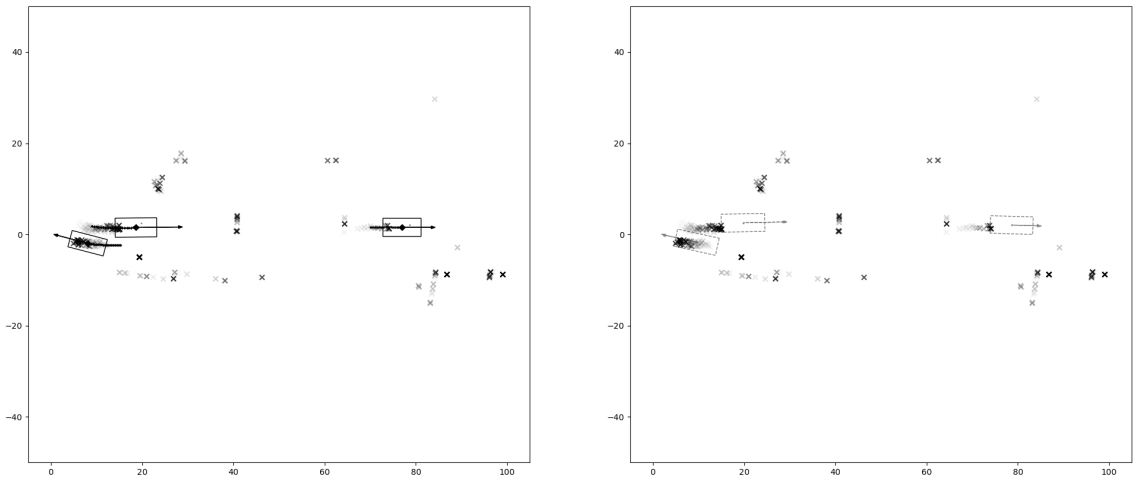


Extending tracking with deep learning on radar detections

End-to-end tracking on simulated data

Haik-David Avetian
Emil Hedlund

Master's thesis in Complex Adaptive Systems

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2024
www.chalmers.se

Extending tracking with deep learning on radar detections

End-to-end tracking on simulated data

HAIK-DAVID AVETIAN
EMIL HEDLUND

CHALMERS

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2024

Extending tracking with deep learning on radar detections

End-to-end tracking on simulated data

HAIK-DAVID AVETIAN
EMIL HEDLUND

Supervisor and examiner at CSE: Lars Hammarstrand
Supervisor at SafeRadar Research: Johan Degerman

Department of Electical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 (0)31-772 1000

Cover: Separated ground truth (left) and estimation of the state space made by the MET3v2 network (right) on a sequence generated from CARLA after training the MET3v2 network as described in 3.4.

Keywords: Multi object tracking, Radar tracking, Extended tracking, Transformer networks, Detection transformer, Deep machine learning

Typeset in LaTeX
Gothenburg, Sweden 2024

Abstract

Recent advancements in machine learning suggest that adopting an end-to-end deep learning approach for multi object tracking in radar applications could be advantageous compared to the current methods based on Bayesian statistics. Utilizing and building upon the Detection Transformers architecture, developed for images, the MultiTarget Tracking Transformer v2 (MT3v2) was developed to handle point object tracking radar data in a model-based environment with promising potential for extended object tracking tasks.

This work sets out to test previous proof-of-concept and expand into the domain of reality with the MT3v2 network utilized as a base architecture. A semi-model-free environment was used, with radar data generated from the CARLA traffic simulator to test various modes of tracking. Altering the architecture of the MT3v2 network to enable handling of extended objects, making it compatible with a multi extended object tracking scenario. To evaluate the deep learning approach to see if it is comparable with state of the art tracking method, a generalized optimal subpattern assignment metric is used for extended objects to grade the different trackers.

During training and evaluation, the results shows that the MET3v2 network is able to learn from simulated radar data to enhance tracking performance over time. The results also suggest that a model-free approach when working with multiple extended object tracking problems for radar detections could be used to yield improved tracking performance.

Acknowledgements

We want to begin with thanking our supervisor at Chalmers University of Technology Lars Hammarstrand for valuable input related to the project as well as help with the administrative parts of writing a thesis.

We would also like to thank the entire team at SafeRadar Research for answering our questions on a daily basis sharing their expertise in radar and radar tracking. We would especially want to thank our supervisor at SafeRadar Research, Johan Degerman as well as Alexander Åström for giving valuable input and engaging in discussions revolving around this project. Without your support and willingness to help, this work would have lost considerable quality.

Haik-David Avetian & Emil Hedlund, August 2024

Contents

List of Figures

List of Tables

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AI	Artificial Intelligence
CARLA	CAR Learning to Act
CIoU	Complete Intersection over Union
CNN	Convolutional Neural Network
DETR	DEtection TRansformer
DL	Deep Learning
EOT	Extended Object Tracking
FFN	Feedforward Neural Network
FMCW	Frequency-Modulated Continuous-Wave
FOV	Field Of View
GGIW	Gamma Gaussian Inverse Wishart
GOSPA	Generalized Optimal Subpattern Assignment
Lidar	Light Detection And Ranging
MBM	Multi-Bernoulli Mixture
MEOT	Multi Extended Object Tracking
MET3v2	Multi Extended Target Tracking Transformer version 2
ML	Machine Learning
MLP	Multi Layer Perceptron
MOT	Multi Object Tracking
MT3v2	Multi Target Tracking Transformer version 2
PMBM	Poisson Multi Bernoulli Mixture
POT	Point Object Tracking
PPP	Poisson Point Process
Radar	Radio Detection And Ranging
RCS	Radar Cross Section
ReLU	Rectified Linear Unit
SNR	Signal to Noise Ratio
SOTA	State Of The Art
TN	Transformer Network

1

Introduction

Significant advancements in Deep Learning (DL) have allowed the domain of image processing, among others, to rapidly gain momentum in recent years. Tasks that earlier required manual labour to complete are now often assisted or completely carried out with DL methods. Among its uses, a common task that has proven useful in a wide variety of applications is multi object tracking (MOT) that predicts the states of objects that are of interest. Use cases are diverse and can include advanced driver-assistance systems (ADAS) [1], traffic surveillance [2], or animal behaviour [3].

This paper aims to investigate MOT with with radio detection and ranging (radar) sensing in a traffic environment. The task involves maintaining the identities and trajectories of multiple objects in dynamic setting, contending with detection uncertainties, intricate data association due to proximity and occlusion, dynamic and unpredictable object motion and identity switches. This differs from domains like images, where the tracking data is more discernible and deterministic, being less ambiguous and more concrete when working with high resolution data.

