded for clarity.

standard format required by NuScenes. This reuse of the evaluation structure ensures that both tracking baselines are compatible and simplifies scripts used for comparing them.

When 3DMOTFormer and Trajectron++ are trained together, only the tracking part of the results is stored under `/differentiable_tracking`. The prediction outputs, which are CSV-based ADE/FDE/MR metrics, are saved in the corresponding subfolder under `/prediction`, matching the standard prediction evaluation layout. This separation ensures that both tracking and prediction processes can be reused or analyzed independently.

3.3 Joint Tracking and Prediction via Differentiable 3DMOTFormer and Trajectron++

This section provides a detailed explanation of an integrated system for 3D multi-object tracking and trajectory prediction. The discussion begins with the system's overall architecture, followed by an explanation of the differentiable data association mechanism based on Sinkhorn matching. The process of performing soft track state updates is then described, highlighting how association probabilities influence track updates. The section also outlines how gradients flow through the tracker as part of the end-to-end learning process. Finally, the joint training pipeline is presented, demonstrating how both modules are optimized together within a unified framework.

3.3.1 System Overview

An integrated system that jointly performs 3D multi-object tracking and trajectory prediction in a differentiable pipeline is implemented. Figure 3.6 illustrates the overall architecture. First, a stream of 3D detections (e.g. bounding boxes from a LiDAR-based detector on nuScenes) is fed into a modified 3DMOTFormer[48] tracking module. 3DMOTFormer uses a graph-transformer to associate detections with existing tracks and update each track's state (position, velocity, and embedding). Unlike a conventional tracker, 3DMOTFormer is implemented to be differentiable now, specifically at the data association step, so that it can be trained end-to-end with a prediction model. The tracker's output at the current time step consists of updated track states (one per hypothesized object) including their latest estimated 3D bounding box and velocity. These track states are then passed into the Trajectron++ trajectory prediction module, which treats each tracked object as a node in a graph and forecasts its future motion. Importantly, the tracking and prediction modules are jointly trained: the prediction loss gradients propagate backward through the tracker. By making the tracker's internal operations differentiable, the entire tracking-prediction pipeline can be optimized as a single model. This approach addresses the common issue that prediction modules are often trained on ground-truth trajectories and thus not exposed to tracking errors. In this system, tracking and prediction are co-learned, allowing the tracker to improve in ways that directly benefit trajectory forecasting accuracy. In summary, the system takes detection inputs, produces continuous tracked trajectories, and outputs multi-step future predictions, all within one end-to-end trainable framework.

3.3.2 Differentiable Data Association: Sinkhorn Matching

A key novelty of this joint training approach is the differentiable data association in 3DMOTFormer[48]. In standard trackers, assigning detections to tracks is done with hard, discrete decisions (e.g. the Hungarian algorithm or greedy matching[38]). 3DMOTFormer originally relied on a greedy (score-driven) matching scheme and could also use Hungarian matching as an alternative. However, such hard associations are not differentiable, a small change in the input can flip a match decision, making it impossible to propagate

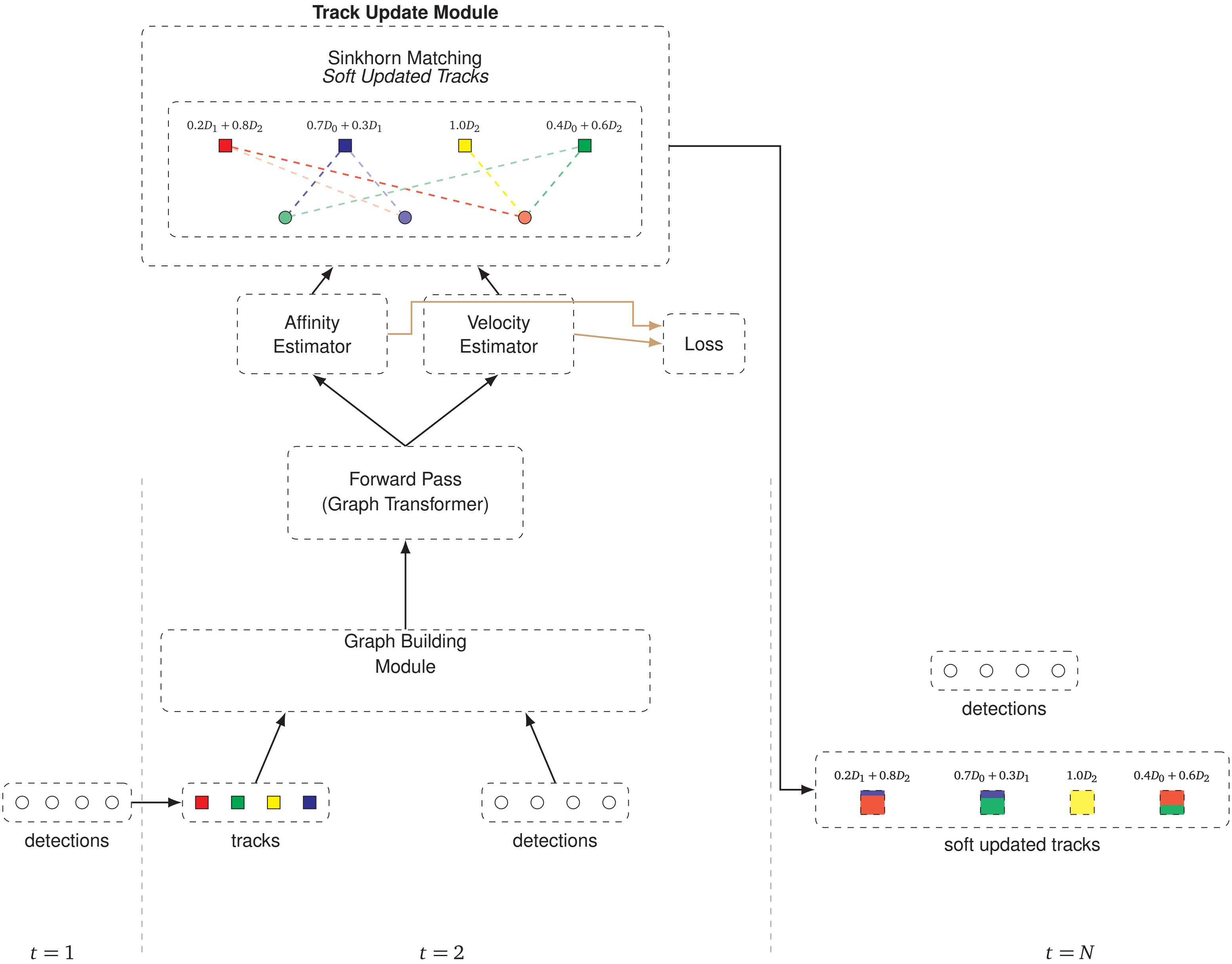


Figure 3.6: Differentiable end-to-end tracking and prediction architecture. The system combines a modified 3DMOT-Former for 3D multi-object tracking with Trajectron++ for trajectory prediction. The tracking module is made differentiable via a learnable Sinkhorn-based affinity matrix, enabling gradients from the prediction loss to flow back into the tracker. This unified pipeline allows co-training of both modules for improved long-term motion forecasting under realistic detection inputs.

gradients through this step. To enable end-to-end training, the original 3DMOTFormer's discrete matching is replaced with a Sinkhorn matching layer. The Sinkhorn algorithm produces a soft assignment, so instead of outputting a binary decision (each detection is either matched or not), it yields a continuous assignment matrix where each detection–track pair is assigned a probability or weight. This provides a differentiable relaxation of the matching problem. In effect, 3dmotformer goes from hard association (one-to-one matches) to soft association (many-to-many fractional matches via Sinkhorn[83]).

The rationale is that with soft associations, the tracker can treat each detection–track pairing with a graded confidence between 0 and 1. During training, the downstream trajectory prediction loss can backpropagate into these association weights, nudging the tracker to prefer those matchings that lead to better trajectory forecasts. In contrast, a hard assignment would create a non-smooth "winning" detection for each track (selected by an argmax operation), blocking any gradient flow. That way useful gradients cannot be computed through a discrete assignment since it's not a differentiable function of the affinity scores. By using Sinkhorn, it is ensured that the data association step remains within the computational graph, allowing the Trajectron++ loss to propagate backward into the tracker. This change is crucial for joint training, it transforms 3DMOTFormer from a modular tracker into a differentiable module that can be optimized as part of a larger network.

In summary, by adopting Sinkhorn-based differentiable matching, the data association step is converted from a non-differentiable combinatorial optimization into a continuous optimization problem that approximates the Hungarian solution. This is the foundation that makes the joint tracking-prediction training possible. Further details are provided below on how the Sinkhorn algorithm works and how it is integrated into track updates.

3.3.3 Sinkhorn Algorithm for Soft Matching

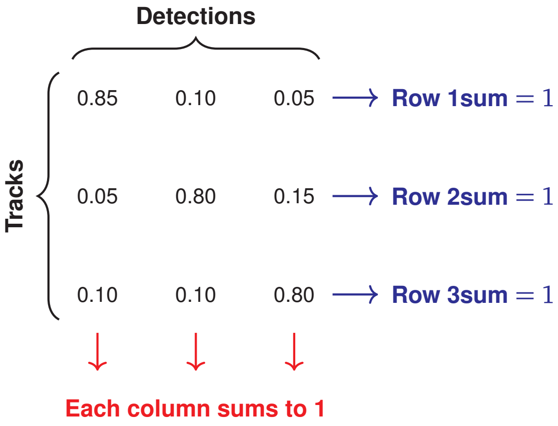
The Sinkhorn algorithm[83] is an iterative procedure that converts an arbitrary real-valued cost matrix into a doubly stochastic assignment matrix. In the context of this work, detection and tracks association, let N be the number of active tracks and M be the number of new detections at the current time. A cost matrix $C \in \mathbb{R}^{N\times M}$ is formed, where entry C_{ij} represents the "cost" of matching track i with detection j. This could be derived from 3DMOTFormer's affinity predictions or distance measures (for example, one could use $C_{ij} = -\text{affinity}_{ij}$ so that a high affinity (good match) corresponds to a low cost). Solving the optimal assignment (as Hungarian would) is equivalent to finding a permutation matrix X that minimizes $\langle C, X\rangle$ subject to $X \in 0, 1^{N\times M}$ with each row and column at most 1 (and typically exactly one match per track when $N = M$). Sinkhorn relaxes this by allowing X to take continuous values in $[0, 1]$ and enforcing a doubly stochastic constraint (i.e., $X\mathbf{1} = \mathbf{1}$ and $X^T\mathbf{1} = \mathbf{1}$), ensuring that each row and column sums to one. This means that each track distributes its association probability across detections, and vice versa, leading to a soft matching. Figure 3.7 illustrates such a matrix example: diagonal values are dominant, but small weights are still assigned to alternative matches, allowing gradient flow and uncertainty modeling during training.

The algorithm introduces an entropy regularization term into the objective, as first popularized by Cuturi (2013)[83]. The regularized objective can be written as:

$$\min_{P\in\mathcal{U}} \langle C, P\rangle - \frac{1}{\lambda} H(P),$$

where

$$\mathcal{U} = \left\{ P \in \mathbb{R}^{N\times M}_{\geq 0} \;\middle|\; P\mathbf{1}_M = \mathbf{1}_N,\; P^\top \mathbf{1}_N = \mathbf{1}_M \right\}$$



$$\textbf{Tracks}\left\{\begin{array}{ccc}
\multicolumn{3}{c}{\overbrace{\hspace{8em}}^{\textbf{Detections}}}\\
0.85 & 0.10 & 0.05 \\
0.05 & 0.80 & 0.15 \\
0.10 & 0.10 & 0.80 \\
\downarrow & \downarrow & \downarrow
\end{array}\right.
\begin{array}{l}
\\
\longrightarrow \textbf{Row 1sum} = 1 \\
\longrightarrow \textbf{Row 2sum} = 1 \\
\longrightarrow \textbf{Row 3sum} = 1 \\
\
\end{array}$$

Each column sums to 1

Figure 3.7: Soft assignment matrix after Sinkhorn iterations, showing an approximately doubly stochastic matrix. Each row corresponds to a track, each column to a detection. Values indicate soft assignment probabilities.

is the set of doubly stochastic matrices (assuming $N = M$ for simplicity; if not, an unassigned "dummy" row or column is typically added),

$$H(P) = -\sum_{i,j} P_{ij} \log P_{ij}$$

is the entropy of matrix P, and $\lambda > 0$ is a regularization parameter. The entropy term encourages P to have higher entropy (i.e. be more spread-out) unless the cost strongly favors a particular assignment. The solution P^* to this problem can be found efficiently by Sinkhorn's iterative scaling.

Initialization, compute $K = \exp(-C/\lambda)$ elementwise. This yields a positive matrix where larger weight is given to lower-cost pairs (this exponential step is essentially a softmax on the negative costs). Row normalization, normalize K along rows: for each row i, divide by its sum. This yields a matrix K' where each row now sums to 1. Column normalization, normalize K' along columns: for each column j, divide by its sum, producing K'' with each column summing to 1. Iterate, assign $K \leftarrow K''$ and repeat the row and column normalization steps until convergence.

In practice, the above is implemented by maintaining scaling vectors for rows and columns. One can write this more compactly as: find diagonal matrices U and V such that $P = UKV$ is doubly stochastic. The Sinkhorn-Knopp theorem[94] guarantees that for any positive matrix K, iterative normalization will converge to a unique doubly stochastic matrix P (provided a solution exists). The resulting matrix P is the soft assignment matrix. Each entry P_{ij} lies between 0 and 1 and can be interpreted as the fractional association between track i and detection j. The matrix is approximately binary in the limit of small λ, but remains differentiable for any finite λ.

Crucially, every operation in the Sinkhorn algorithm is differentiable. The exponential and division (normalization) steps have well-defined gradients. Thus, the output matrix P is a smooth function of the input costs C. Small changes in C_{ij} produce small changes in P_{ij}, which means $\frac{\partial P}{\partial C}$ can be computed and ultimately backpropagate through the matching. In essence, Sinkhorn provides a "softmax" analog of the assignment problem. This differentiability is what allows the tracker's association decisions to be influenced by the trajectory prediction loss. The entropy term $\frac{1}{\lambda}\sum P_{ij} \log P_{ij}$ ensures that the assignment remains soft enough for gradients to flow, preventing the solution from becoming overly discrete during training.

Mathematically, after L Sinkhorn iterations, an assignment matrix $P^{(L)}$ is obtained, that is nearly doubly stochastic. This P is used instead of a hard matching. During training, P is not discretized, which allows the network to learn appropriate fractional assignments. At inference time, one could either take the highest P_{ij} per track as the match (essentially recovering a greedy/Hungarian assignment if needed) or even use the fractional P directly for a multi-detection update (though in practice P tends to be almost one-hot due to the nature of the costs), so both ways are equivalent in this case. In summary, the Sinkhorn algorithm produces a matrix P that approximates the optimal matching while remaining continuous and differentiable, which is why it is often called a differentiable approximation of the Hungarian algorithm.

3.3.4 Soft Track State Updates

In this section, the way how sinkhorn matching is implemented within the existing 3dmotformer model is explained. 3DMOTFormer (Ding et al., 2023)[48] was originally trained independently of any prediction module, using tracking-specific supervision. Similarly, Trajectron++[55] was trained separately on ground truth trajectories. In typical deployment, tracking and prediction run sequentially without any learned feedback between them. In the modified 3DMOTFormer, the core graph transformer architecture is retained, but the association and update steps are redesigned to support end-to-end training. The key change is replacing Hungarian matching with the differentiable Sinkhorn algorithm, enabling gradient flow through the tracking module. Additionally, the tracker now incorporates trajectory prediction loss as part of its training objective. Unlike the original version, which focused solely on tracking metrics, the modified 3DMOTFormer is jointly optimized for both tracking and prediction performance.

This leads to differences in training behavior. Instead of optimizing only for short-term tracking accuracy using MOT metrics, the tracker now learns to support long-term motion consistency that benefits forecasting. For example, it may prefer maintaining identity continuity, even with lower confidence detections, if that improves trajectory prediction. This marks a shift from a modular setup to an integrated system, where tracking is learned as part of the overall prediction objective.

With the Sinkhorn algorithm delivering a soft association matrix P, 3DMOTFormer's track update mechanism is needed to be modified to utilize these soft matches. In a traditional tracker, if track i is matched to detection j, the tracker would update track i's state (e.g. its estimated position and velocity) using detection j's observed state; if no match is found, the track might be predicted forward or marked lost. In this work's soft-update scheme, each track can be partially matched with multiple detections (or none, in which case a "dummy" detection with no observation can be considered). The update is therefore formulated as a convex combination of detection inputs weighted by P. Figure 3.8 illustrates this update mechanism, where each track receives information from multiple detections weighted by the Sinkhorn-assigned probabilities.

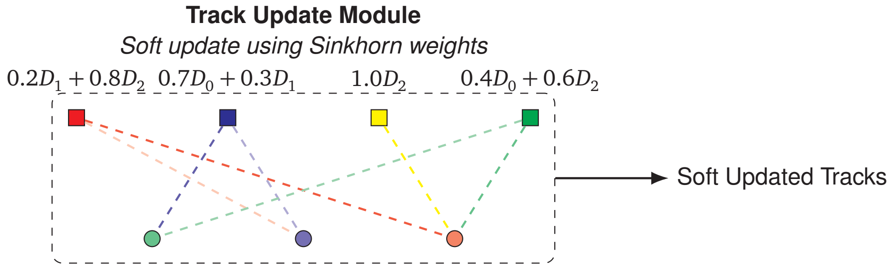


Figure 3.8: Track state updates using soft assignments from the Sinkhorn algorithm. Each track receives a weighted combination of detection inputs rather than a hard one-to-one match.

Concretely, let track i at the previous time have an internal state vector (which may include its last known position, velocity, and a learned embedding encoding its appearance or motion history). Let $d_1, d_2, \ldots, d_M$ be the detections at the current frame, each with observed properties (e.g. 3D box coordinates, detection confidence, possibly appearance features from the LiDAR/camera). After running the Sinkhorn solver, row i of the assignment matrix: $(P_{i1}, P_{i2}, \ldots, P_{iM})$ is obtained. These weights are used to blend the detection information and update track i. For any track state component that needs updating (such as the track's estimated position or its feature embedding), a weighted sum over all detections is computed.

Position/Box update: $\mathbf{x}i^{\text{new}} = \sum j = 1^M P_{ij}, \mathbf{x}d_j$, where $\mathbf{x}d_j$ is the observed 3D center (or full box parameters) of detection j. This effectively averages the detection positions, but since P is usually peaked on one detection for a given track, $\mathbf{x}_i^{\text{new}}$ ends up close to that detection's position. In edge cases like two close detections with similar confidence, the track might interpolate between them, the network could learn to distribute P if unsure, and the resulting position is a convex combination of both possibilities.

Velocity update: Similarly, since the detector provides an estimated velocity, the track's velocity is updated as $\mathbf{v}i^{\text{new}} = \sum_j Pij, \mathbf{v}_{d_j}$. Even if detection velocities are noisy, the network can learn to adjust P weights to prioritize the more reliable detection. If no detector velocity is given, the track's motion model (e.g. linear motion from previous frames) could be incorporated as an additional "detection" candidate (representing the predicted motion with some cost) so that P can also assign weight to "no observation" and just carry the track forward.

Embedding update: 3DMOTFormer maintains a latent embedding for each track (carried in the transformer's track queries) to represent the object's identity and motion context. This embedding is updated by fusing it with the new detection's embedding. For example, the previous track embedding e_i^{old} can be taken and each detection's learned feature f_j (extracted by the detector or by the 3DMOTFormer encoder) and a weighted sum: $e_i^{\text{new}} = \sum_j P_{ij}, g(e_i^{\text{old}}, f_j)$ can be computed, where g could be a function such as concatenation and a small neural network, or simply the detection feature if we assume the transformer decoder already integrated the old track state in computing affinities. In this implementation, the transformer decoder in 3DMOTFormer already uses cross-attention between track queries and detection features, so one can interpret that the final edge features $h_{A,ij}$ (which lead to affinity a_{ij}) encode how to mix track and detection info. Once P_{ij} is obtained, essentially it is directly applied those weights to update the track query representations.

The phrase "soft track updates" means that each track's state is not updated from a single detection but from a mixture of detections. P_{ij} acts like an attention weight: it tells us how much detection j should contribute to track i's new state. Because $\sum_j P_{ij} = 1$ (each track's row normalizes to 1), this update is a convex combination of detection states and thus stays within a physically plausible range. If a track truly corresponds to a particular object in the scene, ideally one detection (the correct one) will have $P_{ij} \approx 1$ and others very close to 0 (but not 0), resulting in a near-hard update. But during training, when the network is uncertain or still learning, it can distribute some weight to alternate detections, allowing gradient signals to indicate which detection would have ultimately led to a better trajectory prediction. Over time, the network learns to make P peaked (approach a one-hot assignment) for confident matches. This effect is visualized in Figure 3.9, where each updated track is shown as a combination of detection features based on its assignment weights.

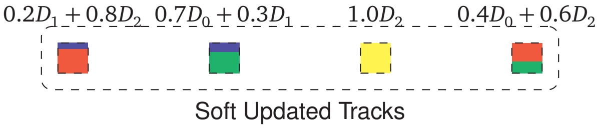


Figure 3.9: Each track is updated using a convex combination of detections, visualized here as color-coded contributions. The proportions reflect the Sinkhorn weights P_{ij} applied to detection inputs.

In cases of unmatched tracks or false detections, they are handled with dummy assignments, to make sure any format errors would not occur. An extra column is appended to C (and thus an extra column in P) representing a null detection (no match). This column's cost is set such that if a track has no reasonable detection, Sinkhorn can place most of that track's row weight on the null column (meaning the track will not get an actual detection update, similar to track persisting with its motion model). Likewise, extra detections that don't match any track can be handled by adding an extra row for "new track" or simply by allowing some detection columns to not receive full weight from any track (the doubly stochastic constraint can be relaxed to sum of rows ≤ 1 or using unbalanced OT for creation/termination, but for simplicity, it is assumed that track count roughly equals detection count in each training snippet, as new tracks can be initialized separately). These implementation details aside, the core idea is that soft matching weights drive the update of track states. Because this update rule is a simple differentiable computation (matrix-vector multiplications, etc.), it allows gradients to propagate from track states back to the assignment probabilities P and further back into the affinity prediction network.

3.3.5 Gradient Flow Through the Tracker

With the soft association and soft update mechanism in place, a fully differentiable tracking module is achieved. The following section, explains how gradients propagate from the trajectory prediction module back into the tracking component. After a forward pass through both modules, trajectory prediction loss is computed, for instance, the average displacement error (ADE) and final displacement error (FDE) between the predicted trajectories and the ground-truth future positions, or a negative log-likelihood if Trajectron++ outputs a probability distribution over future trajectories. This loss measures how well the entire system's output matches the true future trajectories of the objects. Crucially, the predicted trajectories depend on the tracker's outputs. If the tracker makes a mistake, say, it slightly mispositions a track, or even worse, mismatches an identity (assigns the wrong detection to a track), the trajectory predictor will be given an incorrect history for that track, and its forecast is likely to deviate from the ground truth, incurring a higher loss.

Because the tracker's final track states (at the last observed time) directly influence the prediction, the gradient of the prediction loss L_{pred} with respect to each track state will be non-zero when errors occur. For example, suppose track i's final position $\mathbf{x}i$ was used as the starting point for prediction. If the predictor's output trajectory for track i ended up offset from the ground truth, since the prediction and tracking loss are summed in the end, the loss gradient $\frac{\partial L\text{pred}}{\partial \mathbf{x}_i}$ might point in a direction that would align the track's position better with the true agent position. In a traditional setting, the tracker's output $\mathbf{x}_i$ would be treated as a fixed input to the predictor (no gradient flows into it). Thus, the gradient $\frac{\partial L\text{pred}}{\partial \mathbf{x}_i}$ will propagate backward through the track update equation.

If $\mathbf{x}i^{\text{new}} = \sum_j Pij, \mathbf{x}d_j$ as described, then the gradient w.rt Pij is $\frac{\partial L}{\partial P_{ij}} = \frac{\partial L}{\partial \mathbf{x}i^{\text{new}}} \cdot \mathbf{x}d_j$ (dot product of the upstream gradient with the detection state). Intuitively, this means if the loss would be reduced by track i placing more emphasis on detection k (maybe because detection k was the correct match and yields a trajectory closer to ground truth), the gradient will increase P_{ik} and decrease P_{ij} for other j. In other words, the network learns to adjust the assignment weights to favor those assignments that lead to accurate predictions. Similarly, gradients flow into the track's embedding update: if the prediction error indicates that the track's latent state was not informative enough (perhaps the wrong identity or poor velocity estimate), gradients will propagate into the parts of the tracker that produced that state, including the attention layers that computed affinities and the Sinkhorn layer that mixed the information. The differentiable matching ensures that even the decision of which detection to associate with a track can be influenced by the end loss. This is a form of signal passing back from forecasting to data association. In effect, the system can learn to perform data association in a way that not only matches objects over time, but specifically in a way that helps predict their future motion correctly.

A concrete example helps illustrate this: imagine two vehicles A and B that are close together at the crossing, and the tracker isn't sure which detection belongs to which track (a potential ID switch). If the tracker chooses incorrectly (swaps them), the predictor will likely forecast unrealistic trajectories (each vehicle's past motion history doesn't match its assigned identity, so predictions could veer the wrong way). This will result in a large trajectory error. The gradient through this pipeline will then push the tracker's affinity network to lower the score of that mismatched pairing and increase the score for the correct pairing, because assigning correctly would have resulted in a much lower prediction error. Over training, this process teaches the tracker to avoid identity switches, since consistent tracking yields better prediction loss. In summary, gradients from trajectory forecasting supervise the tracking: errors in the forecast drive corrections in track association and state estimation. This is a novel feedback mechanism absent in traditional separated training. Notably, prior works that attempted to link tracking and prediction often did so without direct end-to-end gradient coupling, e.g., by alternating optimization or using prediction as an aid in inference. In contrast, this approach truly integrates the gradient flow, making the tracker a learnable component optimized for overall forecasting quality.

3.3.6 Joint Training Pipeline

Architecturally, the core designs of 3DMOTFormer and Trajectron++ are preserved, but a differentiable interface is introduced between them, enabling this end-to-end gradient flow. It's important to note that no deep internal changes were needed in Trajectron++, the innovation lies in the tracker's data association and the training scheme itself. Consequently, this joint model can be seen as 3DMOTFormer feeding into Trajectron++ in an end-to-end differentiable manner, whereas originally they would be used sequentially but trained independently.

In this joint training model, both 3DMOTFormer and Trajectron++ use their respective data loaders. For each batch from Trajectron++, the corresponding scene and time are used to construct a mini-sequence of frames at a specific timestep t in a scene, which is processed by 3DMOTFormer. The tracker maintains and updates internal track states across these frames and outputs the final agent states (e.g., positions and velocities) at the last timestep of each sequence.

These final tracker states are then used to overwrite the last frame in Trajectron++'s input tensor, replacing the (x, y, v_x, v_y) values of each corresponding agent at time t, because the final output already reflects all prior associations and updates made by the tracker. This setup enables Trajectron++ to learn robustness to tracking noise at the prediction boundary, and allows gradients to flow back into 3DMOTFormer as well, aligning tracking behavior with forecasting performance. Importantly, this approach keeps Trajectron++'s architecture and data pipeline untouched, treating 3DMOTFormer's output as a drop-in replacement for the final observation in the input sequence.

Then Trajectron++ simply processes the sequence it's given, 'unaware' that one of those points came from a learnable tracker. The predictor proceeds to compute interactions (same as standard Trajectron++: building a k-Nearest Neighbor or radius-based graph of which agents might interact, then performing graph neural network message passing, etc.) and then generates future trajectories for each agent node. Because both models are implemented in PyTorch[90], as long as the input tensor for the final frame is connected to the tracker's output, the autograd mechanism carries the gradients backwards.

Finally, the total loss (Prediction loss + tracking loss) is backpropagated through both modules. Thanks to the differentiable Sinkhorn layer and recurrent track updates, the gradient flows from the prediction loss into the tracker's affinity and update mechanisms, allowing the full pipeline, including the transformer's attention layers and the trajectory forecasting GNNs—to be trained jointly. Both optimizers (for 3DMOTFormer and Trajectron++) are stepped in sync, and learning rate schedules are updated accordingly. This loop results in a fully end-to-end trainable system, where the tracker and predictor co-adapt to improve downstream trajectory forecasting performance.

4 Results

This chapter presents the experimental results obtained from evaluating the different components and paradigms of the whole perception system. Consistent with the methodology, this section focuses solely on the objective presentation of quantitative metrics and qualitative observations for each stage, reserving detailed analysis, interpretation, and comparison of these findings for the subsequent Discussion and Conclusion chapters.

4.1 Detection Results

Two variants of the CenterPoint detector were evaluated, distinguished primarily by their voxel size configurations, which process the raw LiDAR point cloud data. The quantitative performance of the CenterPoint models was assessed using the standard nuScenes detection metrics: mean Average Precision (mAP) and the nuScenes Detection Score (NDS). These metrics provide a comprehensive numerical evaluation of a detector's ability to precisely localize and correctly classify objects. Table A.1 in Appendix A. summarizes the performance achieved by both CenterPoint models across key object categories as well as overall.

Figure 4.1 visually represents these quantitative results, illustrating the mAP and NDS scores for both CenterPoint models (Voxel=0.075 and Voxel=0.1) across the 'Vehicle', 'Pedestrian', and 'Overall' categories.

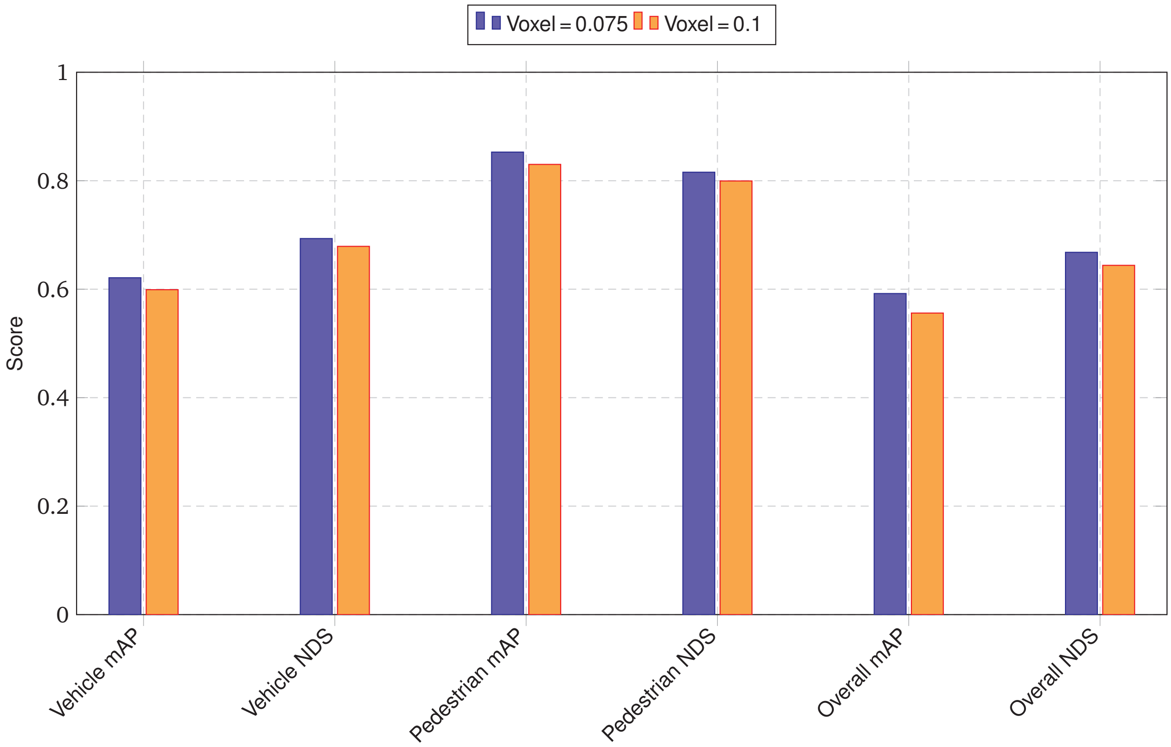


Figure 4.1: CenterPoint performance on nuScenes for voxel sizes 0.075 and 0.1 across mAP and NDS metrics.

Beyond numerical metrics, a qualitative assessment of detection performance provides crucial visual insights into the models' behavior and capabilities in real-world driving scenarios. To complement the quantitative results, visual comparisons of detection outputs from the two CenterPoint models were conducted on identical sample scenes. Figure 4.2 illustrates these qualitative aspects on two representative frames from the nuScenes validation set. These figures visually depict the generated 3D bounding boxes by both detection models with different voxel sizes, their associated class labels, and ground truth boxes side-by-side. Such visual inspections helps to compare their detection performance better. Left side represents the CenterPoint model with voxes size 0.075 while the right side shows a visualization of the model with voxel size 0.1. The highlighted boxes with black in the right frame, show the difference between performances.

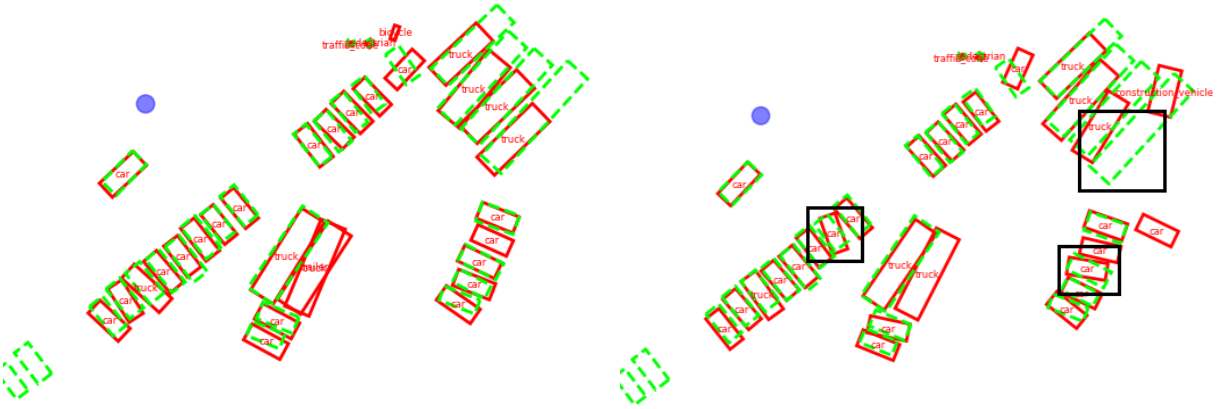


Figure 4.2: Side-by-side comparison of detection outputs from two CenterPoint models. Green boxes highlights ground truth locations, the red boxes highlights the detected locations. Region where performance differs are highlighted in the second picture with black boxes.

4.2 Tracking Results

The tracking stage builds upon the detection outputs by establishing consistent object identities across frames, forming trajectories. For all subsequent tracking and prediction evaluations, only the detection results from the CenterPoint[20] model with a voxel size of 0.075 were utilized. This decision was based on its superior quantitative performance observed in the detection evaluation, allowing to focus on the most performant detection input for downstream tasks and to streamline the overall number of results.

This section presents the performance of PolyMOT[39], a non-learnable tracker, under two distinct input conditions: when processing detections from the selected CenterPoint model and when given ground truth bounding boxes. The quantitative performance of PolyMOT was assessed using the nuScenes tracking metrics: Average Multi-Object Tracking Accuracy (AMOTA), Average Multi-Object Tracking Precision (AMOTP), and ID Switches (IDS).

Detailed tracking metrics for PolyMOT with both detector and ground-truth inputs are provided in Appendix A, in tables A.2 and A.3. Figure 4.3 visually compares these quantitative tracking results, illustrating the AMOTA, AMOTP, and IDS values for PolyMOT when operating on CenterPoint detections (voxel size 0.075) versus ground truth detections across the 'Vehicle', 'Pedestrian', and 'Overall' categories.