Images have thus taken the spotlight when it comes to computer vision research and practical applications. This is likely attributable to the rapid advancement of camera technology over the years, resulting in high-resolution images at low cost, analogous to how advancements in GPUs have accelerated deep learning progress. However, this modality has inherent limitations, as while images provide spatial information about the arrangement of objects, they do not convey data about object distances or velocities.

As an alternative to the camera modality, light detection and ranging (lidar) technologies have become integral to tasks such as object detection in autonomous driving systems and topographic mapping. Lidar systems are renowned for their high resolution, accuracy in 3D mapping, and ability to operate effectively both day and night.

On the other hand, radar systems, despite having a longstanding history, have not garnered the same research attention during the recent "Artificial Intelligence (AI) boom" as image-based and lidar technologies. radar offers distinct advantages however, including all-weather capability, long-range detection, and robustness against interference through various filtering processes. While both lidar and radar systems are expensive and complex, integrating radar with deep learning methods could potentially deliver comparable results to lidar, with the added benefits of being more compact and cost-effective.

The current state of the art (SOTA) methodology for implementing MOT faces several

challenges, particularly when using filtering techniques. A variety of filters exist, each employing different approaches, and have demonstrated impressive accuracy and computational efficiency, however they often require extensive parameter tuning to achieve optimal performance. For instance, the Poisson Multi Bernoulli Mixture (PMBM) filter has been shown to effectively track objects when given radar data [4], [5]. The tuning process however, is heavily dependent on the specific characteristics of the radar data, which introduces additional complexity. Given these challenges, it is worth exploring the potential of replacing traditional filtering methods with an end-to-end DL approach.

Transformer architectures are a relatively recent addition to the AI toolbox [6], bringing the attention mechanism to DL. This approach has demonstrated remarkable success in sequence-to-sequence tasks [7] and even image-based detection and tracking [8], [9] in computer vision. Since their introduction, transformers have gained significant momentum, potentially emerging as the new SOTA across various fields where Convolutional Neural Networks (CNNs) have traditionally been the preferred method. Given these results, it is worthwhile to explore whether this DL methodology could serve as an effective alternative to, or even surpass, the current SOTA radar MOT filters that are based on Bayesian statistics.

Recent research on radar tracking using DL methods has shown promising potential, particularly with the application of transformer networks. The MultiTarget Tracking Transformer v2 (MT3v2) as introduced in [5] as a proof-of-concept tracker, demonstrating that synthetic radar data fed into the network could perform competitively with current SOTA tracking filters without the need of mathematical models for predictions. The MT3v2 has only been evaluated in a controlled point object tracking (POT) scenario and has not yet been tested in real-world setting. Reality often entails complex MOT tasks that requires extended object tracking (EOT) since most objects span a volume instead of a point in space.

In an attempt to further bridge the gap between proof-of-concept and practical application, this thesis investigates the performance of the MT3v2 transformer network in a more realistic, stochastic simulated environment requiring solving multi extended object tracking (MEOT). Although the ideal scenario would involve real-world data, the scarcity of data annotations necessitates the use of the CAR Learning to Act (CARLA) traffic simulator, where a simulated radar tool will generate the required data and labels as an intermediary step towards reality. This work extends the research conducted in [5], advancing the development of the MT3v2 network with focus being on assessing how the network adapts to more complex, lifelike data and identifying potential improvements that could bring the network closer to processing real-world data with minimal preprocessing.

By examining the compatibility and application of machine learning in this context, this work seeks to build on the promising results of previous studies, despite the inherent challenges posed by radar's low-resolution and highly correlated data.

1.1 Problem formulation and research question

- How well can a transformer-based architecture learn MEOT from radar data generated by the CARLA simulator?

- How does a transformer-based MEOT approach compare to a current SOTA model based method, evaluated with GOSPA?

1.2 Scope

This paper focuses on training the tracker using data and measurements from a single stationary radar. The data is adapted to be predicting bounding boxes in 2D, although the collected points are registered in 3D. This is done in order to reduce the dimensionality and complexity of the task.

The data is collected from CARLA using an in-house developed radar module, designed to emulate the radar employed by Saferadar. Consequently, the trained network is tailored specifically to this radar's characteristics and may not generalize well to data from radars with significantly different specifications; the base case is however, to be established.

Although a DL solution could be trained to track a variety of different objects, such as pedestrians or two-wheelers, this work sets out to track vehicles that are predefined in CARLA using only deterministic driving models. In addition only one CARLA map is used for generating training and validation data.

2

Theory

2.1 Radar

Radar detection technology is based on echo-localization and uses radio waves to detect objects in an area that the radio waves span. The waves are emitted from a transmitter, which travel through space until they hit an object which in turn reflects the wave back to the radars receiver. The waves that are returned are then measured in terms of amplitude and phase to determine amongst other, the position and velocity of the hit object [10]. A major advantage of using radar apart from other modalities such as lidar and camera is that the radar can yield velocity measurements from the targets hit.

The measurements yielded by a radar is a point cloud that is the union of the set of clutter measurements and the set of readings from real targets. This can be expressed as $\mathbf{z}_t = \mathbf{m}_t \cup \mathbf{c}_t$ where $\mathbf{z}_t$ denotes the total amount of measurements in a time step $\mathbf{m}_t$ the measurements from objects and c_t the clutter measurements [11]. Clutter readings are unpredictable and can cause more or less problems when interpreting point clouds depending on the intricacy of the task at hand. When dealing with radars and radar technology, statistics plays a significant role. Interpretation of the state space by using radar technology naturally deals with uncertainty from noise since disturbances and random events are interfering with the radio waves that the radar emits and later receives.

Since the technology has been around for a long time, extensive research has been conducted in the field and has resulted in different radar techniques and architectures. This is to tailor the radars behaviour for the specific application yielding the desired measuring behaviour. One of the key difference of a radar is if the radar waves are pulse based or continuous [10].

2.1.1 FMCW-radar

The radar that is intended to be of interest in this work is the Frequency-Modulated Continuous-Wave (FMCW) radar. It is commonly used and has been proven effective in the automotive industry [12], [13] and in DL applications [14] and builds on the continuous wave technique where the radar emits a continuous wave and registers the points from where the radar gets a signal back and their frequencies. In order to separate the different measurements over time, the frequencies are modulated over time in chirps that increases the frequency over a period of time and then start over after a set time to begin the next chirp. By receiving these signals the radar can estimate the radial distance,

angle, radial velocity and signal strength of the objects from where it receives detections. Other implicit measurements used to gain information about the measurements are the signal to noise ratio (SNR) and radar cross section (RCS).