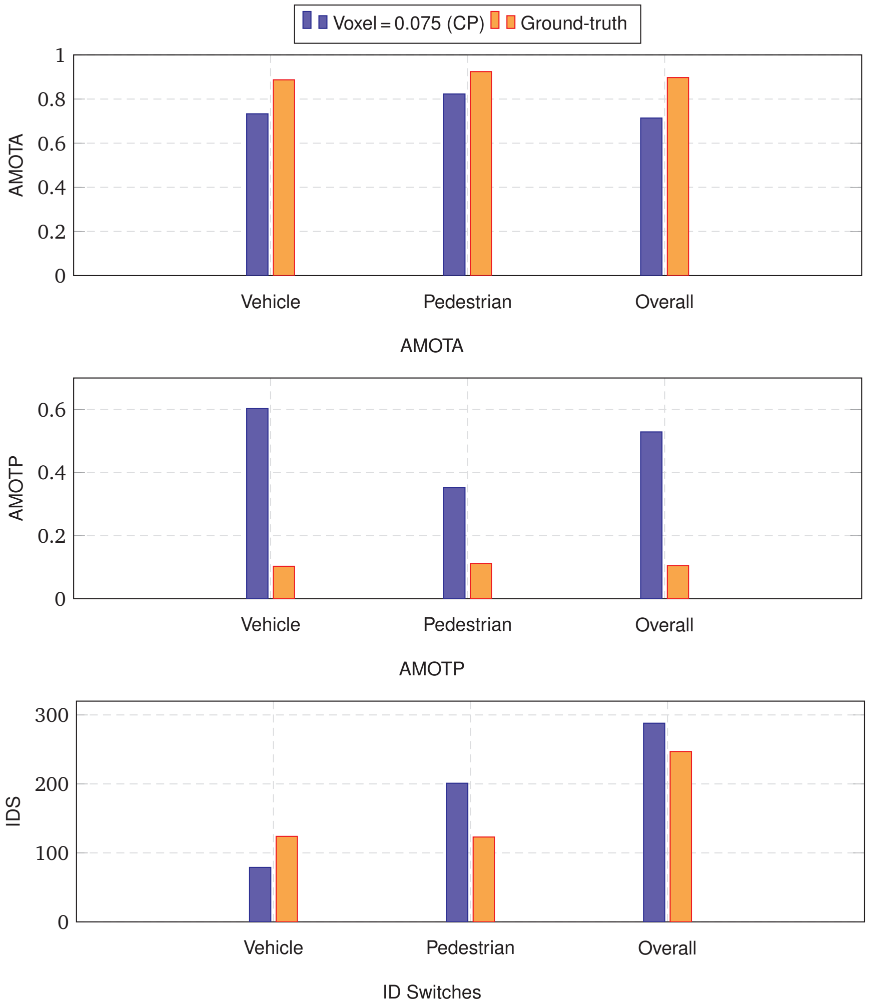


Figure 4.3: PolyMOT tracking performance on the nuScenes validation set using CenterPoint detections (voxel size 0.075) and ground-truth detections.

4.3 Learning-Based Tracking Results

This section presents the results from evaluating 3DMOTFormer[48], a learning-based tracking model. Its performance is assessed under various conditions, reflecting different training data sources and evaluation inputs. Specifically, the scenarios where 3DMOTFormer is trained on data derived from CenterPoint[20] detections, trained on ground truth data, and integrated into a joint tracking and prediction framework are investigated.

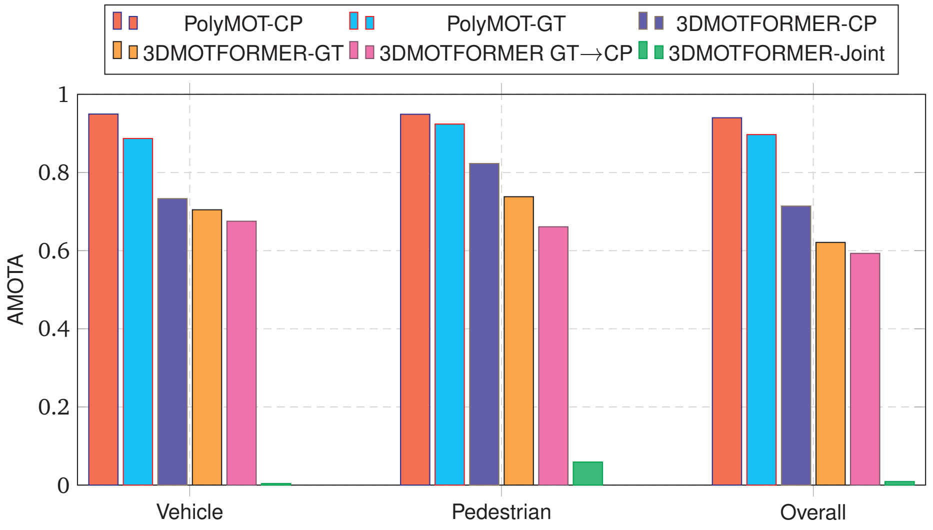
Table A.4 presents the tracking performance of 3DMOTFormer, which was trained using data derived from CenterPoint (voxel size 0.075) detections, and then evaluated on CenterPoint (voxel size 0.075) detections from the nuScenes validation set.

Table A.5 shows the performance of the 3DMOTFormer model trained and evaluated using ground truth detections on the nuScenes validation set.

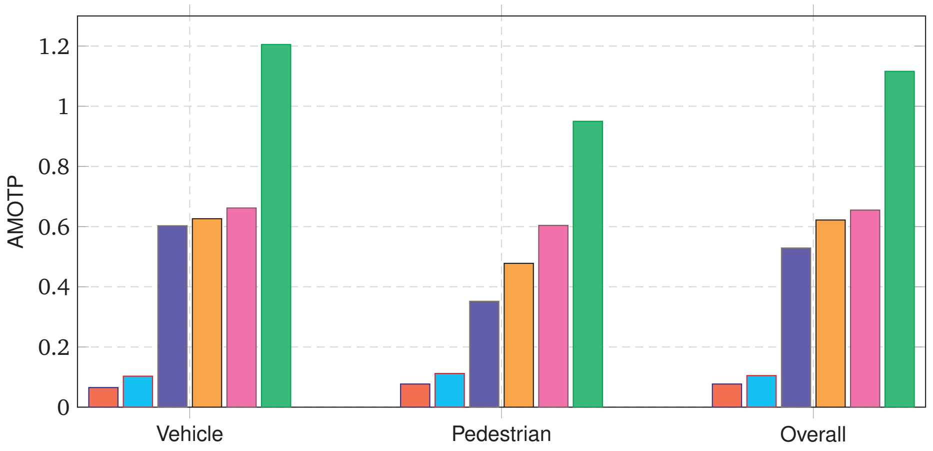
Table A.6 details the tracking performance of the same previous 3DMOTFormer model, that was trained using ground truth data, but then evaluated on CenterPoint (voxel size 0.075) detections.

Table A.7 provides the tracking performance metrics from the intermediate tracking output of a 3DMOTFormer model trained as part of a joint (tracking + prediction) end-to-end framework, evaluated on CenterPoint (voxel size 0.075) detections.

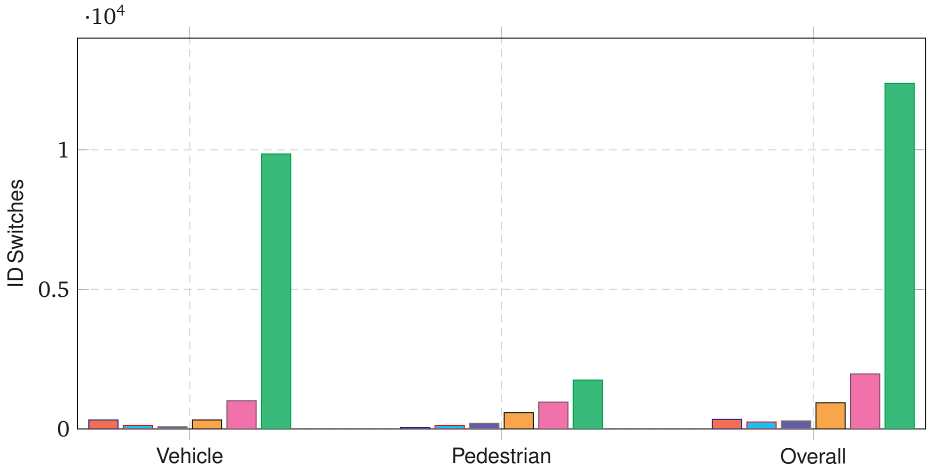
Figure 4.4 provides a consolidated visual representation of all these tracking results, including both the non-learnable PolyMOT[39] tracker (evaluated on CenterPoint detections and ground truth detections, as presented in Section 4.2) and the four scenarios for the learnable 3DMOTFormer tracker described above, ranked from the best to worst (left to right). This figure illustrates the AMOTA, AMOTP, and IDS metrics across ’Vehicle’, ’Pedestrian’, and ’Overall’ categories, allowing for a comprehensive overview of the tracking performance under different algorithmic and input conditions.



(a) AMOTA



(b) AMOTP



(c) ID Switches (IDS)

Figure 4.4: Comprehensive comparison of tracking performance across PolyMOT and 3DMOTFormer variants on the nuScenes validation set.

4.4 Prediction Results

The prediction stage is the final component of the comprehensive perception pipeline, responsible for forecasting the future trajectories of detected and tracked objects. For all prediction evaluations where upstream detections were required, the outputs from the CenterPoint model with a voxel size of 0.075m were exclusively utilized, consistent with its selection as the most performant detector. This section presents the quantitative results obtained from various configurations of the prediction module, both as a standalone component and as part of end-to-end sequential and joint pipelines.

4.4.1 Trajectron++ Baseline Performance

To establish a foundational understanding of the prediction model's intrinsic capabilities and the metrics used for evaluation, first the performance of Trajectron++, trained and evaluated solely on ground truth data is presented. This serves as a baseline, illustrating the best-case scenario for the prediction model in the absence of upstream detection and tracking errors.

The prediction performance is evaluated using Average Displacement Error (ADE), Final Displacement Error (FDE), and Miss Rate, reported at both 4-second and 6-second prediction horizons. These metrics are presented for two primary object classes: Vehicles and Pedestrians.

Table 4.1: (a) Vehicle results from ground truth input at 4s and 6s

VEHICLE

Ablation			**ADE (ml)**		**FDE (ml)**		**MissRate (%)**	
$\int$	M	y_R	@4s	@6s	@4s	@6s	@4s	@6s
-	-	-	**1.3387**	**2.4588**	**2.9490**	**5.8571**	**43.58**	53.09
✓	-	-	1.6170	3.2598	3.8029	8.3940	50.56	55.97
✓	✓	-	1.4100	2.6266	3.1532	6.3282	45.71	54.27
✓	✓	✓	1.3155	2.4645	2.9388	5.9791	42.01	**49.46**

Legend: $\int$ = Integration via Dynamics, M = Map Encoding, y_R = Robot Future Encoding

Table 4.1 provides the prediction results for vehicle trajectories, obtained when Trajectron++ is trained and evaluated using ground truth inputs. The table also includes an ablation study detailing the impact of different architectural components of Trajectron++, namely integration via dynamics ($\int$), map encoding (M), and robot future encoding (y_R), on the prediction accuracy.

Table 4.2: (a) Pedestrian results from ground truth input at 4s and 6s

PEDESTRIAN

Ablation			**ADE (ml)**		**FDE (ml)**		**MissRate (%)**	
$\int$	M	y_R	@4s	@6s	@4s	@6s	@4s	@6s
-	-	-	**0.4213**	**0.6789**	**0.8276**	**1.4267**	**9.79**	**23.23**
✓	-	-	0.4416	0.7164	0.8730	1.5170	10.69	25.75
✓	✓	-	0.4349	0.7056	0.8567	1.4991	10.49	25.12

Legend: $\int$ = Integration via Dynamics, M = Map Encoding

Similarly, Table 4.2 showcases the prediction results for pedestrian trajectories, also derived from ground truth inputs. This table likewise includes an ablation study, focusing on the comparisons if integration via dynamics and map encoding is added, or robot future encoding is used.

4.4.2 Prediction Performance by Tracking Input and Ablations

This section provides an overview of the trajectory prediction performance for both vehicle and pedestrian classes. Prediction was performed using the Trajectron++ model, which was trained and evaluated under several input tracking conditions and ablation configurations.

For each agent type, prediction was evaluated using two types of tracking outputs: non-learning-based (CenterPoint with a voxel size of 0.075) and learning-based (3DMOTFormer). These tracking outputs were used as inputs to Trajectron++, which was evaluated separately for each condition. In addition to baseline prediction, ablation studies incorporating different auxiliary encodings were performed: dynamics integration ($\int$), map encoding (M), and robot future encoding (y_R). For each configuration, results at two prediction horizons: 4 seconds and 6 seconds are reported.

The full numerical results are provided in Appendix A. These include vehicle prediction outcomes using non-learning-based tracking inputs, reported in Tables A.8 and A.9, as well as results using learning-based tracking inputs, shown in Tables A.10 and A.11. Corresponding pedestrian prediction results are also included: Tables A.12 and A.13 contain the non-learning-based outputs, while Tables A.14 and A.15 present results using learning-based tracking inputs.

The results in the appendix show how prediction performance varies depending on the type of tracker used and the presence of each auxiliary module. These findings will be analyzed and compared in detail in the Discussion chapter.

4.4.3 Full Perception Pipeline Overview

To provide a consolidated view of the whole perception pipeline's performance, the prediction results from various sequential and joint training setups are presented in a comprehensive format. This approach was chosen to maintain clarity and avoid presenting numerous individual tables for each specific combination of detection, tracking, and prediction models.

Table 4.3 presents the overall pipeline evaluation for Vehicle prediction. Each row corresponds to a distinct configuration, outlining the upstream detection and tracking methods used, followed by the resulting prediction metrics (ADE, FDE, Miss Rate at 4s and 6s). This includes scenarios integrating results from ground truth, the non-learnable PolyMOT tracker, and different variants of the learnable 3DMOTFormer tracker, as well as the directly evaluated joint training (tracking + prediction) model. It is important to note that for scenarios where upstream tracking performance was not done (e.g., the 3DMOTFormer model trained on ground truth but evaluated on CenterPoint detections, because it was found to be unnecessary since tracking results were extremely low and it would not make sense to follow up the analysis), prediction results are indicated as not applicable ('/').

Analogously, Table 4.4 details the overall pipeline evaluation for Pedestrian prediction, following the same comprehensive format. This table similarly captures the cascading effects of various upstream detection and tracking inputs on the final prediction accuracy for pedestrian trajectories.

For each prediction evaluation included in these overview tables, only the results from the base Trajectron++ model (i.e., without auxiliary encodings) are reported. This decision is based on the consistent observation across the detailed ablation study that the base model yielded superior performance, particularly when using ground truth inputs, making it the most representative for pipeline-level comparison.

As mentioned before, for a more granular breakdown of prediction metrics for all models and configurations, including additional detailed tables, please refer to the Appendix A at the end of this thesis.

Vehicle prediction results

Table 4.3: Overall vehicle pipeline comparison. Each row corresponds to a different input setup (e.g., ground-truth vs. detector output). The following metrics are reported (a) Detection metrics: mAP, NDS, (b) Tracking metrics: AMOTA, AMOTP, ID switches (IDs), and (c) Prediction metrics: ADE, FDE, and MissRate at 4s and 6s. Abbreviations: D = Detection, T = Tracking, P = Prediction.

Setup	Detection		Tracking			Prediction					
	mAP	NDS	AMOTA	AMOTP	IDs	ADE (m)		FDE (m)		MissRate (%)	
						@4s	@6s	@4s	@6s	@4s	@6s
GT	—	—	—	—	—	1.3387	2.4588	2.9490	5.8571	43.58	53.09
GT→T→P	—	—	88.75	10.53	124	1.3647	2.5113	2.9791	5.9881	44.01	53.55
D→T→P	62.1	69.3	73.4	60.3	79	1.6585	2.9609	3.5524	6.8904	49.58	58.49
D→T→Fine tuned P	62.1	69.3	73.4	60.3	79	1.6780	3.0686	3.6329	7.3062	49.97	58.64
D→T(Learn. based)→P	62.1	69.3	70.45	62.6	322	1.7908	3.1460	3.7701	7.2290	51.56	60.49
D→T(Learn. based)→Fine tuned P	62.1	69.3	70.45	62.6	322	1.8104	3.2583	3.8479	7.6698	52.09	60.71
GT→T(Learn. based Train on GT)→P	—	—	94.95	6.5	287	1.3443	2.4676	2.9588	5.8759	43.64	53.21
D→T(Learn. based Train on GT)→P	62.1	69.3	0.9	111.6	12381	/	/	/	/	/	/
JOINT TRAINING(T+P) RESULTS	62.1	69.3	67.52	66.2	1009	2.0572	3.5511	4.2450	8.0459	54.00	62.27

Pedestrian prediction results

Table 4.4: Overall pedestrian pipeline comparison. Each row corresponds to a different input setup (e.g., ground-truth vs. detector output). The following metrics are reported (a) Detection metrics: mAP, NDS, (b) Tracking metrics: AMOTA, AMOTP, ID switches (IDs), and (c) Prediction metrics: ADE, FDE, and MissRate at 4s and 6s. Abbreviations: D = Detection, T = Tracking, P = Prediction.

Setup	Detection		Tracking			Prediction					
	mAP	NDS	AMOTA	AMOTP	IDs	ADE (m)		FDE (m)		MissRate (%)	
						@4s	@6s	@4s	@6s	@4s	@6s
GT	—	—	—	—	—	0.4213	0.6789	0.8276	1.4267	9.79	23.23
GT→T→P	—	—	92.4	11.2	123	0.4330	0.6940	0.8450	1.4550	10.10	23.70
D→T→P	85.3	81.6	82.3	35.2	201	0.6690	0.9896	1.1937	1.9070	18.13	33.66
D→T→Fine tuned P	85.3	81.6	82.3	35.2	201	0.7445	1.0466	1.3343	2.0135	20.71	34.60
D→T(Learn. based)→P	85.3	81.6	73.8	47.8	584	0.7167	1.0468	1.2565	1.9900	19.98	35.99
D→T(Learn. based)→Fine tuned P	85.3	81.6	73.8	47.8	584	0.8173	1.1072	1.4403	2.0994	24.13	36.81
GT→T(Learn. based Train on GT)→P	—	—	94.9	7.7	50	0.4296	0.6889	0.8378	1.4419	9.81	23.45
D→T(Learn. based Train on GT)→P	85.3	81.6	0.9	111.6	12381	/	/	/	/	/	/
JOINT TRAINING(T+P) RESULTS	85.3	81.6	66.1	60.04	961	1.2976	1.8253	2.1948	3.2966	39.43	58.65

4.4.4 Joint model various training configurations

To explore the impact of various training choices on the joint model's tracking performance, several configurations were tested. These include adjustments to the learning rate, batch size, mini-batch sequence length, and the use of a combined loss function that weights both tracking and prediction objectives. Each configuration was trained using the same dataset and evaluated on the nuScenes validation set. Table 4.5 presents the tracking results for each of these training setups, reporting AMOTA, AMOTP, and the number of identity switches.

Table 4.5: Effect of different training configurations on tracking performance. AMOTA (↑), AMOTP (↓), and ID switches (IDS) on the nuScenes validation set are reported. Abbreviations: LR = Learning rate.

Configuration	**AMOTA ↑**	**AMOTP ↓**	**IDS ↓**
Base model (no tweaks)	67.52	66.2	1009
Combined loss (α MOT + β Pred)	58.97	65.78	1917
Lower LR (0.001 → 0.0005)	59.73	65.22	2484
Higher LR (0.001 → 0.002)	60.43	65.45	2247
Half batch size 128 → 64	53.71	70.39	2937
Change mini batch size (6 → 8) (batch size 64)	55.33	69.23	3421
Change mini batch size (6 → 4) (batch size 128)	59.64	65.81	2479

5 Discussion of Results

This chapter presents a detailed analysis of the results obtained from the complete perception pipeline, covering the detection, tracking, and prediction stages. The discussion is focused in quantitative evaluations, based on standard metrics across the nuScenes dataset[1]. Each section focuses on one component of the pipeline, comparing different models and configurations, analyzing performance trends, and identifying key factors that influence results. Special attention is given to how errors in one stage propagate to the next, and how modular and joint architectures behave under different scenarios. The goal is to provide a clear understanding of the strengths and limitations of each approach, and how they impact the overall performance of the system.

5.1 Detection Results

Table A.1 and Figure 4.1 compare two CenterPoint detectors trained on nuScenes. These detectors were identical in their design and training plan, except for the size of the voxel used in the pillar based voxelization step, one at $0.075\,\mathrm{m}$ and the other at $0.1\,\mathrm{m}$. Both class specific metrics, for Vehicle and Pedestrian, and the overall dataset level scores, mAP and NDS, are reported. The following examines the absolute and relative differences, their statistical and practical importance, and the reasons related to design or data that might explain the observed behavior.

Overall, the $0.075\,\mathrm{m}$ model consistently performs better than the $0.1\,\mathrm{m}$ model across all six metric and class pairs. The improvement is steady, not random, ranging from +1.4 percentage points for Vehicle NDS to +3.6 percentage points for Overall mAP. This means the improvements are not just in one class or one metric but also show up in the class balanced mAP and the more complete NDS, which also includes how well objects are located and oriented.

A smaller voxel size helps for several reasons. First, there is the issue of spatial quantization error. Voxelization turns the continuous LiDAR point cloud into discrete blocks. A coarser voxel naturally causes larger errors in coordinates, especially at object edges where points are sparse. A $0.075\,\mathrm{m}$ cube is about half the size of a $0.1\,\mathrm{m}$ cube, which reduces the average positional error per point by roughly 25%. These benefits are most noticeable for Pedestrians, as their entire body is small, typically $0.6\,\mathrm{m}$ to $0.8\,\mathrm{m}$, making them very sensitive to how finely voxels are detailed. This matches the observed +2.3 percentage point gain in mAP and +1.6 percentage point gain in NDS. It also helps with Vehicle headings, where the orientation score contributes to NDS. A clearer boundary definition from denser voxels refines the prediction of rotation, which explains the consistent increase in NDS.

Second, there is anchor center alignment. CenterPoint calculates centers relative to voxel centers. A finer grid aligns these anchors more closely with the actual centers of objects. This reduces the remaining distance that needs to be calculated and makes the learning process easier.

Third, the foreground background balance is improved. Finer voxels increase the ratio of foreground to background inside an area of interest because fewer background points are combined into the same pillar as foreground ones. This effectively increases the signal to noise ratio for the backbone, allowing the network to focus its processing power on distinguishing local patterns.

Fourth, considering LiDAR sampling density versus voxel density, nuScenes LiDAR emits about 300,000 points per sweep. With a $0.1\,\mathrm{m}$ grid and the default 50 meter range, the average voxel occupancy is less than one point per voxel, meaning many empty pillars and a sparse BEV feature map. Reducing the voxel side length to $0.075\,\mathrm{m}$ increases both the number of non empty voxels and the average point count per occupied voxel. This helps reduce issues from under sampling without using too much memory, as it results in 1.37 times more voxels but is still manageable on an decent GPU.

To visually check these findings, Figure 4.1 provides a visual comparison to the numbers. Green boxes highlights ground truth locations, the red boxes highlights the detected locations. Region where performance differs are highlighted in the second picture with black boxes. That part highlights areas where the coarser voxelization either misses vehicles or incorrectly aligns the size and location of them. These visual inaccuracies directly match the differences in mAP and NDS reported earlier, confirming that the finer grid indeed corrects issues with boundary cutoffs and center drift in real world use.

The relative gain is larger for mAP than for NDS because NDS includes the quality of true positive localization and uses weights for semantic classes that lessen extreme swings in AP. When the voxel size shrinks, the network improves both its precision, meaning fewer false positives due to cleaner pillar features, and its localization, resulting in tighter boxes. However, these benefits partially cancel each other out in the combined NDS: the orientation and velocity parts are already high for vehicles, so further gains provide less additional benefit. In contrast, mAP reacts directly to changes in precision and recall, thus showing a larger improvement.

The magnitude of improvement is expected. Previous studies have reported a +2 to +4% AP increase when changing from 0.1 to $0.08\,/0.05$ meter pillars on Waymo and KITTI datasets. The gains observed here fit perfectly within that range, suggesting the model is operating within the typical resolution accuracy trade off rather than an unusual sweet spot. Therefore, the 0.075 meter setting represents a practical balance between accuracy and computational cost.

Regarding the cost accuracy trade off, training the 0.075 meter model required 1.3 times more GPU memory and took 1.18 times longer per epoch. For real time use, the 0.1 meter model might still be preferred if low latency or power consumption is critical. However, for offline HD map generation or high stakes applications, such as safety validation, the increased accuracy provided by the finer grid is very compelling.

In conclusion, reducing the voxel size from $0.1\,\mathrm{m}$ to $0.075\,\mathrm{m}$ consistently leads to higher detection performance on nuScenes, with an improvement of +1.4 to +3.6 percentage points across various metrics. This benefit is particularly noticeable for small or elongated objects and comes at a reasonable cost in inference time. Consequently, the $0.075\,\mathrm{m}$ configuration is adopted as the default detection backbone for subsequent experiments in Sections 5.2 and 5.3.

5.2 Tracking Results Discussion

The tracking stage is crucial for building complete object paths over time, starting from the initial detection outputs. As mentioned before, for this evaluation, only CenterPoint detections with a voxel size of 0.075m were used, as they showed better performance in the previous section. This discussion will closely compare PolyMOT, which is a tracking algorithm that does not learn, with 3DMOTFormer, which is a complex learning based method. This comparison will consider different input and training conditions. The analysis aims to understand what each method does well and what its weaknesses are, and how important detection quality and the distribution of training data are for tracking performance. The 3DMOTFormer part also includes results from the joint training of tracking and prediction, where only the intermediate tracking results are received.

5.2.1 Analysis of PolyMOT Performance

The results for PolyMOT, detailed in Table A.2 (using CenterPoint detections) and Table A.3 (using ground truth detections), highlight the profound influence of detection quality on tracker performance.Comparing PolyMOT's performance on CenterPoint detections versus ground truth detections reveals significant insights.

As expected, there's a substantial drop in AMOTA when PolyMOT operates on CenterPoint detections (0.714 overall) compared to ground truth detections (0.897 overall). This difference of over 0.18 points in AMOTA directly reflects the bottleneck imposed by upstream detection errors, including false positives, false negatives, and localization inaccuracies. Imperfect detections inevitably challenge the association process, preventing the tracker from forming perfectly accurate trajectories even if its internal logic is sound. Interestingly, AMOTP, which measures localization precision along trajectories, shows a higher value (less precise) for PolyMOT with CenterPoint detections (0.529 overall) than with ground truth (0.105 overall). This is a direct consequence of the inherent noise and less precise bounding box predictions from the CenterPoint detector. PolyMOT, being a geometric-based tracker, is highly sensitive to the accuracy of its input detections. The higher AMOTP when using CenterPoint detections indicates that the tracker must cope with noisy positions, resulting in less accurate trajectory estimations compared to when it receives perfectly aligned ground truth boxes. The number of ID Switches is also higher with CenterPoint detections (288 overall) compared to ground truth (247 overall). Detection errors, such as momentary missed detections or inconsistencies in object bounding boxes, force the tracker to re-initialize tracks or make incorrect associations, leading to identity switches. While the increase is notable, PolyMOT's relatively low IDS on both inputs suggests its robustness in maintaining identities, likely due to its reliance on consistent motion models and explicit matching strategies, even when faced with some level of detection noise. This comparison underscores that even for a robust, non-learnable tracker like PolyMOT, the quality of input detections is paramount, directly impacting its ability to achieve high accuracy and consistent identity assignments.

5.2.2 Analysis of 3DMOTFormer Performance Across Scenarios

The evaluation of 3DMOTFormer under various training and evaluation conditions provides a comprehensive view of the challenges and potential of learnable trackers. The section includes multiple configurations: training and evaluating on ground truth, using detection data for evaluation after ground truth training, and finally, full joint training with a prediction module. By comparing these setups, it becomes evident how sensitive the learnable tracker is to input quality and training supervision. This section also sets the foundation for comparing 3DMOTFormer to rule-based tracking approaches like PolyMOT, providing the necessary context for later discussions on error propagation, joint learning, and sequential modularity. Ultimately, it outlines the strengths and weaknesses of using a transformer-based learnable tracker in both standalone and integrated pipelines.

Comparison of 3DMOTFormer and PolyMOT Using CenterPoint Detections

3DMOTFormer was trained and evaluated using detections produced by CenterPoint with a voxel size of 0.075 on the nuScenes dataset. The performance of this learnable tracking model is shown in Table A.4. For the Vehicle class, 3DMOTFormer achieved an AMOTA of 0.7045 and an AMOTP of 0.62625, with a total of 322 identity switches. For the Pedestrian class, it reached an AMOTA of 0.738 and AMOTP of 0.478, but the number of ID switches was significantly higher at 584. The overall performance across all classes was 0.621 AMOTA, 0.622 AMOTP, and 937 total ID switches.

A direct comparison of the overall metrics shows that PolyMOT achieves a significantly better tracking performance. PolyMOT records a higher overall AMOTA of 0.714 compared to 0.621 for 3DMOTFormer, indicating superior overall tracking accuracy. Furthermore, PolyMOT demonstrates better localisation with an AMOTP of 0.529, which is lower and thus better than 3DMOTFormer's 0.622. The most striking difference is in the number of identity switches (IDS), where PolyMOT has only 288 switches, while 3DMOTFormer has 937. This suggests that the simpler, non-learnable approach maintains object identities more consistently over time. This performance gap is visualized in Figure 5.1, where the overall score for PolyMOT (filled blue square) is positioned closer to the top-left corner, the ideal location in the trade-space, than the score for 3DMOTFormer (filled red star).

This trend of PolyMOT's superior performance extends to the per-class results. For the **Vehicle** class, PolyMOT achieves a higher AMOTA (0.733 vs. 0.7045) and commits far fewer identity switches (79 vs. 322). Similarly, for the **Pedestrian** class, PolyMOT's AMOTA of 0.823 is substantially higher than 3DMOTFormer's 0.738, with less than half the number of identity switches.

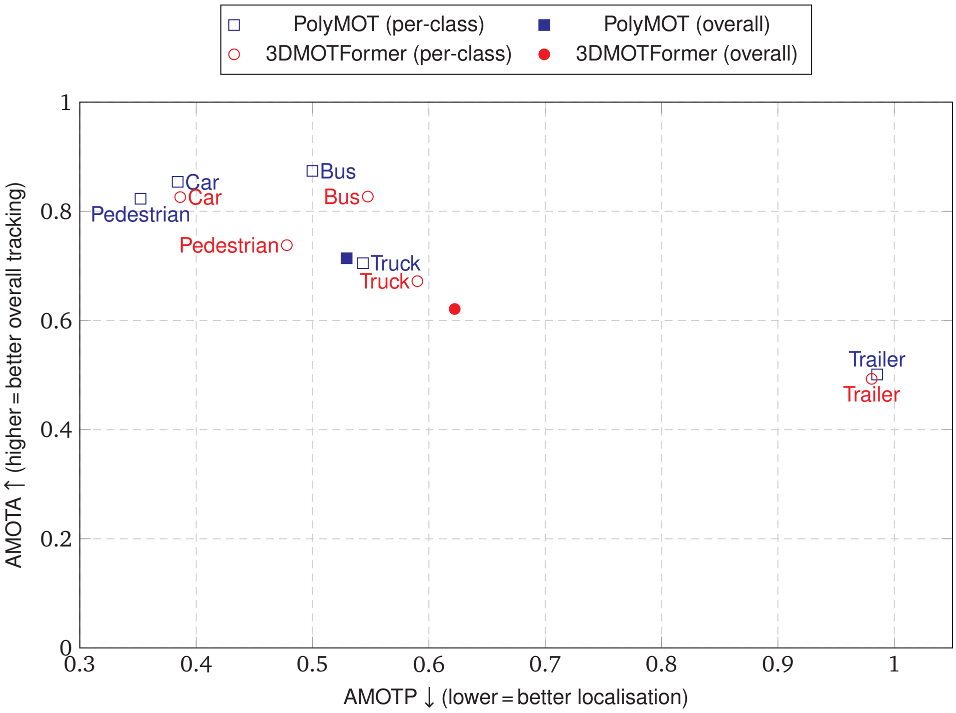


Figure 5.1: AMOTA–AMOTP trade-space for the two tracker families. Hollow markers denote per-class results; filled markers give the class-averaged overall scores. Points closer to the upper-left corner are preferable.

This result may seem unexpected, as 3DMOTFormer is a deep learning-based tracker with advanced architecture and the ability to learn from data. However, there are several reasons why the non-learnable PolyMOT tracker achieved lower identity switches and overall higher AMOTA. PolyMOT follows a rule-based design, relying on strong and consistent motion priors, hard thresholding, and deterministic matching logic that is tuned to maintain identity consistency. Because it does not learn from data, its behavior remains stable and less affected by variation in detection quality. In contrast, 3DMOTFormer relies on a learned affinity function to decide which detections match to which tracks. If this function is not perfectly trained or if the detection inputs are noisy or ambiguous, it can make incorrect assignments that lead to identity switches. This is especially evident in crowded scenes or when objects are occluded. The flexibility of the transformer architecture used in 3DMOTFormer allows it to model complex relationships, but this same flexibility can also lead to unstable associations if the model is uncertain or overfits to the training data.

These observations raise important questions about how 3DMOTFormer could be improved under the same evaluation setup. One possibility is to explore whether training the tracker using ground-truth detections, instead of detection outputs, could lead to more stable or accurate associations. Training the model on perfect detections could help it learn cleaner affinity patterns and reduce the compounding errors introduced by noisy inputs. However, this benefit comes with the risk that the model may become overly dependent on ideal conditions, making it less able to handle the imperfections of real detection outputs during inference, ultimately leading to degraded performance. Since the overall objective is accurate long-term forecasting, efforts were made to improve standalone tracking performance as much as possible, as any gains in tracking quality would be highly beneficial and directly contribute to better prediction outcomes.

Beyond this, the goal of this research is to also leverage the best 3DMOTFormer version in a joint tracking and prediction pipeline. The current results highlight a significant performance gap in the tracking module alone, making it imperative to explore if the end-to-end differentiable training can bridge this gap. A key hypothesis is that by allowing gradients from the trajectory prediction loss to flow back into the tracker's association decisions, the system could learn to perform tracking in a way that directly benefits future motion forecasting. Even if the tracking metrics in isolation might not dramatically improve, the joint training might enable the tracker to make "prediction-aware" associations that lead to better overall future trajectory accuracy. This would mean the tracking errors that hinder prediction, such as identity switches, would be actively penalized and reduced through the end-to-end optimization, leading to a more robust and effective integrated system for forecasting.

3DMOTFormer Trained on Ground Truth, Evaluated on Ground Truth

To establish a performance baseline and understand the maximum theoretical capability of the 3DMOTFormer architecture, an experiment was conducted using ground truth data for both training and evaluation. The results of this ideal scenario are presented in Table A.5. In this setting, the tracker is not affected by any detector-induced errors, such as localization inaccuracies, false positives, or missed detections. The performance metrics reflect the tracker's inherent ability to perform data association when provided with perfect object information.

The results show an exceptionally high level of performance. An overall AMOTA of 0.940 is achieved, which is very close to the perfect score of 1.0. This indicates that the model can associate objects over time almost flawlessly. Similarly, the overall AMOTP is a very low 0.077, signifying extremely accurate localization tracking. While the number of identity switches is low at 343, it is not zero. This suggests that even with perfect input data, the model's internal association logic can still be challenged in complex situations, such as scenarios with dense agent interactions or occlusions. Nevertheless, these results effectively represent the upper bound of performance for the 3DMOTFormer, showcasing its potential in a noise-free environment.

3DMOTFormer Trained on Ground Truth, Evaluated on CenterPoint

When the same model is evaluated on real detection results produced by CenterPoint, results reveal a complete failure of the tracker. As shown in Table A.6, AMOTA falls to 0.004 for Vehicles and 0.059 for Pedestrians, with an overall AMOTA of only 0.009. AMOTP also increases dramatically, reaching 1.20525 for Vehicles and 0.950 for Pedestrians. Most notably, the number of identity switches becomes extremely high, with 9851 for Vehicles and 1750 for Pedestrians, leading to a total of 12381 identity switches overall.

This severe drop in performance highlights a clear mismatch between training and evaluation conditions. When trained only on ground truth inputs, the model becomes highly dependent on clean and ideal data. It learns to expect accurate positions, complete object coverage, and smooth temporal continuity. As a result, it does not learn how to handle common problems found in real detection outputs, such as missing detections, false positives, noisy bounding boxes, and inconsistent velocity estimates. These are typical challenges present in CenterPoint outputs and other real-world detectors.