As in all radars, noise in the measurements collected with a FMCW-radar is common. Velocity measurements from a vehicle can for instance vary despite that the entire vehicle and its body moves at the exact same speed. Two measurements, that came from the exact same spot on a vehicle moving at constant speed, might give different velocity readings. The same goes for other measurements including distance and especially angle measurements.

2.2 Radar tracking

Object tracking in radar applications includes estimating the state space in the target area by evaluating point clouds from measurements that is yielded from a radar unit. The tracking is often derived from data association between the points in space and time and the similarities or dissimilarities between them. Doing this over time lets the data association work recurrently between the time steps and weights the current information with the information gained from the next time step to set up a hypothesis of the current state space. Since the data yielded from radar units often can be noisy, statistical models are often used to incorporate the uncertainty in the model.

When tracking objects in radar applications a prediction step is usually included. Predicting includes estimating the position and kinematic state, and in the case of tracking extended objects, also the extent. This gives information about where the objects position is but also the motion including the velocity, acceleration and turn-rate. This in turn gives information of where the objects position in the next time step by extrapolating the state and then comparing it with the actual information given from the measurements in the point cloud.

The ability to keep tracks is directly dependant on the data that the tracker receives to base its estimates on. A differentiation between data containing high and low amounts of information can be made to define the challenges that comes with it. With state representations with data that contains low information such as the radar, one of the main challenges are to make correct assumptions of which measurements in the point cloud are objects of interest and which are clutter and do robust tracks on those data points. This makes tracking with radar measurements, especially in cluttered and noisy environments, a rather difficult task and different methods have been developed to resolve these issues [15], [16]. The ideal environments to use radars for tracking purposes can therefore be argued to be low clutter environments such as the sea and the sky. Since there is a probability to detect the objects of interest as well as other objects that are labeled clutter, the distinguishability between objects and clutter can become very hard if they share similar traits.

Another challenge on working with tracking on radar data is that the radar yields probabilistic readings, which builds on statistics to account for the measurement noise, clutter and occlusions. A usual method to handle the low information or uncertain data is to use Kalman filters and are present in most model based trackers [17]. The Kalman filters however are also probabilistic which means that the output yielded is yet another representation of the data that corresponds more or less well to the true state.

2.2.1 MOT

Multi object tracking, in contrary to single object tracking, is the case when the task is to track several objects in each time step. The case of MOT can be seen as the more challenging one compared to single object tracking since the measurements have the possibility of being associated to different objects and the uncertainty that it entails.

Multi object tracking entails added complexity since data association becomes an increasingly hard task compared to the single object case. Multiple objects moving in the state space yields points that can be more or less noisy and the measurements that are deemed to stem from objects i.e. not clutter needs to be attributed to an object in order to make an estimation of the current state. This is often a difficult task in scenarios where objects tend to be moving close to each other since there are issues attributing a measurement to the correct objects.

Another difficulty when working with MOT is that different objects may have their own unique motion model. This becomes a problem when trying to extrapolate between the time steps to identify object based on their movement. Combining multiple unique behaviors with a dynamic velocity profile makes the task at hand hard to tackle in order to get a good and robust tracking performance.

2.2.2 EOT

Another important aspect that effects the tracking task is if the targets are modelled as points or extended objects. When dealing with extended object tracking the data association becomes increasingly difficult since objects that span an area or volume can yield multiple measurements in a single time step. Thus, point object tracking is often seen as a simplification of the problem making the assumption that each object in the state space only can produce one measurement per time step. Most cases of radar tracking in real life when in short range environments such as urban environments are considered to be extended and is therefore a highly relevant topic [18].

Tracking extended objects is interesting when the radar receives points from objects that are likely to produce more than a single point in each time step. When dealing with EOT the state estimation goes from estimating the centroid and its kinematic state to also include the extent, which describes the shape of the object. Since there is the possibility of receiving multiple points from one object in EOT, it leads to a problem in determining which measurements that come from that object and how to attribute them. This leads to new challenges as to associating the data points received to one another and to make an accurate estimation. When also dealing with a case containing MOT, this becomes one of the main problems since the uncertainties naturally are very high.

Tracking multiple objects which are extended at the same time, becoming a multi extended object tracking (MEOT) task, can be seen as a more complex and more realistic task since the number of objects over the course of time is varying and may also be occluded in parts of the duration. The uncertainty of the measurements are high in the sense that multiple measurements can be associated to one same object while still needing to be aware of object birth. This gives rise to the need of trackers that can track in highly uncertain environments [18].

2.2.3 Tracking performance metrics

To evaluate how well a tracker manages to estimate the state space in a time step, a way to measure performance is needed. The most interesting metric to evaluate is in first hand accuracy, in terms of position. But metrics concerning also velocity or cardinality of the estimations are also of interest. Other metrics that may be interesting is how well the tracker can distinguish the identities of tracks coming from different objects if doing labeled MOT, discern readings generated by objects from clutter that does not stem from objects that are of interest or how the extent of the estimation correlates to the true extent of objects if dealing with EOT. Other performance metrics that does not concern the tracks directly could be to measure the inference times and the scalability of the tracker when increasing complexity of the data from the radar.

2.2.4 GOSPA

The Generalized Optimal Subpattern Assignment (GOSPA) metric is a performance measurement for multi object tracking systems that evaluates trackeraccuracy. The metric was introduced in [19] building on the optimal subpattern assignment metric, with the basic principles involving computing the cost of associating estimated tracks with ground truth tracks, considering both localization errors and cardinality differences. Mathematically, it can be expressed as

$$d_p^{c,2}(X,Y) = \left[\min_{\gamma \in \Gamma} \left(\sum_{(i,j)\in\gamma} d(x_i, y_j)^p + \frac{c^p}{2}(|X| + |Y| - 2|\gamma|) \right) \right]^{\frac{1}{p}} \tag{2.1}$$

Where X is the set of objects present and Y the set of predictions made by the tracker. The equation can be divided into two different terms, one that penalizes bad estimates that measures the offset between prediction and current state the other term is the cardinality error that penalizes if the tracker predicts too many or too few objects present in the state space, this encourages a balance between track purity and completeness.

The GOSPA metric is defined by a set of parameters including the localization error threshold, the cardinality error threshold, and a decay parameter that controls the impact of distance on the assignment cost. By varying these parameters, GOSPA can be tailored to different tracking scenarios and application requirements. Evaluating tracking performance using GOSPA provides insights into the overall accuracy and robustness of the tracking system, allowing for comparisons between different algorithms and configurations.