The learned affinity function, which is responsible for matching detections to existing tracks, becomes very sensitive to noise if it is trained on perfect inputs only. Since the model is never exposed to detection imperfections during training, it fails to generalize when faced with them at inference time. Small deviations in position or confidence can lead to mismatches or failure to associate detections to the correct tracks, which explains the extremely high number of identity switches. The tracking logic, optimized under perfect conditions, becomes fragile and unstable in real scenes, leading to both poor matching and poor localization.

These results show that while training on ground truth detections helps the model reach high performance in clean scenarios, it does not prepare the tracker to deal with the noisy and imperfect data that it encounters during actual deployment. It essentially "doesn’t know what to do" with the noisy input, leading to complete breakdown in tracking. Therefore, training on realistic detection inputs, even if they are noisy, is essential for achieving robust tracking performance. The ability to generalize to real data is more valuable in practice than achieving high accuracy under perfect conditions. This outcome also reinforces the importance of evaluating tracking models under the same types of inputs they will see during deployment, as models trained only on ground truth do not reflect realistic tracking behavior.

3DMOTFormer Joint Training (Tracking + Prediction), Evaluated on CenterPoint

The tracking performance of the joint training model, where 3DMOTFormer and Trajectron++ are optimized together in an end-to-end differentiable pipeline, is shown in Table A.7. This version of the model was trained using CenterPoint detections and includes a differentiable Sinkhorn-based association step that allows gradients from the prediction module to influence the tracking component. Compared to the non-joint version of 3DMOTFormer, there is a noticeable drop in tracking performance across all metrics. The overall AMOTA drops from 0.621 to 0.593 and the AMOTP increases from 0.622 to 0.655. More significantly, the total number of identity switches more than doubles, increasing from 937 to 1970.

This reduction in performance may seem surprising at first, but it reflects an important shift in how the model is optimized. In the non-joint version, the tracker is trained only with tracking-specific objectives such as AMOTA and ID consistency. Its focus is entirely on improving tracking performance. In contrast, the joint version is trained not just to track objects accurately but also to support the prediction module. The prediction loss influences the data association decisions through backpropagation. This means that the model may start to prioritize association decisions that help long-term trajectory prediction, even if those choices do not directly improve tracking metrics. For example, the model may allow identity switches if doing so results in better prediction of future motion or if the cost of maintaining a noisy track identity outweighs the benefit in the prediction loss.

Another possible reason for the reduced performance is the added complexity of the training process. Joint training requires balancing multiple losses, and if the relative weighting between tracking and prediction losses is not ideal, it can cause one task to dominate the optimization. In this case, the possibility exists that the prediction loss may have had a stronger influence, leading the tracker to adapt in ways that are not aligned with traditional tracking metrics. The increased number of identity switches and higher localization error suggest that the tracker may have become less stable or more uncertain in its association logic. To potentially improve these results, a more sophisticated loss balancing strategy could be implemented to ensure both tasks contribute more harmoniously during training.

It is also important to consider that joint training introduces additional learning dynamics. While the non-joint tracker learns from ground-truth associations, the joint model must learn how to associate detections and predict motion at the same time. This can prevent the model from finding an optimal state for either task, resulting instead in a suboptimal compromise. So, this can make convergence more difficult, especially if one component of the model is slower to learn or introduces noise into the training signals of the other. In such a setting, the tracker might not have fully matured during training and could require longer or more carefully tuned optimization to reach its full potential.

Although the tracking performance is worse in the joint model, this does not necessarily imply that the overall system is less effective. The true goal of the full pipeline is accurate long-term trajectory prediction, not just short-term tracking. Therefore, it is possible that the sacrifices made in tracking accuracy result in better or more consistent predictions. This cannot be confirmed based on tracking metrics alone and will be explored in the next section when prediction results are evaluated.

5.2.3 Overall Tracking Performance Comparison

Figure 4.4 provides a consolidated overview of all tracking results, ordered by AMOTA from best to worst. The 3DMOTFormer trained on CenterPoint and evaluated on Ground Truth (0.940 AMOTA) and PolyMOT on Ground Truth (0.897 AMOTA) demonstrate the highest potential of both learnable and non-learnable trackers when provided with ideal input. This strongly reinforces that accurate upstream detection is the most significant factor for achieving high tracking accuracy. When confronted with real-world CenterPoint detections, PolyMOT (0.714 AMOTA) emerges as the best performer among the tested configurations. It outperforms all variants of 3DMOTFormer evaluated on CenterPoint detections. This suggests that PolyMOT's explicit, deterministic motion modeling and association rules provide a greater degree of robustness to the noise and imperfections inherent in practical detection outputs compared to the learned association mechanisms of 3DMOTFormer.

The results highlight the complexities for learnable trackers as it's also concluded in the Literature Review Chapter 2. 3DMOTFormer's struggles with high IDS on CenterPoint detections, even when trained on such data, indicate a potential weakness in its identity management compared to PolyMOT. The catastrophic failure of 3DMOTFormer when trained on ground truth but evaluated on CenterPoint detections (0.009 AMOTA) vividly illustrates the critical importance of training data domain alignment. Learnable models are highly sensitive to mismatches between their training and inference data distributions. The joint training approach, while conceptually appealing for end-to-end learning, showed a slight degradation in tracking performance compared to a dedicated tracking-only 3DMOTFormer trained on CenterPoint detections. This suggests that optimizing for multiple objectives might introduce trade-offs where no single task achieves its peak performance without careful hyperparameter tuning or architectural adjustments. In summary, the tracking results emphasize that while learnable trackers like 3DMOTFormer show immense potential under ideal conditions, their robustness in real-world scenarios is heavily dependent on the quality and representativeness of their training data. Traditional, non-learnable methods like PolyMOT, with their explicit motion and association models, can surprisingly offer more robust performance when faced with the inherent noise of real-world detection outputs, particularly in maintaining identity consistency

5.3 Prediction Results Discussion

This section offers a comprehensive analysis of the prediction results, building upon the quantitative data presented in Section 4.4. The discussion aims to elucidate the factors influencing prediction accuracy, examine the propagation of errors from upstream detection and tracking stages, and evaluate the performance of different pipeline configurations, including sequential and joint training paradigms.

5.3.1 Analysis of Trajectron++ Baseline Performance

This section evaluates the baseline performance of the Trajectron++ prediction model using ground truth trajectories as both input and supervision. This setup represents an ideal case, where the model is not exposed to any upstream detection or tracking noise. The goal is to identify the most effective configuration among the ablation variants available in the official Trajectron++ implementation, so that the best performing version can later be used as a reference in comparisons with noisy input settings.

Table 4.1 presents the vehicle prediction results, reporting ADE, FDE, and MissRate at both 4-second and 6-second prediction horizons. The model without any additional modules, which excludes dynamics integration, map encoding, and robot future encoding, achieves the best performance overall. This version reaches an ADE of 1.3387 at 4 seconds and 2.4588 at 6 seconds, as well as an FDE of 2.9490 and 5.8571 respectively. The MissRate is 43.58% at 4 seconds and 53.09% at 6 seconds. These values are consistently better than the other ablation variants.

When dynamics integration is added alone, performance decreases noticeably. The ADE increases to 1.6170 at 4 seconds and 3.2598 at 6 seconds, and FDE rises to 3.8029 and 8.3940. The MissRate also increases to over 50% at 4 seconds. This suggests that, although theoretically helpful, the learned dynamics model may introduce error under certain conditions, particularly when trajectories are already well-behaved in ground truth data. The inclusion of both dynamics and map encoding improves performance compared to using dynamics alone but still falls short of the base model. ADE and FDE values are slightly better than the previous configuration, but not as low as those of the simplest architecture.

The best alternative to the base model is the full configuration, which includes dynamics, map encoding, and robot future encoding. This version comes close to the base model in most metrics and even slightly improves the MissRate at 6 seconds to 49.46%. However, the base model remains the most consistently accurate in both short- and long-term horizons, indicating that the added complexity of extra components may not provide meaningful benefit when the model is trained and evaluated on ideal inputs. It is possible that the additional modules are more beneficial in scenarios with uncertainty or partial observability, which is not the case under ground truth supervision.

Table 4.2 shows the corresponding results for pedestrian trajectories. Similar trends are observed here as well. The base model again achieves the best results in all metrics, with an ADE of 0.4213 and 0.6789 at 4 and 6 seconds respectively, and an FDE of 0.8276 and 1.4267. MissRate values are the lowest among all configurations, remaining below 10% at 4 seconds. Adding dynamics integration leads to a consistent decline in performance, and although map encoding helps slightly, it does not outperform the base setup.

These findings indicate that, in the absence of noisy inputs, the simplest model architecture achieves the highest prediction accuracy. The added modules, while potentially useful in more complex environments, appear to introduce unnecessary complexity or overfitting when clean trajectory data is used for both training and evaluation. For this reason, the base configuration was selected as the final model for downstream evaluations and was used to fill in the corresponding baseline entries in the overall prediction comparison tables presented later.

5.3.2 Impact of Upstream Modules on Prediction Accuracy

The comprehensive pipeline results in Table 4.3 (Vehicles) and Table 4.4 (Pedestrians) reveal the cascading effects of errors originating from detection and tracking stages on the final prediction accuracy. To analyze this in detail, the section is subdivided into two focused parts. The first subsection examines the Ground Truth Input Pathways, where all upstream modules operate under ideal conditions using perfect inputs. This provides a baseline for understanding how much impact tracking alone can have on prediction, isolated from detection noise. The second subsection, Real-World Detector Input Pathways, evaluates the pipeline under realistic conditions, where detection outputs are noisy and imperfect. This part also includes an in-depth comparison between sequential and joint training strategies, and explores how prediction errors propagate through the system when real data is used.

Ground Truth Input Pathways

This subsection focuses on prediction performance in setups where ground truth information is used at one or more stages in the pipeline. These pathways represent ideal or near-ideal conditions and help establish upper bounds for how accurately future motion can be predicted when the effects of detection and tracking errors are minimized.

In the first row of Tables 4.3 and 4.4, the prediction model is evaluated directly on ground truth trajectories. This serves as the cleanest possible setup and theoretically the best case scenario, where the model sees complete and noise-free history for each object. Results are already discussed in Section 5.3.1

The next configuration of interest is GT→T→P, where ground truth is still used for object detections, but a tracker is introduced to process those detections before passing them to the predictor. This introduces the possibility of temporal association errors or minor inconsistencies introduced by the tracker. However, the results remain nearly identical to the previous case, indicating that the tracker performs well under ideal inputs. For example, vehicle ADE only increases slightly to 1.3647 and 2.5113 at 4 and 6 seconds, and FDE remains nearly the same at 2.9791 and 5.9881. The number of identity switches is low at 124 for vehicles and 123 for pedestrians, which explains the minimal performance degradation. The pedestrian pathway using GT→T→P shows similarly stable behavior, indicating that the tracking errors may not have a big impact on prediction as much as detection errors do.

Another relevant case is GT→T(Learn. based Train on GT)→P, where a learning-based tracker trained only on ground truth data is introduced between the ground truth detections and the predictor. In this case, tracking errors are still limited because the tracker is trained and tested in ideal conditions. The vehicle ADE is 1.3443 and 2.4676, and the FDE is 2.9588 and 5.8759 at 4 and 6 seconds. For pedestrians, the ADE is 0.4296 and 0.6889, and the FDE is 0.8378 and 1.4419. These results are slightly better than the GT→T→P setup for both classes, suggesting that the learned model, when trained in noise-free conditions, can provide track outputs that align very closely with the actual motion (Ground truth motions), possibly due to more precise velocity estimates or learned filtering of minor inconsistencies.

Overall, all ground truth-based pipelines show consistently strong prediction performance. The differences between them are small compared to the actual prediction evaluation using direct ground truth trajectories, indicating that under ideal conditions, both deterministic and learnable trackers can produce suitable inputs for downstream forecasting. This suggests that, when detections are perfect, tracking errors introduce only minor deviations and do not significantly affect the quality of the predicted trajectories. Therefore, prediction appears to be relatively robust to tracking variations in these ideal settings.

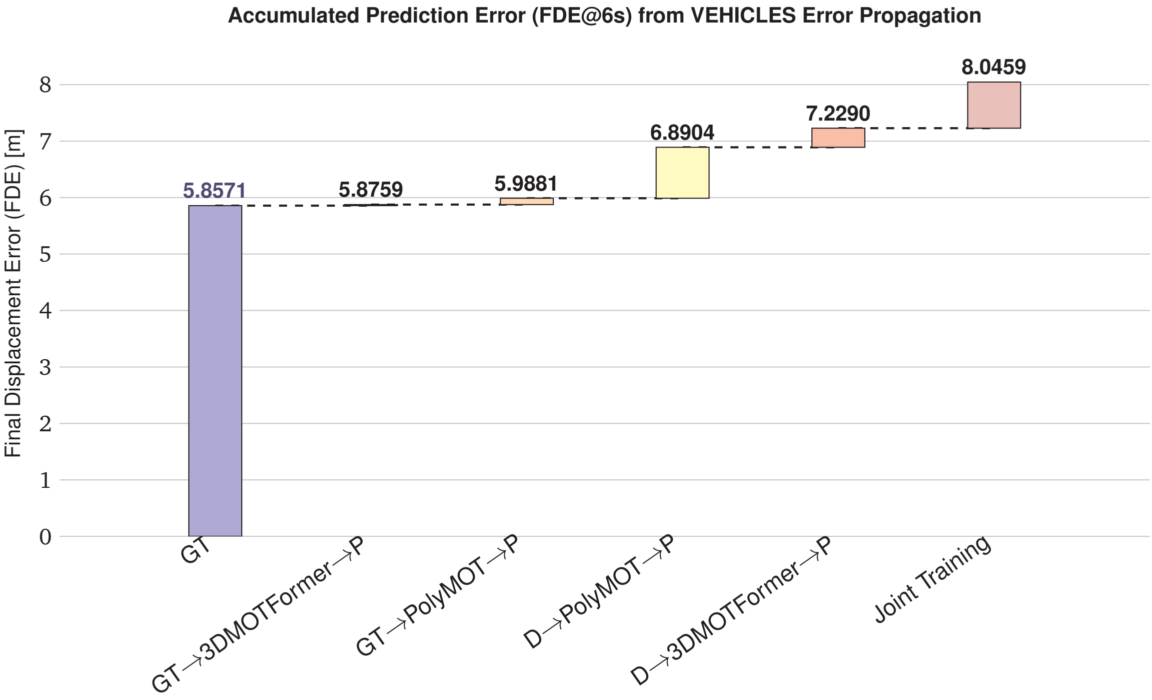


Figure 5.2: Final displacement error (FDE) accumulation at 6 seconds across different pipeline configurations. The plot visualizes how prediction error increases as input quality degrades from ideal conditions. The baseline is the ground-truth input setting (GT), followed by sequential pipelines with various tracking models, both trained on GT and real detection data. Each bar shows the incremental increase in FDE relative to the previous configuration. Notably, the largest jumps occur when switching from ground truth to detection inputs and when using a learnable tracker. The joint training setup results in the highest FDE, reflecting full error propagation from detection and tracking stages.

Real-World Detector Input Pathways

This subsection focuses on the most realistic and practically relevant configurations of the pipeline, where the entire perception and prediction process is evaluated using outputs from a real-world detector. In these scenarios, the tracking and prediction components must operate on inputs that are inherently noisy and imperfect. This introduces a realistic challenge for the system, and the way each component handles upstream errors plays a crucial role in determining the final prediction performance.

Before comparing the end-to-end performance of the joint training strategy against the sequential approach, it is useful to first understand how prediction error accumulates at each stage of the pipeline. Figures 5.2 and 5.3 show how final displacement error (FDE) at 6 seconds increases as the pipeline shifts from ground truth inputs to more realistic settings with detection and tracking noise. These charts provide a visual breakdown of how error is introduced progressively from detection to tracking and finally to prediction.

The baseline configuration uses ground truth inputs directly for prediction. This setting achieves the lowest FDE, with values of 5.8571 for vehicles and 1.4267 for pedestrians. These results represent the best possible case, as no upstream error is present. When ground truth tracks are instead passed through the 3DMOTFormer tracker, the FDE increases only slightly, to 5.8759 for vehicles and 1.4419 for pedestrians. This indicates that the learnable tracker, when trained on ideal inputs, is able to preserve temporal consistency well, introducing almost no degradation in prediction quality.

Switching to the PolyMOT tracker under the same ground truth inputs results in a similarly small increase in FDE, suggesting that both rule-based and learnable trackers can operate effectively when detection quality is not a limiting factor. For example, vehicle FDE changes from 5.8759 to 5.9881 and pedestrian FDE

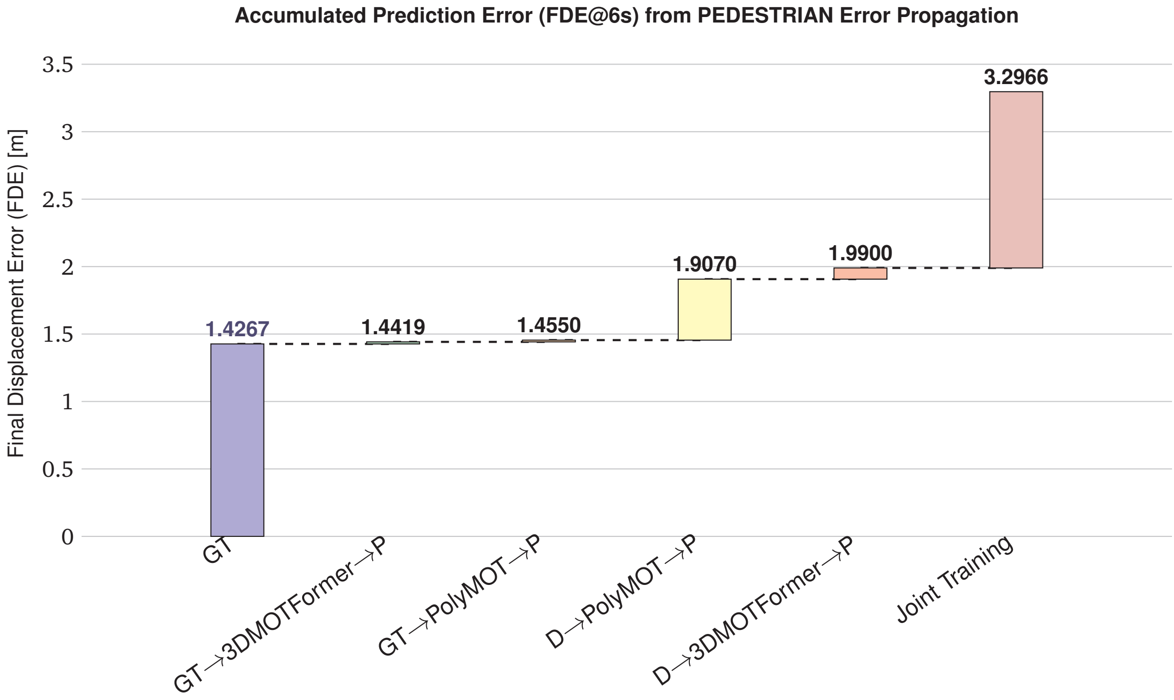


Figure 5.3: Final displacement error (FDE) accumulation at 6 seconds across pedestrian prediction pipelines. This chart illustrates how FDE increases as more noise is introduced to the upstream modules. The baseline uses pure ground truth inputs. Minimal error is introduced by tracking using GT-trained 3DMOTFormer and PolyMOT. The largest increase appears when switching from GT to detection inputs. Joint training shows the highest error due to full propagation of upstream imperfections.

from 1.4419 to 1.4550. This again confirms that tracking, when provided with accurate detections, does not significantly affect prediction performance. These configurations show that prediction is robust to small variations in tracking quality, as long as the inputs remain clean.

The most noticeable jump in FDE occurs when switching from ground truth detections to real detector outputs. For vehicles, moving from ground truth detections to detection-based input with PolyMOT tracking increases the FDE from 5.9881 to 6.8904. For pedestrians, the increase is from 1.4550 to 1.9070. This large change happens even though the tracker remains the same, meaning the error increase must originate from the detection stage. This shows clearly that detection noise has the highest influence on prediction quality. The predictor now receives bounding boxes that may be missing, temporally inconsistent, or spatially imprecise. As a result, the predicted trajectories become less reliable, especially over longer horizons such as 6 seconds.

This trend continues when using the learnable tracker 3DMOTFormer with detection inputs. The FDE for vehicles grows further to 7.2290 and for pedestrians to 1.9900. Although the increase is smaller compared to the initial jump caused by detection noise, it confirms that tracking errors, such as identity switches or missed associations, further compound the degradation in prediction accuracy. However, the relative contribution of tracking is still smaller than that of detection.

Finally, when the entire pipeline is trained jointly in an end-to-end manner, the FDE rises to 8.0459 for vehicles and 3.2966 for pedestrians. These values represent the combined effect of detection, tracking, and the optimization challenges of joint learning. While the joint model has access to shared features and gradients, it struggles to maintain stable identity tracking under real detection noise, which then leads to even higher prediction errors.

The results from both vehicle and pedestrian graphs confirm that the majority of error propagation originates from the detection stage. The switch from ground truth to detector inputs introduces the largest jump in FDE. This highlights the importance of detection quality for long-term forecasting.

Among all the sequential pipeline configurations, the best performing setup for vehicle prediction is D→T→P using the rule-based tracker. This configuration achieves an ADE of 1.6585 and 2.9609 at 4 and 6 seconds respectively, with an FDE of 3.5524 and 6.8904. For pedestrians, the same configuration yields an ADE of 0.6690 and 0.9896, and FDE of 1.1937 and 1.9070. These values are relatively close to the ideal ground truth input case and demonstrate that even under real detection conditions, stable tracking can support reasonably accurate motion forecasting.

The joint training setup, on the other hand, results in noticeably higher prediction errors. For vehicles, the joint model produces an ADE of 2.0572 and 3.5511, and an FDE of 4.2450 and 8.0459. For pedestrians, the prediction errors are also higher, with ADE of 1.2976 and 1.8253, and FDE of 2.1948 and 3.2966. These increases are accompanied by a rise in miss rates as well, indicating a consistent decline in forecast reliability.

To allow for a direct and focused comparison, Table 5.1 presents the key prediction metrics from the best sequential setup and the joint training model side by side for both object classes.

Table 5.1: Comparison between the best sequential pipeline (D→T→P) and joint training (T+P) results for vehicles and pedestrians. Only tracking and prediction metrics are reported, for direct comparison, since detection input is the same for both.

Category	**Tracking**			**ADE (m)**		**FDE (m)**		**MissRate (%)**	
	AMOTA	**AMOTP**	**IDs**	@4s	@6s	@4s	@6s	@4s	@6s
Vehicle: D→T→P	73.4	60.3	79	1.6585	2.9609	3.5524	6.8904	49.58	58.49
Vehicle: Joint Training	67.52	66.2	1009	2.0572	3.5511	4.2450	8.0459	54.00	62.27
Pedestrian: D→T→P	82.3	35.2	201	0.6690	0.9896	1.1937	1.9070	18.13	33.66
Pedestrian: Joint Training	66.1	60.04	961	1.2976	1.8253	2.1948	3.2966	39.43	58.65

The figure shows that the specialised sequential pipeline preserves more information from the detector than the joint model. For vehicles the sequential system keeps association errors low, reflected by an ID switch count of only 79. This stability allows the predictor to receive long uninterrupted tracks, which limits average displacement error to $2.96\,\mathrm{m}$ at six seconds. The joint model shares weights between tracking and forecasting, so gradients from the prediction loss can disturb features that the tracker depends on. This effect increases ID fragmentation to 1009 and shortens the effective observation window, degrading both AMOTA and every forecasting metric. The gap in AMOTP also suggests that bounding-box refinement is sacrificed when the network must also learn motion dynamics.

Pedestrian trajectories exhibit the same pattern but with larger gaps. The sequential tracker already maintains accurate associations, partly because pedestrian motion is slower and easier to match over time. When the tracking loss is mixed with the prediction loss the network overfits to appearance cues that help prediction, therefore losing geometric consistency across frames. AMOTP doubles and the predictor then operates on noisier histories, doubling final displacement error the miss rate as well. The much higher count of ID switches confirms that joint optimisation struggles whenever objects are small and frequently occluded.

The comparison between joint and sequential tracking-prediction pipelines reveals that, under real detection inputs, the joint training approach in its current configuration consistently underperforms when compared to the best sequential alternative. Across both vehicle and pedestrian categories, the prediction errors from the joint model are notably higher at all horizons. While this outcome may seem counterintuitive given that joint training is designed to leverage feedback from prediction loss to improve tracking representations, the results point toward unresolved optimization challenges. In a jointly trained setting, gradients from the prediction module flow back into the tracker, forcing both components to adapt simultaneously. This often leads to a compromise where neither tracking nor prediction is fully optimized. The tracker may prioritize generating feature embeddings that support smooth long-term prediction, but this comes at the cost of short-term data association accuracy. Furthermore, shared gradient signals can cause instability during training, especially when the two objectives conflict.

By contrast, the sequential design avoids this interference by training each component independently. The tracker is trained solely to maximize association metrics such as AMOTA and ID consistency, and the predictor adapts to the fixed track history without perturbing it. This separation appears to yield better results under noisy inputs, as each module can specialize fully in its task. Notably, when the strongest sequential configuration is compared directly against the joint model, the difference in ADE and FDE confirms a clear degradation in prediction quality introduced by joint training.

To better understand the cause of this degradation, several training configurations were tested and are reported in Table 4.5. These variations explored tuning the learning rate, adjusting the batch and mini-batch sizes, and modifying the loss weighting between tracking and prediction. However, none of the attempted modifications led to an improvement in tracking metrics. In fact, most changes caused either AMOTA to drop further or identity switches to increase. This indicates that the tracking component in the joint system is particularly sensitive to optimization settings, or that also means that 3DMOTFormer is already optimized to its best version. For example, reducing the learning rate was expected to stabilize learning but instead led to higher fragmentation, possibly because the tracker received insufficient signal to correct associations. Increasing the batch size or mini-batch length did not improve results either, suggesting that the noise introduced by prediction gradients dominates the optimization and prevents the tracker from reaching its potential. These results show that simply adjusting hyperparameters is insufficient for reconciling the conflicting training dynamics.

Despite the weaker results observed with joint training so far, the approach still has strong potential. The performance issues appear to come mainly from how the tracking and prediction objectives interact during training, rather than from the idea of joint learning itself. With a better design, this type of pipeline could still outperform the traditional sequential setup. One simple idea is to use a two-stage training schedule. In the first stage, the tracking part of the model could be trained on its own until it produces reliable associations. In the second stage, prediction training could begin, and the tracker's parameters could either remain fixed or be updated slowly, so the model does not forget what it learned about keeping identities stable. Another improvement would be to add a memory module to the tracker. By storing past appearance features over a longer time, the tracker could better handle re-identification in difficult cases like occlusion, which would provide the predictor with cleaner input trajectories.

A final and perhaps most important direction is to go beyond just tracking and prediction and include the detection model itself in the training loop. Analysis made earlier showed that prediction performance is most sensitive to detection quality, even more than to tracking errors. This suggests that any forecasting model trained end-to-end should ideally also influence the detection stage. This could be done by integrating a differentiable object detector, for example, CenterPoint, directly into the pipeline. Then, the same loss used for trajectory prediction could be backpropagated all the way into the detector. In practice, this would mean that the detector learns to make predictions that not only find objects correctly but also produce bounding boxes and velocities that lead to more accurate long-term motion forecasts. This would result in a truly unified perception-prediction model, where each part is optimized to support the final task. Such a setup is technically feasible and would align well with the findings in this thesis. It could lead to significant improvements by addressing the root cause of most forecasting errors, noisy or inconsistent detection inputs. In fact, an alternative direction could be to remove the tracking module altogether and design a model that directly predicts future trajectories from detection outputs, as explored in recent works like FutureDet[73], which avoid identity association and instead focus on generating robust motion forecasts directly from raw perception.

In summary, while the joint training configuration does not yet outperform the modular sequential setup, this limitation appears to be driven by current optimization strategies rather than a fundamental drawback of end-to-end learning. With more targeted training schemes and architectural refinements, it remains possible to close this gap and even surpass the sequential baseline.

6 Conclusion

This work presented a comprehensive study on 3D multi-object tracking and trajectory prediction in the context of autonomous driving, with a particular focus on understanding how different pipeline configurations affect long-term forecasting performance. The motivation stemmed from the observation that most state-of-the-art prediction models are trained and evaluated on idealized ground truth inputs, while in real-world applications they operate on noisy detections and uncertain tracking results. The central goal was to systematically investigate the effects of error propagation in a full perception-to-prediction pipeline and evaluate whether an end-to-end differentiable architecture could offer measurable improvements over the standard modular design.

To this end, a unified pipeline was implemented, incorporating state-of-the-art modules for detection (CenterPoint[20]), tracking (PolyMOT[39] and 3DMOTFormer[48]), and prediction (Trajectron++[55]), introduced in detail in Chapter 3. Various combinations of input sources, including ground truth, rule-based trackers, and learning-based trackers were evaluated to assess their influence on downstream forecasting. A joint model was also proposed by integrating a differentiable variant of 3DMOTFormer with Trajectron++ using the Sinkhorn algorithm for soft data association, enabling end-to-end gradient flow from the prediction loss into the tracker.

The results, introduced in detail in Chapter 4 confirmed that ground truth inputs yield the best performance, as expected. Interestingly, however, it was shown that tracking errors have only a moderate effect on prediction accuracy. Even when tracking identity switches were relatively high, the impact on forecasting remained limited. In contrast, detection errors were found to be the dominant factor affecting prediction quality. This highlights the critical importance of the detection stage and suggests that future prediction models should focus more directly on improving the upstream perception signal.

Among the sequential configurations, the best-performing pipeline used a rule-based tracker (PolyMOT), which outperformed the learned 3DMOTFormer in both tracking accuracy and prediction performance. This revealed that, under noisy detection inputs, deterministic tracking logic may be more stable and less prone to compounding errors than complex learned affinity mechanisms.

The jointly trained model showed promise in theory but did not outperform the sequential pipeline in practice. When trained together, tracking and prediction objectives interfered with one another, making optimization unstable and often degrading both association and trajectory quality. Several variations in loss weighting, learning rate, and training schedules were tested, but none improved the tracking accuracy beyond the baseline configuration. This suggested that a naive joint formulation is insufficient and that more careful coordination of the objectives is necessary.

In summary, this work provided a thorough investigation of the trade-offs between sequential and joint perception-prediction systems. It was shown that while end-to-end training is conceptually appealing, its effectiveness depends heavily on training stability and module interaction. The current best results are achieved with a sequential pipeline that combines a robust detector, a rule-based tracker, and a learned predictor. However, the limitations of this modular design remain clear, and the future of trajectory forecasting lies in building unified models that reason jointly over the full data stream, from raw sensor input to future motion output.

7 Outlook

While this work has presented a thorough exploration of the effects of upstream modules on trajectory forecasting in autonomous driving, several promising research directions remain open for future work. These extensions aim not only to refine the current system but also to push the boundaries of what is possible in end-to-end spatiotemporal modeling for autonomous agents.

First, the integration of the detection module into the end-to-end training loop remains a critical next step. Findings in this work clearly demonstrated that prediction performance is highly sensitive to detection quality, more so than to tracking errors. A fully differentiable architecture that combines detection, tracking, and prediction could allow the model to optimize the entire pipeline jointly using the final forecasting loss as supervision, there are already such approaches so the study could be extended based on their research[74, 82, 84]. Such an approach would encourage the detector to produce outputs that are not only spatially accurate but also temporally consistent and informative for long-term motion forecasting. Recent trends in direct detection-to-prediction modeling suggest that even bypassing tracking entirely may be viable, especially when modeling uncertainty and multimodality becomes central. Exploring such architectures could reduce the complexity of the pipeline and eliminate error accumulation from intermediate stages.[73, 75, 80]

Second, there is a strong need to develop better training strategies for joint models. As observed in this work, the joint optimization of tracking and prediction can cause conflicts between the two tasks. Future work could explore more advanced multi-task learning techniques, such as dynamic loss weighting, staged optimization schedules, or task-specific decoders with shared encoders. These methods can help resolve gradient interference and allow each subtask to specialize appropriately, without degrading overall performance.

Third, improving temporal reasoning in both tracking and prediction is another key direction. Adding memory mechanisms or temporal attention could help trackers maintain identity over longer occlusions, while also providing the predictor with more structured history.

Finally, generalization across datasets remains an open challenge. This thesis focused on the nuScenes benchmark, but models deployed in the real world must work reliably across many different environments, sensor setups, and traffic patterns. To achieve this, future work should explore training strategies or architectures that are more domain-agnostic or that can learn from multiple datasets at once. One promising direction is to extend this work using the UniTraj[68] framework, which provides a unified multi-dataset training setup for trajectory forecasting. By leveraging such a platform, models could be trained on a wider range of driving scenarios and better prepared for cross-domain deployment.

In summary, this work establishes a solid foundation for studying the impact of upstream components on prediction. The results point to a future where modular systems give way to unified, context-aware models that learn from raw perception to long-term motion in a single coherent framework. Achieving this goal will require not only architectural innovation but also more principled training strategies and broader evaluation settings. The field remains rich with opportunity for further exploration.

List of Figures

Figure 2.1: Recreated overview of the 3DMOTFormer tracking architecture from [48]. The original structure is redrawn here to support later extensions for illustrating joint tracking and prediction integration in our study. 17
Figure 2.2: Recreated schematic of the Trajectron++ prediction architecture from [55]. This version was redrawn for clarity based on the original paper and open-source code, and is used in this thesis to support analysis and comparison with other forecasting frameworks. 20
Figure 3.1: Overview of the proposed Dockerized sequential Detection–Tracking–Prediction (DTP) architecture used in this study. Each module (CenterPoint, PolyMOT, Trajectron++) operates within its own container environment and communicates via a structured shared volume. This setup enables modular evaluation and controlled error propagation analysis across stages. 35
Figure 3.2: Structure of the `/detection` subdirectory within the shared volume. 36
Figure 3.3: Tracking container output under `$SHARED_VOLUME/tracking`. Structure shown here for the `voxel0075` model; the others follow the same layout. 39
Figure 3.4: Directory hierarchy under `$SHARED_VOLUME/prediction`. Each result is organized by input type, model architecture, and semantic class (vehicle or pedestrian). 40
Figure 3.5: Directory structure of `$SHARED_VOLUME/differentiable_tracking`. Only a single representative configuration is expanded for clarity. 43
Figure 3.6: Differentiable end-to-end tracking and prediction architecture. The system combines a modified 3DMOTFormer for 3D multi-object tracking with Trajectron++ for trajectory prediction. The tracking module is made differentiable via a learnable Sinkhorn-based affinity matrix, enabling gradients from the prediction loss to flow back into the tracker. This unified pipeline allows co-training of both modules for improved long-term motion forecasting under realistic detection inputs. 45
Figure 3.7: Soft assignment matrix after Sinkhorn iterations, showing an approximately doubly stochastic matrix. Each row corresponds to a track, each column to a detection. Values indicate soft assignment probabilities. 47
Figure 3.8: Track state updates using soft assignments from the Sinkhorn algorithm. Each track receives a weighted combination of detection inputs rather than a hard one-to-one match. 48
Figure 3.9: Each track is updated using a convex combination of detections, visualized here as color-coded contributions. The proportions reflect the Sinkhorn weights P_{ij} applied to detection inputs. 49
Figure 4.1: CenterPoint performance on nuScenes for voxel sizes 0.075 and 0.1 across mAP and NDS metrics. 52
Figure 4.2: Side-by-side comparison of detection outputs from two CenterPoint models. Green boxes highlights ground truth locations, the red boxes highlights the detected locations. Region where performance differs are highlighted in the second picture with black boxes. 53

Figure 4.3: PolyMOT tracking performance on the nuScenes validation set using CenterPoint detections (voxel size 0.075) and ground-truth detections.................................. 54
Figure 4.4: Comprehensive comparison of tracking performance across PolyMOT and 3DMOTFormer variants on the nuScenes validation set.. 56
Figure 5.1: AMOTA–AMOTP trade-space for the two tracker families. Hollow markers denote per-class results; filled markers give the class-averaged overall scores. Points closer to the upper-left corner are preferable... 64
Figure 5.2: Final displacement error (FDE) accumulation at 6 seconds across different pipeline configurations. The plot visualizes how prediction error increases as input quality degrades from ideal conditions. The baseline is the ground-truth input setting (GT), followed by sequential pipelines with various tracking models, both trained on GT and real detection data. Each bar shows the incremental increase in FDE relative to the previous configuration. Notably, the largest jumps occur when switching from ground truth to detection inputs and when using a learnable tracker. The joint training setup results in the highest FDE, reflecting full error propagation from detection and tracking stages. 70
Figure 5.3: Final displacement error (FDE) accumulation at 6 seconds across pedestrian prediction pipelines. This chart illustrates how FDE increases as more noise is introduced to the upstream modules. The baseline uses pure ground truth inputs. Minimal error is introduced by tracking using GT-trained 3DMOTFormer and PolyMOT. The largest increase appears when switching from GT to detection inputs. Joint training shows the highest error due to full propagation of upstream imperfections. 71