2.2.5 GOSPA for extended targets

When dealing with targets that have an extent, both the orientation of the objects and their extent becomes factors to be accounted for in order to rank how well the tracks approximate the objects true state. To account for these factors the original GOSPA metric can be modified by adding a Gaussian Wasserstein distance as the localization error in place of the sole distance between the points that represents the targets. This is shown to be effective for evaluating elliptical representations of shapes [20]. The distance is measured as

$$d_{GWD}(\mu_{1,2}, \Sigma_{1,2}) = \sqrt{\|\mu_1 - \mu_2\|_2^2 + \mathrm{Tr}(\Sigma_1 + \Sigma_2 - 2(\Sigma_1^{1/2}\Sigma_2\Sigma_1^{1/2})^{1/2})} \tag{2.2}$$

Where μ is denoting the centroid and Σ is denoting the covariance matrix of the shape.

2.3 Conventional approach to radar tracking

The conventional approach when tracking objects from a feed of radar data in a model based setting is to use filters that takes the data and try to create a relationship between the points in the cloud at each time step to estimate the true state. The filters are recursively updating the estimates using Bayes' rule, which combines prior knowledge with the newly gained information fed from the radar in each time step. The tracking behaviour depends on a set of parameters specific for the filter used. The parameters are tuned for the filter to give desired tracking behaviour depending on the data yielded from the radar to start and continue a track over time. Tracks are started and continued by evaluating the correlation between the set of readings between time steps and giving each track an existence probability. The filtering procedure can therefore be seen as having a current state and an updated state, where the updated state in the next step is dependant on the current state from the previous time step [21]. This can be expressed as

$$p(x_k|Z_k) = \frac{p(x_k|z_k, Z_{k-1})p(z_k|Z_{k-1})}{p(z_k|Z_{k-1})} = \frac{p(z_k|x_k)p(x_k|Z_{k-1})}{p(z_k|Z_{k-1})} \tag{2.3}$$

where Z_k denotes the available measurements, x_k the state vector and z_k the newly gained measurements at time k. Because of the recurrency of the state estimation x_k with a posterior and an update, it can be expressed as a function of Z_k between time steps and can be seen as a Markovian chain where the gained information from the measurements increase or decrease the probability of existence from a track. The state at time x_k can therefore be obtained by using the Chapman-Kolmogorov equation [21].

$$p(x_k|Z_{k-1}) = \int p(z_k|x_{k-1})p(x_{k-1}|Z_{k-1})dx_k \tag{2.4}$$

Since the filtering uses random finite sets to work with the probabilistic data association, some kind of heuristics are often included to reduce the computational load and the accuracy of the estimations. A possible approach to use weights and gating that keeps the hypothesis of the closest detections. The heuristics becomes increasingly important when dealing with tracking filters using multiple hypothesis in the tracking procedure.

2.3.1 PMBM

The Poisson Multi-Bernoulli Mixture filter was first introduced in [22] and has since shown good performance in both MOT and EOT scenarios [23],[24],[25]. The main idea of the PMBM filter is to combine a Multi-Bernoulli mixture (MBM) filter and a Poisson Point Process (PPP).

The MBM filter keeps track of the detected objects updating likelihoods and hypotheses depending on the measurements received. The PPP models the undetected objects missed in the measurements. Thus the addition of a PPP prior helps with the birthing of objects that are as of yet, occluded. This results in a joint probability density function

$$p(\lambda, x) = \mathcal{P}(\lambda) \cdot \mathcal{MBM}(x; m, P) \tag{2.5}$$

By combining these models, the PMBM filter uses the PPP and the MBM to create a tracking behaviour particularly helpful in uncertain environments. The MBM filter then updates the existence likelihood of detected targets, represented by a mixture of hypotheses [22]. The kept state is then predicted by incorporating a motion model that predicts the offset of the state between time steps and are then compared to the new measurements fed to the filter as described in Eq. 2.3. The overall functionality is illustrated below highlighting the primary components of the PMBM filter.

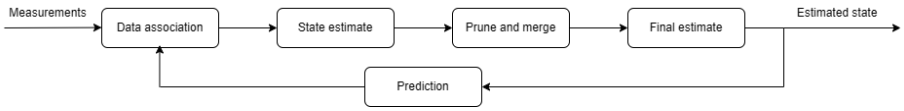


Figure 1: Simplified procedure of a PMBM filter.

The PMBM filter is then tuned depending on the data fed to the filter and the individual task to best represent the true state space for accurate tracking behaviour. Further explanation and derivation of the PMBM filter can be read in [22].

PPP

A Poisson Point Process (PPP) is a stochastic model that describes the distribution of random points. The process uses a Poisson distribution to model the points over a given area that is independent of earlier outcomes. The model receives the randomly birthed points according to λ depending on the area of possible estimations over a Gaussian distribution with mean m and covariance P

$$p(\lambda) = \mathcal{P}(\lambda) \cdot (\mathcal{G}(x; m, P)) \tag{2.6}$$

Since the PPP model is independent of earlier outcomes, it serves as a good tool to model randomness and unpredictable events.

MBM filter

The Multi-Bernoulli Mixture (MBM) filter associates data given to the filter by building cost matrices [26]. From the cost matrix, the filter generates tracks in form of independent Bernoulli components which have weights and state distributions where each component can be seen as a potential track. The Bernoulli components each have an existence probability and a weight as a kind of confidence measurement in the components. The components hypothesised states are summed with regards to the weight assigned to make a global hypothesis that results in the track. If the components have an existence probability below a set pruning threshold, the component becomes removed in the pruning process and discontinued in the tracking process. The tracks that are kept in the pruning process are then extrapolated by incorporating a motion model that the tracks are expected to move accordingly to. Instead of just using one hypothesis for each track, a mixture of multiple hypotheses are kept with weights tied to them and evaluated in each step depending on the measurements given in the next step.

$$p(x) = \sum_i w_i \prod_j (r_{i,j}, p(x)) \tag{2.7}$$

Where $\sum_i$ is the sum of the i different hypotheses connected to a track and $\prod_j$ a single hypothesis and w_i the weight connected to it. $r_{i,j}$ denotes the existence probabilities of a component and $p_{i,j}(x)$ the state distribution. Further derivation can be found in [26].

2.3.2 GGIW-PMBM

When wanting to perform extended object tracking with the PMBM filter, it is of interest to model the predicted extent of the objects. One way to do this is to use the gamma Gaussian inverse Wishart (GGIW) model to represent the object state [27]. This makes the filter apart from only tracking the center point of the objects also estimate how the object spans the room. The general tracking behaviour is alike the case of using a PMBM filter for POT by incorporating motion and measurement models to estimate the state but with the addition of the GGIW distributions. A novel challenge however is the fact that an object can give more than one reading per time step which gives rise to the association problem as introduced in 2.2.2. To address this, the filter introduces merging that is the result of different Bernoulli components becoming merged together if they meet a set criteria since they are considered to stem from the same object. This is seen to have been effective and a common methodology when working with extended objects [28],[29]. The PMBM filter then updates the state in a similar fashion as described in figure 1, where the weights of the Bernoulli components depending on how well the GGIW state corresponds to the measurements given. This results in a model based state space estimation that updates with each time step depending on the current estimated state, the prediction depending on the motion model and the new data from the next time step.

GGIW

The GGIW model uses the gamma distribution, Gaussian distribution and the inverse Wishart distribution to model the properties of the extended target and yields the scale, centroid and shape that is used in the estimation. A more thorough explanation of the model can be found in [30].

$$p(\lambda, \mathbf{x}, X) = \mathcal{G}(\lambda; a, b) \cdot \mathcal{G}(\mathbf{x}; \mathbf{m}, P) \cdot \mathcal{IW}(X; V, \nu) \tag{2.8}$$

Where λ, x and X is given by the different distributions.

The gamma distribution deals with measurement rate and scale of the tracked object, yielding the measurement rate denoted as λ.

$$\mathcal{G}(\lambda; a, b) \tag{2.9}$$

Where a denotes the shape parameter and b the rate parameter which both are connected to the size of the tracks and how often the object should have produced readings. The shape parameter is connected to the variance of the expected size of the tracked objects with a low value resulting a bigger variance of size or uncertainty from readings and the rate parameter effecting how the size from the initial track is changing over time depending on the amount of measurements received close to the track.

The Gaussian distribution deals with the kinematic state to attribute the track a position and velocity which are given as the variable x.

$$\mathcal{G}(\mathbf{x}; \mathbf{m}, P) \tag{2.10}$$

With m denoting the state vector which holds information about the position and velocity of the centroid and P the covariance matrix that are connected to the uncertainties of the state vector estimates.

The inverse Wishart distribution models the shape, size and orientation of the object by creating an elliptic shape as the track. From the distribution one receives a covariance

matrix X.

$$\mathcal{IW}(X; V, \nu) \tag{2.11}$$

Where V is the scale matrix and ν the degrees of freedom. The scale matrix controls the expected shape and extent of the tracked objects and the variance or uncertainties in the different directions of the extent. The degrees of freedom can be seen a level of confidence of the scale matrix where a high value correlates to a high confidence and thus will concentrate the distribution around the scale matrix.

2.4 Transformer network

The transformer network (TN) architecture was originally proposed in [6], submitted by Google, which aimed to improve sequence-to-sequence data handling. A key innovation introduced is the self-attention mechanism which allows the model to weigh the importance of different elements within a sequence, enabling it to attend to different parts of the input for each output. This ultimately helps the network capture long-range dependencies and contextual relationships within the sequence.

The architecture includes two modules called the encoder and decoder stacks, composed of multiple layers. Within each layer, the Transformer utilizes self-attention mechanisms to capture contextual relationships between words in the input sequence. Multi-head attention involves running multiple self-attention operations in parallel, each with different learned parameters. This allows the model to attend to information from various representation subspaces simultaneously, enhancing its focus on multiple aspects of the input data.

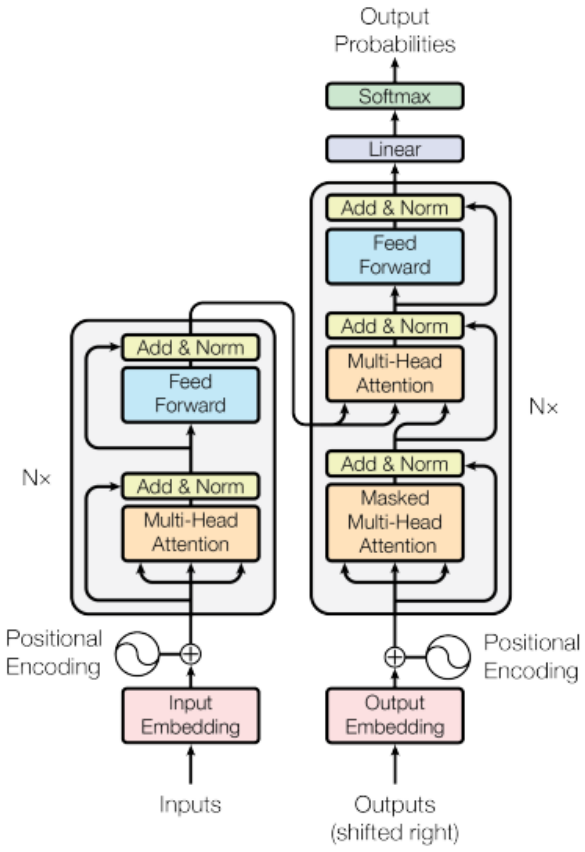


Figure 2: The transformer network as proposed by [6].

Input Embedding and Positional Encoding

The initial step is converting the input sequence, typically a sequence of tokens, into a continuous vector space. This transformation is achieved through an embedding layer that maps each token in the sequence to a corresponding dense vector. This can be achieved through a Feed Forward Network (FFN) or a learned positional encoder. Given an input sequence of tokens $\mathbf{z}_{1:n}$ where n denotes the number of tokens, the embedding layer projects each token into a d_m-dimensional space:

$$\mathbf{z}_{1:n} \in \mathbb{R}^n \rightarrow \mathbf{e}_{1:n} \in \mathbb{R}^{n \times d_m} \tag{2.12}$$

Here, $\mathbf{e}_{1:n}$ represents the embedded sequence, and d_m is the dimensionality of the embedding space, which corresponds to the size of the output from the embedding layer.

Since the transformer architecture does not inherently capture the sequential nature of the data, a positional encoding is added to the embedded tokens to provide information about the relative positions of tokens in the sequence. The positional encoding can be a deterministic function of the position index i and the dimension j of the embedding vector, as defined by:

$$PE(i, 2j) = \sin\left(\frac{i}{10000^{2j/d_m}}\right), \quad PE(i, 2j+1) = \cos\left(\frac{i}{10000^{2j/d_m}}\right) \tag{2.13}$$

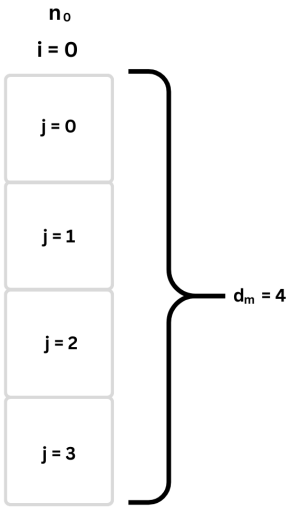


Figure 3: Illjustrating positional encoding of Eq. 2.13.