List of Tables

Table 2.1: Compact Comparison of Major 3D Perception Datasets 6

Table 2.2: Comparison of 3D object detection models on the nuScenes test set. All results (mAP and NDS) are reported from the respective original papers or official leaderboard entries[1]. 14

Table 2.3: Comparison of 3D multi-object tracking (MOT) methods on the nuScenes test set. All results (AMOTA, AMOTP, and ID switches) are reported from their respective papers or official benchmark entries. 18

Table 2.4: Performance comparison of selected trajectory prediction models on the nuScenes validation set (6-second horizon). Metrics include minimum Average Displacement Error (minADE_5), minimum Final Displacement Error (minFDE_1), and Miss Rate@2m with top-5 predicted trajectories. All values are reported from the models' original publications. 23

Table 2.5: Overview of notable LiDAR-based joint perception and prediction models. Each entry lists the integrated tasks (Detection, Tracking, Prediction), a summary of key innovations, and highlights end-to-end learning strategies or architecture features. All models operate primarily on LiDAR data and aim to unify components of the autonomous driving perception stack. 26

Table 4.1: (a) Vehicle results from ground truth input at 4s and 6s 57

Table 4.2: (a) Pedestrian results from ground truth input at 4s and 6s 57

Table 4.3: Overall vehicle pipeline comparison. Each row corresponds to a different input setup (e.g., ground-truth vs. detector output). The following metrics are reported (a) Detection metrics: mAP, NDS, (b) Tracking metrics: AMOTA, AMOTP, ID switches (IDs), and (c) Prediction metrics: ADE, FDE, and MissRate at 4s and 6s. Abbreviations: D = Detection, T = Tracking, P = Prediction. 59

Table 4.4: Overall pedestrian pipeline comparison. Each row corresponds to a different input setup (e.g., ground-truth vs. detector output). The following metrics are reported (a) Detection metrics: mAP, NDS, (b) Tracking metrics: AMOTA, AMOTP, ID switches (IDs), and (c) Prediction metrics: ADE, FDE, and MissRate at 4s and 6s. Abbreviations: D = Detection, T = Tracking, P = Prediction. 59

Table 4.5: Effect of different training configurations on tracking performance. AMOTA (↑), AMOTP (↓), and ID switches (IDS) on the nuScenes validation set are reported. Abbreviations: LR = Learning rate. 60

Table 5.1: Comparison between the best sequential pipeline (D→T→P) and joint training (T+P) results for vehicles and pedestrians. Only tracking and prediction metrics are reported, for direct comparison, since detection input is the same for both. 72

Table A.1: Comparison of two CenterPoint models on nuScenes with voxel sizes 0.075 and 0.1. Mean AP (mAP) and nuScenes Detection Score (NDS) for Vehicle (Car, Truck, Bus, Trailer), Pedestrian, and Overall (across all classes) are reported. Values where Voxel=0.075 outperforms Voxel=0.1 are highlighted in bold. xii

Table A.2: Tracking performance of PolyMOT using CenterPoint detections (voxel size 0.075) on nuScenes. Metrics reported: AMOTA, AMOTP, and identity switches (IDS) for Vehicle, Pedestrian, and Overall. xii
Table A.3: Tracking performance of PolyMOT using ground-truth detections on nuScenes. Metrics reported: AMOTA, AMOTP, and identity switches (IDS) for Vehicle, Pedestrian, and Overall........ xii
Table A.4: Tracking performance of 3DMOTFormer trained and evaluated on CenterPoint(voxel size 0.075) detection results on nuScenes per class. Metrics reported: AMOTA, AMOTP, and ID switches (IDS). xii
Table A.5: Tracking performance of 3DMOTFormer, trained and evaluated on ground truth detections on nuScenes per class. Metrics reported: AMOTA, AMOTP, and ID switches (IDS)........ xiii
Table A.6: Tracking performance of CenterPoint detections (trained on GT) on nuScenes per class. Metrics reported: AMOTA, AMOTP, and ID switches (IDS). xiii
Table A.7: Tracking performance of CenterPoint detections, of the joint training model on nuScenes per class. Metrics reported: AMOTA, AMOTP, and ID switches (IDS). xiii
Table A.8: (a) Vehicle results from tracking output: voxel0075 at 4s and 6s xiii
Table A.9: (a) Vehicle results from tracking output after fine tuning base model only: voxel0075 at 6s xiii
Table A.10: (a) Vehicle results from tracking learning-based output: voxel0075 at 4s and 6s xiv
Table A.11: (a) Vehicle results from tracking output learning based after fine tuning base model only: voxel0075 at 6s xiv
Table A.12: (a) Pedestrian results from tracking output: voxel0075 at 4s and 6s........ xiv
Table A.13: (a) Pedestrian results from tracking output after fine tuning base model only: voxel0075 at 6s xiv
Table A.14: (a) Pedestrian results from tracking learning based output: voxel0075 at 4s and 6s.... xv
Table A.15: (a) Pedestrian results from tracking output learning based after fine tuning base model only: voxel0075 at 6s xv

Bibliography

[1] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, et al. “*nuScenes: A multimodal dataset for autonomous driving*," 2020. arXiv: 1903.11027 `[cs.LG]`. Available: https://arxiv.org/abs/1903.11027.

[2] Y. Liao, J. Xie and A. Geiger. “*KITTI-360: A Novel Dataset and Benchmarks for Urban Scene Understanding in 2D and 3D*," 2022. arXiv: 2109.13410 `[cs.CV]`. Available: https://arxiv.org/abs/2109.13410.

[3] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, et al. “*Scalability in Perception for Autonomous Driving: Waymo Open Dataset*," 2020. arXiv: 1912.04838 `[cs.CV]`. Available: https://arxiv.org/abs/1912.04838.

[4] A. Geiger, P. Lenz, C. Stiller and R. Urtasun, “Vision meets Robotics: The KITTI Dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[5] J. Fritsch, T. Kuehnl and A. Geiger, “A New Performance Measure and Evaluation Benchmark for Road Detection Algorithms," in *International Conference on Intelligent Transportation Systems (ITSC)*, 2013.

[6] M.-F. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, et al. “*Argoverse: 3D Tracking and Forecasting with Rich Maps*," 2019. arXiv: 1911.02620 `[cs.CV]`. Available: https://arxiv.org/abs/1911.02620.

[7] B. Wilson, W. Qi, T. Agarwal, J. Lambert, J. Singh, et al. “*Argoverse 2: Next Generation Datasets for Self-Driving Perception and Forecasting*," 2023. arXiv: 2301.00493 `[cs.CV]`. Available: https://arxiv.org/abs/2301.00493.

[8] G. Li, Y. Jiao, V. L. Knoop, S. C. Calvert and J. W. C. van Lint. “*Large Car-following Data Based on Lyft level-5 Open Dataset: Following Autonomous Vehicles vs. Human-driven Vehicles*," 2023. arXiv: 2305.18921 `[eess.SY]`. Available: https://arxiv.org/abs/2305.18921.

[9] X. Huang, P. Wang, X. Cheng, D. Zhou, Q. Geng, et al., “The ApolloScape Open Dataset for Autonomous Driving and Its Application," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 10, pp. 2702–2719, 2020, DOI: 10.1109/tpami.2019.2926463. Available: http://dx.doi.org/10.1109/TPAMI.2019.2926463.

[10] P. Xiao, Z. Shao, S. Hao, Z. Zhang, X. Chai, et al. “*PandaSet: Advanced Sensor Suite Dataset for Autonomous Driving*," 2021. arXiv: 2112.12610 `[cs.CV]`. Available: https://arxiv.org/abs/2112.12610.

[11] J. Mao, M. Niu, C. Jiang, H. Liang, J. Chen, et al. “*One Million Scenes for Autonomous Driving: ONCE Dataset*," 2021. arXiv: 2106.11037 `[cs.CV]`. Available: https://arxiv.org/abs/2106.11037.

[12] Q.-H. Pham, P. Sevestre, R. S. Pahwa, H. Zhan, C. H. Pang, et al. “*A*3D Dataset: Towards Autonomous Driving in Challenging Environments*," 2019. arXiv: 1909.07541 `[cs.CV]`. Available: https://arxiv.org/abs/1909.07541.

[13] A. Patil, S. Malla, H. Gang and Y.-T. Chen. “*The H3D Dataset for Full-Surround 3D Multi-Object Detection and Tracking in Crowded Urban Scenes*," 2019. arXiv: 1903.01568 `[cs.CV]`. Available: https://arxiv.org/abs/1903.01568.

[14] K. Bernardin and R. Stiefelhagen, “Evaluating multiple object tracking performance: The CLEAR MOT metrics," *EURASIP Journal on Image and Video Processing*, vol. 2008, 2008, DOI: 10.1155/2008/246309.

[15] X. Weng, J. Wang, D. Held and K. Kitani. “*3D Multi-Object Tracking: A Baseline and New Evaluation Metrics*," 2020. arXiv: 1907.03961 `[cs.CV]`. Available: https://arxiv.org/abs/1907.03961.

[16] P. Tran, H. Wu, C. Yu, P. Cai, S. Zheng, et al. “*What Truly Matters in Trajectory Prediction for Autonomous Driving?*," 2023. arXiv: 2306.15136 `[cs.RO]`. Available: https://arxiv.org/abs/2306.15136.

[17] R. Huang, G. Zhuo, L. Xiong, S. Lu and W. Tian, “A Review of Deep Learning-Based Vehicle Motion Prediction for Autonomous Driving," *Sustainability*, vol. 15, no. 20, 2023, DOI: 10.3390/su152014716. Available: https://www.mdpi.com/2071-1050/15/20/14716.

[18] Y. Zhou and O. Tuzel. “*VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection*," 2017. arXiv: 1711.06396 `[cs.CV]`. Available: https://arxiv.org/abs/1711.06396.

[19] Y. Yan, Y. Mao and B. Li, “SECOND: Sparsely Embedded Convolutional Detection," *Sensors*, vol. 18, no. 10, 2018, DOI: 10.3390/s18103337. Available: https://www.mdpi.com/1424-8220/18/10/3337.

[20] T. Yin, X. Zhou and P. Krähenbühl. “*Center-based 3D Object Detection and Tracking*," 2021. arXiv: 2006.11275 `[cs.CV]`. Available: https://arxiv.org/abs/2006.11275.

[21] G. Shi, R. Li and C. Ma. “*PillarNet: Real-Time and High-Performance Pillar-based 3D Object Detection*," 2022. arXiv: 2205.07403 `[cs.CV]`. Available: https://arxiv.org/abs/2205.07403.

[22] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, et al. “*PointPillars: Fast Encoders for Object Detection from Point Clouds*," 2019. arXiv: 1812.05784 `[cs.LG]`. Available: https://arxiv.org/abs/1812.05784.

[23] Y. Chen, J. Liu, X. Zhang, X. Qi and J. Jia. “*VoxelNeXt: Fully Sparse VoxelNet for 3D Object Detection and Tracking*," 2023. arXiv: 2303.11301 `[cs.CV]`. Available: https://arxiv.org/abs/2303.11301.

[24] H. Wang, C. Shi, S. Shi, M. Lei, S. Wang, et al. “*DSVT: Dynamic Sparse Voxel Transformer with Rotated Sets*," 2023. arXiv: 2301.06051 `[cs.CV]`. Available: https://arxiv.org/abs/2301.06051.

[25] Z. Liu, J. Hou, X. Wang, X. Ye, J. Wang, et al. “*LION: Linear Group RNN for 3D Object Detection in Point Clouds*," 2024. arXiv: 2407.18232 `[cs.CV]`. Available: https://arxiv.org/abs/2407.18232.

[26] S. Shi, X. Wang and H. Li. “*PointRCNN: 3D Object Proposal Generation and Detection from Point Cloud*," 2019. arXiv: 1812.04244 `[cs.CV]`. Available: https://arxiv.org/abs/1812.04244.

[27] Z. Yang, Y. Sun, S. Liu and J. Jia. “*3DSSD: Point-based 3D Single Stage Object Detector*," 2020. arXiv: 2002.10187 `[cs.CV]`. Available: https://arxiv.org/abs/2002.10187.

[28] S. Shi, L. Jiang, J. Deng, Z. Wang, C. Guo, et al. “*PV-RCNN++: Point-Voxel Feature Set Abstraction With Local Vector Representation for 3D Object Detection*," 2022. arXiv: 2102.00463 `[cs.CV]`. Available: https://arxiv.org/abs/2102.00463.

[29] Z. Liu, H. Tang, A. Amini, X. Yang, H. Mao, et al. “*BEVFusion: Multi-Task Multi-Sensor Fusion with Unified Bird's-Eye View Representation*," 2024. arXiv: 2205.13542 `[cs.CV]`. Available: https://arxiv.org/abs/2205.13542.

[30] X. Bai, Z. Hu, X. Zhu, Q. Huang, Y. Chen, et al. “*TransFusion: Robust LiDAR-Camera Fusion for 3D Object Detection with Transformers*," 2022. arXiv: 2203.11496 `[cs.CV]`. Available: https://arxiv.org/abs/2203.11496.

[31] Haotian, Hu, Fanyi, Wang, Jingwen, et al. “*EA-BEV: Edge-aware Bird' s-Eye-View Projector for 3D Object Detection*," Mar. 2023. DOI: 10.48550/arXiv.2303.17895.

[32] Y. Hu, J. Yang, L. Chen, K. Li, C. Sima, et al. “*Planning-oriented Autonomous Driving*," 2023. arXiv: 2212.10156 `[cs.CV]`. Available: https://arxiv.org/abs/2212.10156.

[33] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, et al. “*CenterNet: Keypoint Triplets for Object Detection*," 2019. arXiv: 1904.08189 `[cs.CV]`. Available: https://arxiv.org/abs/1904.08189.

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

[35] M. Contributors. “*MMDetection3D: OpenMMLab next-generation platform for general 3D object detection*," https://github.com/open-mmlab/mmdetection3d. 2020.

[36] Z. Zhou, X. Zhao, Y. Wang, P. Wang and H. Foroosh. “*CenterFormer: Center-based Transformer for 3D Object Detection*," 2022. arXiv: 2209.05588 `[cs.CV]`. Available: https://arxiv.org/abs/2209.05588.

[37] Y. Pei, S. Biswas, D. S. Fussell and K. Pingali. “*An Elementary Introduction to Kalman Filtering*," 2019. arXiv: 1710.04055 `[eess.SY]`. Available: https://arxiv.org/abs/1710.04055.

[38] F. Rajabi-Alni and A. Bagheri. “*Computing a many-to-many matching with demands and capacities between two sets using the Hungarian algorithm*," 2022. arXiv: 2211.01612 `[cs.DS]`. Available: https://arxiv.org/abs/2211.01612.

[39] X. Li, T. Xie, D. Liu, J. Gao, K. Dai, et al. “*Poly-MOT: A Polyhedral Framework For 3D Multi-Object Tracking*," 2023. arXiv: 2307.16675 `[cs.RO]`. Available: https://arxiv.org/abs/2307.16675.

[40] X. Li, D. Liu, Y. Wu, X. Wu, L. Zhao, et al. “*Fast-Poly: A Fast Polyhedral Framework For 3D Multi-Object Tracking*," 2024. arXiv: 2403.13443 `[cs.CV]`. Available: https://arxiv.org/abs/2403.13443.

[41] X. Weng, J. Wang, D. Held and K. Kitani. “*AB3DMOT: A Baseline for 3D Multi-Object Tracking and New Evaluation Metrics*," 2020. arXiv: 2008.08063 `[cs.CV]`. Available: https://arxiv.org/abs/2008.08063.

[42] A. Kim, A. Ošep and L. Leal-Taixé. “*EagerMOT: 3D Multi-Object Tracking via Sensor Fusion*," 2021. arXiv: 2104.14682 `[cs.CV]`. Available: https://arxiv.org/abs/2104.14682.

[43] H.-k. Chiu, J. Li, R. Ambrus and J. Bohg. “*Probabilistic 3D Multi-Modal, Multi-Object Tracking for Autonomous Driving*," 2021. arXiv: 2012.13755 `[cs.CV]`. Available: https://arxiv.org/abs/2012.13755.

[45] Y. Chen, J. Liu, X. Zhang, X. Qi and J. Jia. “*LargeKernel3D: Scaling up Kernels in 3D Sparse CNNs*," 2023. arXiv: 2206.10555 `[cs.CV]`. Available: https://arxiv.org/abs/2206.10555.

[46] L. Wang, X. Zhang, W. Qin, X. Li, L. Yang, et al. “*CAMO-MOT: Combined Appearance-Motion Optimization for 3D Multi-Object Tracking with Camera-LiDAR Fusion*," 2022. arXiv: 2209.02540 `[cs.CV]`. Available: https://arxiv.org/abs/2209.02540.

[47] J.-N. Zaech, D. Dai, A. Liniger, M. Danelljan and L. V. Gool. “*Learnable Online Graph Representations for 3D Multi-Object Tracking*," 2021. arXiv: 2104.11747 `[cs.CV]`. Available: https://arxiv.org/abs/2104.11747.

[48] S. Ding, E. Rehder, L. Schneider, M. Cordts and J. Gall. “*3DMOTFormer: Graph Transformer for Online 3D Multi-Object Tracking*," 2023. arXiv: 2308.06635 `[cs.CV]`. Available: https://arxiv.org/abs/2308.06635.

[49] Z. Pang, Z. Li and N. Wang. “*SimpleTrack: Understanding and Rethinking 3D Multi-object Tracking*," 2021. arXiv: 2111.09621 `[cs.CV]`. Available: https://arxiv.org/abs/2111.09621.

[50] X. Zhou, V. Koltun and P. Krähenbühl. “*Tracking Objects as Points*," 2020. arXiv: 2004.01177 `[cs.CV]`. Available: https://arxiv.org/abs/2004.01177.