This positional encoding is then added to the token embeddings:

$$\mathbf{e}_{1:n} = \mathbf{embedding} + PE(i) \tag{2.14}$$

where $\mathbf{e}_{1:n} \in \mathbb{R}^{n \times d_m}$ carries both the content of each token and its position in the sequence.

Self-attention mechanism

The heart of the transformer model is the self-attention mechanism, which allows the model to weigh the importance of each token relative to every other token in the sequence. This mechanism enables the model to focus on different parts of the input sequence regardless of their positional distance.

Given the embedded sequence $\mathbf{E}_{\text{pos}}$, the self-attention mechanism begins by projecting the embeddings into three distinct matrices: Query ($\mathbf{Q}$), Key ($\mathbf{K}$), and Value ($\mathbf{V}$). These projections are achieved through learnable weight matrices $\mathbf{W}_Q$, $\mathbf{W}_K$, and $\mathbf{W}_V$, each of dimension $d_m \times d_m$:

$$\mathbf{Q} = \mathbf{e}\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{e}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{e}\mathbf{W}_V \tag{2.15}$$

Here, $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ each have the shape $n \times d_m$, corresponding to the number of tokens n and the embedding dimensionality d_m.

In multi-head attention blocks, each of these matrices is divided into h heads, splitting the embedding dimension d_m into $d_k = \frac{d_m}{h}$ (k denoting keys in the original paper):

$$\mathbf{Q}^{(i)},\ \mathbf{K}^{(i)},\ \mathbf{V}^{(i)} \in \mathbb{R}^{n \times d_k} \quad \text{for } i = 1, \ldots, h \tag{2.16}$$

The self-attention mechanism calculates the attention scores by taking the dot product of the Query and Key matrices for each head. These scores determine how much focus each token should receive relative to others:

$$A^{(i)} = \frac{\mathbf{Q}^{(i)}\mathbf{K}^{(i)T}}{\sqrt{d_k}} \quad \text{for } i = 1, \ldots, h \tag{2.17}$$

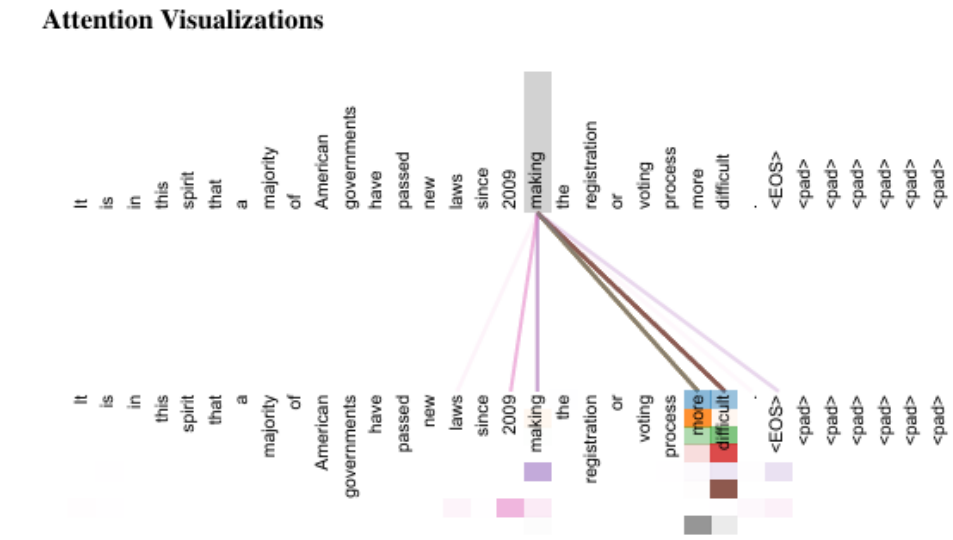


Figure 4: Self attention visualized for a single head, illustrating its learned attention scores between words. Taken from [6].

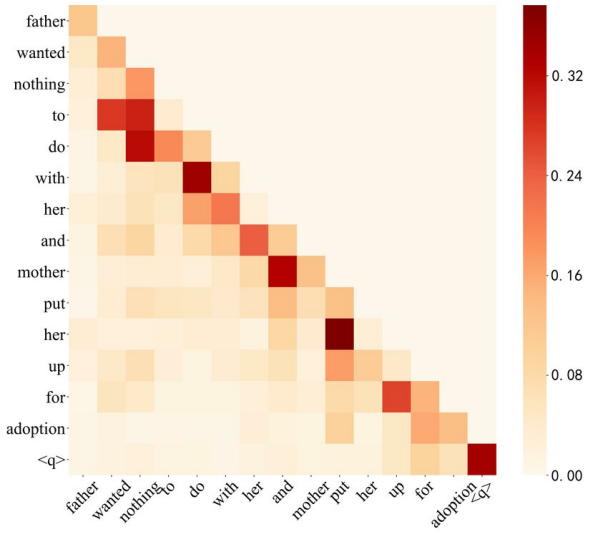


Figure 5: Masked self attention visualized as a heat-map for a single head, illustrating its learned attention scores of words in sequence. Taken from [31].

To ensure numerical stability and prevent excessively large values, the dot products are scaled by $\sqrt{d_k}$. The scaled scores are then passed through a softmax function to produce the attention weights, which normalize the scores across each token:

$$\mathbf{O}^{(i)} = \text{softmax}\left(A^{(i)}\right)\mathbf{V}^{(i)} \tag{2.18}$$

The output of the self-attention mechanism for each head is a weighted sum of the Value vectors, where the weights are determined by the attention scores.

Masking

Masking is an efficient way of training the the self attention weights on words sequentially. Attention between words must never be propagated backwards so that only previous words may influence future words in a sentence. This effectively lets the model train more per sentence as it needs to learn how words propagate forwards in a sentence (or rather in sequence). In simpler words it lets the model predict what token by only knowing came previously in training, here illustrated in Fig. 5.

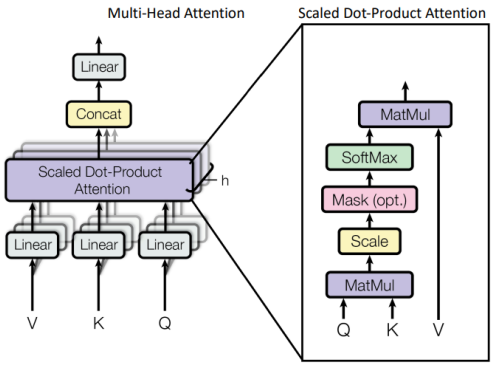


Figure 6: Detailed self attention mechanism module from [6].

Multi-Head Attention and Output

The outputs from each attention head are concatenated and then projected back to the original embedding space d_m:

$$\mathbf{O} = \text{Concat}(\mathbf{O}^{(1)}, \mathbf{O}^{(2)}, \ldots, \mathbf{O}^{(h)})\mathbf{W}_O \tag{2.19}$$

where $\mathbf{W}_O \in \mathbb{R}^{hd_k \times d_m}$ is a learnable weight matrix used to project the concatenated outputs back to the embedding space.

The resulting matrix $\mathbf{O} \in \mathbb{R}^{n \times d_m}$ contains the final representations of the input sequence after applying self-attention, capturing the dependencies and relationships between tokens in a way that accounts for both content and position.

The transformer architecture, through its embedding process, positional encoding, and self-attention mechanism, allows for highly flexible and powerful modeling of sequential data. Through multiple attention heads and parallel processing, transformers can efficiently capture complex dependencies and relationships in data, making them particularly effective for a wide range of tasks in natural language processing, computer vision, and beyond.

Cross-Attention in Transformers

Cross-attention is a variant of the attention mechanism used in transformer architectures, particularly in tasks that involve interactions between two different sequences, such as in sequence-to-sequence models (e.g., machine translation) or in multimodal applications (e.g., combining text and image data). While self-attention operates on a single sequence, allowing the model to focus on different parts of that sequence, cross-attention enables the model to attend to a different sequence entirely, making it crucial for tasks that require the integration of information from multiple sources.

Cross-attention operates similarly to self-attention, but with a key difference in the sequences being attended to. The source Sequence (or "Memory") is the encoder output which will be computed into Key and Value while the Target Sequence is what Queries the source sequence. Note here that the target sequence is an output from the decoder self-attention module as shown in Fig. 2.

The operation is the same as in regular self-attention but the sequences can be of different lengths. The target sequence $\mathbf{Y} \in \mathbb{R}^{m \times d_m}$ is linearly projected to produce the Query matrix $\mathbf{Q}$:

$$\mathbf{Q} = \mathbf{Y}\mathbf{W}_Q, \quad \text{where} \quad \mathbf{Q} \in \mathbb{R}^{m \times d_k} \tag{2.20}$$

The source sequence $\mathbf{X} \in \mathbb{R}^{n \times d_m}$ is linearly projected to produce the Key ($\mathbf{K}$) and Value ($\mathbf{V}$) matrices:

$$\mathbf{K} = \mathbf{X}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}_V \quad \text{where} \quad \mathbf{K}, \mathbf{V} \in \mathbb{R}^{n \times d_k} \tag{2.21}$$

Here, $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d_m \times d_m}$ are learnable weight matrices, and d_k is the dimension of the projected space d_m divided by number of heads as done in Eq. 2.16.

The attention scores are computed by taking the dot product of the Query matrix $\mathbf{Q}$ with the transposed Key matrix $\mathbf{K}^T$:

$$\mathbf{A} = \frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} \quad \text{where} \quad \mathbf{A} \in \mathbb{R}^{m \times n} \tag{2.22}$$

The rest follows the same process as presented in Eq. 2.17 through 2.19.

Layer Normalization

Normalization, or batch-normalization, is a function often applied in deep learning, re-centering and re-scaling inter-layer outputs to 0 ± 1 to avoid gradient explosions [32].

In the transformer architecture, layer normalization is applied at multiple points, helping to stabilize the learning process by ensuring that the inputs for each layer are normalized, reducing the internal covariate shift. Similarly, the input to the self-attention mechanism is often normalized as well, which helps in dealing with varying scales of input tokens, ensuring that the attention mechanism operates more effectively. In both cases, layer normalization contributes to making the transformer model more robust, allowing it to handle the complex dependencies and relationships inherent in sequential data.

Given an input vector $\mathbf{x} \in \mathbb{R}^d$ from a specific layer in the network, layer normalization computes the normalized output $\hat{\mathbf{x}}$ by first calculating the mean μ and variance σ^2 across the elements of $\mathbf{x}$:

$$\mu = \frac{1}{d}\sum_{i=1}^{d} x_i \tag{2.23}$$

$$\sigma^2 = \frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2 \tag{2.24}$$

Where:

x_i is the i-th component of the input vector $\mathbf{x}$. d is the dimensionality of the input vector. The input vector $\mathbf{x}$ is then normalized as follows:

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{2.25}$$

Here:

ϵ is a small constant added for numerical stability, preventing division by zero. After normalization, the output is scaled and shifted using learned parameters γ and β:

$$\mathbf{y}_i = \gamma\hat{x}_i + \beta \tag{2.26}$$

The final output vector $\mathbf{y}$ has the same dimension as the input vector $\mathbf{x}$ but is now normalized, with the option to be scaled and shifted by the learned parameters. This process ensures that the distribution of activations remains consistent, helping the model to converge faster and perform more effectively.

Softmax Function

The softmax function is a fundamental component in many ML applications; converting layer outputs a normalized probability distribution with the sum of 1. Inherent to the function is that the softmax function amplifies the differences between high and low scores through exponentiation. This helps the model to focus more sharply on the most relevant tokens while diminishing the influence of less relevant ones.

In the transformers self-attention mechanism softmax it is used to convert raw attention scores into a probability distribution. This allows the model to weigh the importance of different tokens in the sequence relative to one another, see figures 4 and 5.

For each element $a_{ij}^{(h)}$ in the attention score matrix $\mathbf{A}^{(h)}$ for head h, the softmax function computes the corresponding attention weight $w_{ij}^{(h)}$ as:

$$w_{ij}^{(h)} = \frac{\exp(a_{ij}^{(h)})}{\sum_{k=1}^{n}\exp(a_{ik}^{(h)})} \tag{2.27}$$

where $\sum_{k=1}^{n} \exp(a_{ik}^{(h)})$ is the sum of the exponentials of all the scores in the i-th row of $\mathbf{A}^{(h)}$, ensuring that the resulting weights sum to 1.