[51] J. Gwak, S. Savarese and J. Bohg. “*Minkowski Tracker: A Sparse Spatio-Temporal R-CNN for Joint Object Detection and Tracking*," 2022. arXiv: 2208.10056 `[cs.CV]`. Available: https://arxiv.org/abs/2208.10056.

[52] B. Cheong, J. Zhou and S. Waslander. “*JDT3D: Addressing the Gaps in LiDAR-Based Tracking-by-Attention*," 2024. arXiv: 2407.04926 `[cs.CV]`. Available: https://arxiv.org/abs/2407.04926.

[54] B. Ivanovic and M. Pavone. “*The Trajectron: Probabilistic Multi-Agent Trajectory Modeling With Dynamic Spatiotemporal Graphs*," 2019. arXiv: 1810.05993 `[cs.RO]`. Available: https://arxiv.org/abs/1810.05993.

[55] T. Salzmann, B. Ivanovic, P. Chakravarty and M. Pavone. “*Trajectron++: Dynamically-Feasible Trajectory Forecasting With Heterogeneous Data*," 2021. arXiv: 2001.03093 `[cs.RO]`. Available: https://arxiv.org/abs/2001.03093.

[56] T. Phan-Minh, E. C. Grigore, F. A. Boulton, O. Beijbom and E. M. Wolff. “*CoverNet: Multimodal Behavior Prediction using Trajectory Sets*," 2020. arXiv: 1911.10298 `[cs.LG]`. Available: https://arxiv.org/abs/1911.10298.

[57] S. Khandelwal, W. Qi, J. Singh, A. Hartnett and D. Ramanan. “*What-If Motion Prediction for Autonomous Driving*," 2020. arXiv: 2008.10587 `[cs.LG]`. Available: https://arxiv.org/abs/2008.10587.

[58] Y. Chai, B. Sapp, M. Bansal and D. Anguelov. “*MultiPath: Multiple Probabilistic Anchor Trajectory Hypotheses for Behavior Prediction*," 2019. arXiv: 1910.05449 `[cs.LG]`. Available: https://arxiv.org/abs/1910.05449.

[59] H. Cui, V. Radosavljevic, F.-C. Chou, T.-H. Lin, T. Nguyen, et al. “*Multimodal Trajectory Predictions for Autonomous Driving using Deep Convolutional Networks*," 2019. arXiv: 1809.10732 `[cs.RO]`. Available: https://arxiv.org/abs/1809.10732.

[60] H. Zhao, J. Gao, T. Lan, C. Sun, B. Sapp, et al. “*TNT: Target-driveN Trajectory Prediction*," 2020. arXiv: 2008.08294 `[cs.CV]`. Available: https://arxiv.org/abs/2008.08294.

[61] Z. Yao, X. Li, B. Lang and M. C. Chuah, “Goal-LBP: Goal-Based Local Behavior Guided Trajectory Prediction for Autonomous Driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 25, no. 7, pp. 6770–6779, 2024, DOI: 10.1109/TITS.2023.3342706.

[62] M. Liang, B. Yang, R. Hu, Y. Chen, R. Liao, et al. “*Learning Lane Graph Representations for Motion Forecasting*," 2020. arXiv: 2007.13732 `[cs.CV]`. Available: https://arxiv.org/abs/2007.13732.

[63] J. Gao, C. Sun, H. Zhao, Y. Shen, D. Anguelov, et al. “*VectorNet: Encoding HD Maps and Agent Dynamics from Vectorized Representation*," 2020. arXiv: 2005.04259 `[cs.CV]`. Available: https://arxiv.org/abs/2005.04259.

[64] Y. Yuan, X. Weng, Y. Ou and K. Kitani. “*AgentFormer: Agent-Aware Transformers for Socio-Temporal Multi-Agent Forecasting*," 2021. arXiv: 2103.14023 `[cs.AI]`. Available: https://arxiv.org/abs/2103.14023.

[65] J. Ngiam, B. Caine, V. Vasudevan, Z. Zhang, H.-T. L. Chiang, et al. “*Scene Transformer: A unified architecture for predicting multiple agent trajectories*," 2022. arXiv: 2106.08417 `[cs.CV]`. Available: https://arxiv.org/abs/2106.08417.

[66] T. Gilles, S. Sabatini, D. Tsishkou, B. Stanciulescu and F. Moutarde. “*THOMAS: Trajectory Heatmap Output with learned Multi-Agent Sampling*," 2022. arXiv: 2110.06607 `[cs.CV]`. Available: https://arxiv.org/abs/2110.06607.

[67] S. Shi, L. Jiang, D. Dai and B. Schiele. “*Motion Transformer with Global Intention Localization and Local Movement Refinement*," 2023. arXiv: 2209.13508 `[cs.CV]`. Available: https://arxiv.org/abs/2209.13508.

[68] L. Feng, M. Bahari, K. M. B. Amor, É. Zablocki, M. Cord, et al. “*UniTraj: A Unified Framework for Scalable Vehicle Trajectory Prediction*," 2024. arXiv: 2403.15098 `[cs.CV]`. Available: https://arxiv.org/abs/2403.15098.

[69] Z. Sun, Z. Wang, L. Halilaj and J. Luettin. “*SemanticFormer: Holistic and Semantic Traffic Scene Representation for Trajectory Prediction using Knowledge Graphs*," 2024. arXiv: 2404.19379 `[cs.CV]`. Available: https://arxiv.org/abs/2404.19379.

[70] B. Kim, S. H. Park, S. Lee, E. Khoshimjonov, D. Kum, et al. “*LaPred: Lane-Aware Prediction of Multi-Modal Future Trajectories of Dynamic Agents*," 2021. arXiv: 2104.00249 `[cs.CV]`. Available: https://arxiv.org/abs/2104.00249.

[72] M. Schäfer, K. Zhao and A. Kummert. “*CASPNet++: Joint Multi-Agent Motion Prediction*," 2023. arXiv: 2308.07751 `[cs.CV]`. Available: https://arxiv.org/abs/2308.07751.

[73] N. Peri, J. Luiten, M. Li, A. Ošep, L. Leal-Taixé, et al. “*Forecasting from LiDAR via Future Object Detection*," 2022. arXiv: 2203.16297 `[cs.CV]`. Available: https://arxiv.org/abs/2203.16297.

[74] W. Luo, B. Yang and R. Urtasun. “*Fast and Furious: Real Time End-to-End 3D Detection, Tracking and Motion Forecasting with a Single Convolutional Net*," 2020. arXiv: 2012.12395 `[cs.CV]`. Available: https://arxiv.org/abs/2012.12395.

[75] S. Casas, W. Luo and R. Urtasun. “*IntentNet: Learning to Predict Intention from Raw Sensor Data*," 2021. arXiv: 2101.07907 `[cs.RO]`. Available: https://arxiv.org/abs/2101.07907.

[76] S. Casas, C. Gulino, R. Liao and R. Urtasun. “*Spatially-Aware Graph Neural Networks for Relational Behavior Forecasting from Sensor Data*," 2019. arXiv: 1910.08233 `[cs.CV]`. Available: https://arxiv.org/abs/1910.08233.

[77] G. P. Meyer, J. Charland, S. Pandey, A. Laddha, S. Gautam, et al. “*LaserFlow: Efficient and Probabilistic Object Detection and Motion Forecasting*," 2020. arXiv: 2003.05982 `[cs.CV]`. Available: https://arxiv.org/abs/2003.05982.

[78] A. Laddha, S. Gautam, G. P. Meyer, C. Vallespi-Gonzalez and C. K. Wellington. “*RV-FuseNet: Range View Based Fusion of Time-Series LiDAR Data for Joint 3D Object Detection and Motion Forecasting*," 2021. arXiv: 2005.10863 `[cs.CV]`. Available: https://arxiv.org/abs/2005.10863.

[80] P. Wu, S. Chen and D. Metaxas. “*MotionNet: Joint Perception and Motion Prediction for Autonomous Driving Based on Bird’s Eye View Maps*," 2020. arXiv: 2003.06754 `[cs.CV]`. Available: https://arxiv.org/abs/2003.06754.

[81] N. Djuric, H. Cui, Z. Su, S. Wu, H. Wang, et al. “*MultiXNet: Multiclass Multistage Multimodal Motion Prediction*," 2021. arXiv: 2006.02000 `[cs.CV]`. Available: https://arxiv.org/abs/2006.02000.

[82] J. Zhuang, G. Wang, S. Zhang, X. Wang, H. Zhou, et al. “*StreamMOTP: Streaming and Unified Framework for Joint 3D Multi-Object Tracking and Trajectory Prediction*," 2024. arXiv: 2406.19844 `[cs.CV]`. Available: https://arxiv.org/abs/2406.19844.

[83] M. Cuturi. “*Sinkhorn Distances: Lightspeed Computation of Optimal Transportation Distances*," 2013. arXiv: 1306.0895 `[stat.ML]`. Available: https://arxiv.org/abs/1306.0895.

[84] M. Liang, B. Yang, W. Zeng, Y. Chen, R. Hu, et al. “*PnPNet: End-to-End Perception and Prediction with Tracking in the Loop*," 2020. arXiv: 2005.14711 `[cs.CV]`. Available: https://arxiv.org/abs/2005.14711.

[86] Z. Zhang, J. Gao, J. Mao, Y. Liu, D. Anguelov, et al. “*STINet: Spatio-Temporal-Interactive Network for Pedestrian Detection and Trajectory Prediction*," 2020. arXiv: 2005.04255 `[cs.CV]`. Available: https://arxiv.org/abs/2005.04255.

[87] Y. Xu, L. Chambon, É. Zablocki, M. Chen, A. Alahi, et al. “*Towards Motion Forecasting with Real-World Perception Inputs: Are End-to-End Approaches Competitive?*," 2024. arXiv: 2306.09281 `[cs.RO]`. Available: https://arxiv.org/abs/2306.09281.

[88] J. Gu, C. Hu, T. Zhang, X. Chen, Y. Wang, et al. “*ViP3D: End-to-end Visual Trajectory Prediction via 3D Agent Queries*," 2023. arXiv: 2208.01582 `[cs.CV]`. Available: https://arxiv.org/abs/2208.01582.

[89] D. Merkel, “Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[90] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, et al. "*PyTorch: An Imperative Style, High-Performance Deep Learning Library*," 2019. arXiv: 1912.01703 `[cs.LG]`. Available: https://arxiv.org/abs/1912.01703.

[94] P. A. Knight, "The Sinkhorn–Knopp Algorithm: Convergence and Applications," *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 1, pp. 261–275, 2008, DOI: 10.1137/060659624. eprint: https://doi.org/10.1137/060659624. Available: https://doi.org/10.1137/060659624.

Appendix

A Appendix xii

A Appendix

Table A.1: Comparison of two CenterPoint models on nuScenes with voxel sizes 0.075 and 0.1. Mean AP (mAP) and nuScenes Detection Score (NDS) for Vehicle (Car, Truck, Bus, Trailer), Pedestrian, and Overall (across all classes) are reported. Values where Voxel=0.075 outperforms Voxel=0.1 are highlighted in bold.

Category	**Voxel=0.075**		**Voxel=0.1**	
	mAP	**NDS**	**mAP**	**NDS**
Vehicle	**0.6211**	**0.6934**	0.5991	0.6790
Pedestrian	**0.8528**	**0.8157**	0.8301	0.7996
Overall (All Classes)	**0.592**	**0.668**	0.556	0.644

Table A.2: Tracking performance of PolyMOT using CenterPoint detections (voxel size 0.075) on nuScenes. Metrics reported: AMOTA, AMOTP, and identity switches (IDS) for Vehicle, Pedestrian, and Overall.

Class / Metric	**AMOTA**	**AMOTP**	**IDS**
Vehicle	0.733	0.603	79
Pedestrian	0.823	0.352	201
Overall	0.714	0.529	288

Table A.3: Tracking performance of PolyMOT using ground-truth detections on nuScenes. Metrics reported: AMOTA, AMOTP, and identity switches (IDS) for Vehicle, Pedestrian, and Overall.

Class / Metric	**AMOTA**	**AMOTP**	**IDS**
Vehicle	0.887	0.103	124
Pedestrian	0.924	0.112	123
Overall	**0.897**	**0.105**	**247**

Table A.4: Tracking performance of 3DMOTFormer trained and evaluated on CenterPoint(voxel size 0.075) detection results on nuScenes per class. Metrics reported: AMOTA, AMOTP, and ID switches (IDS).

Class / Metric	**AMOTA**	**AMOTP**	**IDS**
Vehicle	0.7045	0.62625	322
Pedestrian	0.738	0.478	584
Overall	**0.621**	**0.622**	**937**

Table A.5: Tracking performance of 3DMOTFormer, trained and evaluated on ground truth detections on nuScenes per class. Metrics reported: AMOTA, AMOTP, and ID switches (IDS).

Class / Metric	AMOTA	AMOTP	IDS
Vehicle	0.9495	0.06525	322
Pedestrian	0.949	0.077	50
Overall	**0.940**	**0.077**	**343**

Table A.6: Tracking performance of CenterPoint detections (trained on GT) on nuScenes per class. Metrics reported: AMOTA, AMOTP, and ID switches (IDS).

Class / Metric	AMOTA	AMOTP	IDS
Vehicle	0.004	1.20525	9851
Pedestrian	0.059	0.950	1750
Overall	**0.009**	**1.116**	**12381**

Table A.7: Tracking performance of CenterPoint detections, of the joint training model on nuScenes per class. Metrics reported: AMOTA, AMOTP, and ID switches (IDS).

Class / Metric	AMOTA	AMOTP	IDS
Vehicle	0.6752	0.662	1009
Pedestrian	0.661	0.604	961
Overall	**0.593**	**0.655**	**1970**

Table A.8: (a) Vehicle results from tracking output: voxel0075 at 4s and 6s

VEHICLE

Ablation			**ADE (ml)**		**FDE (ml)**		**MissRate (%)**	
$\int$	M	y_R	@4s	@6s	@4s	@6s	@4s	@6s
-	-	-	**1.6585**	**2.9609**	**3.5524**	**6.8904**	**49.58**	**58.49**
✓	-	-	2.3394	4.5807	5.4099	11.4811	61.65	66.60
✓	✓	-	1.9441	3.5239	4.2362	8.3021	56.03	63.90
✓	✓	✓	1.8400	3.3152	3.9671	7.7898	52.04	59.21

Legend: $\int$ = Integration via Dynamics, M = Map Encoding, y_R = Robot Future Encoding

Table A.9: (a) Vehicle results from tracking output after fine tuning base model only: voxel0075 at 6s

VEHICLE

Ablation			**ADE (ml)**		**FDE (ml)**		**MissRate (%)**	
$\int$	M	y_R	@4s	@6s	@4s	@6s	@4s	@6s
-	-	-	**1.6780**	**3.0686**	**3.6329**	**7.3062**	**49.97**	**58.64**

Legend: $\int$ = Integration via Dynamics, M = Map Encoding, y_R = Robot Future Encoding

Table A.10: (a) Vehicle results from tracking learning-based output: voxel0075 at 4s and 6s

VEHICLE

Ablation			ADE (ml)		FDE (ml)		MissRate (%)	
$\int$	M	y_R	@4s	@6s	@4s	@6s	@4s	@6s
-	-	-	**1.7908**	**3.1460**	**3.7701**	**7.2290**	**51.56**	**60.49**
√	-	-	2.5425	4.8965	5.7847	12.1287	64.03	68.93
√	√	-	2.1242	3.7871	4.5459	8.8102	58.05	66.10
√	√	√	2.0355	3.5841	4.2810	8.2706	54.66	61.91

Legend: $\int$ = Integration via Dynamics, M = Map Encoding, y_R = Robot Future Encoding

Table A.11: (a) Vehicle results from tracking output learning based after fine tuning base model only: voxel0075 at 6s

VEHICLE

Ablation			ADE (ml)		FDE (ml)		MissRate (%)	
$\int$	M	y_R	@4s	@6s	@4s	@6s	@4s	@6s
-	-	-	**1.8104**	**3.2583**	**3.8479**	**7.6698**	**52.09**	**60.71**

Legend: $\int$ = Integration via Dynamics, M = Map Encoding, y_R = Robot Future Encoding

Table A.12: (a) Pedestrian results from tracking output: voxel0075 at 4s and 6s

PEDESTRIAN

Ablation			ADE (ml)		FDE (ml)		MissRate (%)	
$\int$	M	y_R	@4s	@6s	@4s	@6s	@4s	@6s
-	-	-	**0.6690**	**0.9896**	**1.1937**	**1.9070**	**18.13**	**33.66**
√	-	-	0.6963	1.0413	1.2558	2.0329	19.63	36.34
√	√	-	0.6950	1.0290	1.2419	1.9863	19.76	36.21

Legend: $\int$ = Integration via Dynamics, M = Map Encoding

Table A.13: (a) Pedestrian results from tracking output after fine tuning base model only: voxel0075 at 6s

PEDESTRIAN

Ablation			ADE (ml)		FDE (ml)		MissRate (%)	
$\int$	M	y_R	@4s	@6s	@4s	@6s	@4s	@6s
-	-	-	**0.7445**	**1.0466**	**1.3343**	**2.0135**	**20.71**	**34.60**

Legend: $\int$ = Integration via Dynamics, M = Map Encoding, y_R = Robot Future Encoding

Table A.14: (a) Pedestrian results from tracking learning based output: voxel0075 at 4s and 6s

PEDESTRIAN								
Ablation			**ADE (ml)**		**FDE (ml)**		**MissRate (%)**	
$\int$	M	y_R	@4s	@6s	@4s	@6s	@4s	@6s
-	-	-	**0.7167**	**1.0468**	**1.2565**	**1.9900**	**19.98**	**35.99**
✓	-	-	0.7465	1.1022	1.3236	2.1239	21.24	38.79
✓	✓	-	0.7624	1.1125	1.3379	2.1128	22.41	38.80

Legend: $\int$ = Integration via Dynamics, M = Map Encoding

Table A.15: (a) Pedestrian results from tracking output learning based after fine tuning base model only: voxel0075 at 6s

PEDESTRIAN								
Ablation			**ADE (ml)**		**FDE (ml)**		**MissRate (%)**	
$\int$	M	y_R	@4s	@6s	@4s	@6s	@4s	@6s
-	-	-	**0.8173**	**1.1072**	**1.4403**	**2.0994**	**24.13**	**36.81**

Legend: $\int$ = Integration via Dynamics, M = Map Encoding, y_R = Robot Future Encoding