The resulting attention weights $\mathbf{W}_{\text{att}}^{(i)}$ are then used to compute the weighted sum of the value vectors $\mathbf{V}^{(i)}$:

$$\mathbf{O}^{(i)} = \mathbf{W}_{\text{att}}^{(i)} \mathbf{V}^{(i)} \tag{2.28}$$

The DEtection TRansformer (DETR) [9] was introduced as a new approach to object detection by formulating it as a direct set prediction task, differing from the traditional ordered multi-stage pipelines. In conventional object detection methods such as the Regional-based Convolutional Neural Network (CNN), the process involves generating a large number of candidate bounding boxes (proposals), classifying each proposal, and then refining the results through techniques like Non-Maximum Suppression to eliminate duplicates and false positives [33]. The ordered nature of traditional object detection models is characterized by this kind of sequential processing and the separate training of individual components, each of which depends on the output of the previous stage, see Figure 8. This ordered structure increases the complexity of the architecture, as it requires tuning and management of multiple interdependent stages. Each stage operates on a sequence of inputs and outputs that must be managed in a specific order, making the entire pipeline more complex and sensitive to the ordering of operations. This traditional approach, while effective, introduces complexity due to the reliance on hand-crafted components and post-processing heuristics.

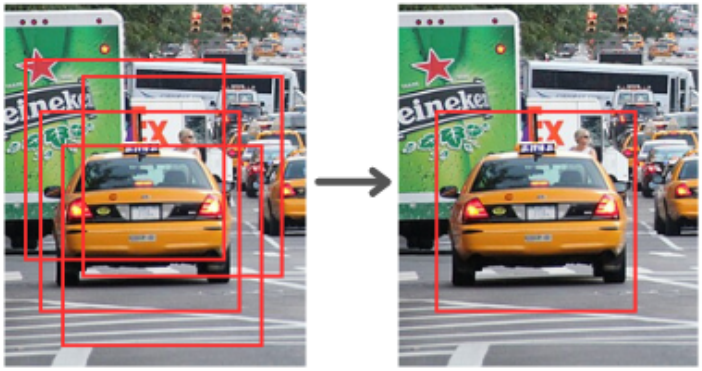


Figure 7: Illustration of non-maximum suppression used in object detection and classification.

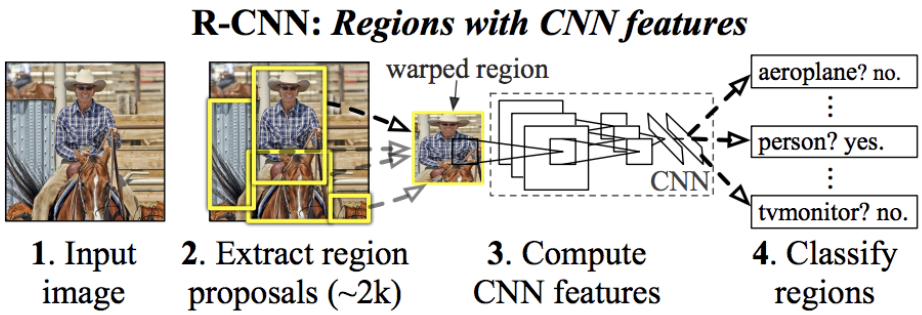


Figure 8: Illustration of an R-CNN pipeline

Furthermore by handling object detection as a direct set prediction problem in an end-to-end manner it diverges from the autoregressive model presented by [6] that predicts the output sequence one element at a time (word by word, beginning with a <START> token and ending with <END>). The model outputs a fixed number of predictions N_q, of which the order does not affect the final results. During training, a bipartite matching algorithm, specifically the Hungarian algorithm, is employed for a one-to-one matching between prediction and ground-truth object in training. This matching is what allows for the model to be permutation-invariant, avoiding duplicate predictions and accurately learn object representations, reflecting the set-based nature of the task.

In the end, transformers allow for capturing global context across the entire image, leading to highly accurate predictions with a simplified and more unified pipeline. However, DETR's training requires large datasets and longer convergence times compared to traditional detectors, but its end-to-end trainable nature and simplicity make it a powerful approach in modern object detection tasks.

2.4.1 DETR

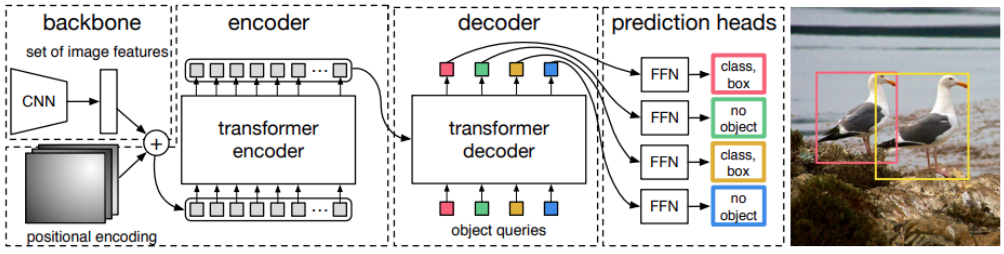


Figure 9: DETR network architecture taken from [9]

The process begins with a CNN, typically a ResNet, which processes an input image $\mathbf{I} \in \mathbb{R}^{H \times W \times 3}$ to generate a lower resolution activation map $\mathbf{F}_a \in \mathbb{R}^{H \times W \times C}$, where $H = \frac{H_0}{32}$ and $W = \frac{W_0}{32}$ are the spatial dimensions, and $C = 2048$ is the channel dimension. The extracted feature map is further processed and then flattened into a into a sequence of vectors $z_0 =\in \mathbb{R}^{hw \times d}$, suitable for processing by the transformer, resulting in a sequence

of length hw with element vectors of d. A standard positional encoding, described in [6], is added as the final prepossessing step.

Each encoder layer then refines the sequence of feature embeddings by allowing each element of the sequence to attend to all others (self-attention), capturing global dependencies and context within the image. The output of the final encoder head is then passed to the decoder which receives a fixed set of learned embeddings, called object queries $\mathbf{Q} \in \mathbb{R}^{N_q \times d}$, where N_q represents the maximum number of objects (eg. $N_q = 100$) the model can predict.

These queries interact with the encoder's output embeddings through the cross-attention mechanism, producing a set of output embeddings $\mathbf{O} \in \mathbb{R}^{N_q \times d}$, where each embedding represents a potential detected object. Each one of these embeddings are passed to a shared prediction head (FFN) which will predict a detection with class and bounding box or a "no-object" class.

DETR Loss functions

DETR uses a bipartite matching loss to ensure a one-to-one correspondence between the predicted objects and the ground truth annotations. Matching is done using the Hungarian algorithm, which finds the optimal assignment of predictions to ground truth objects.

The matching cost between a predicted object and a ground truth object is a combination of the class prediction loss (cross-entropy) and the bounding box regression loss (a combination of ℓ_1 loss and the generalized IoU loss):

$$\text{Cost} = \lambda_{class} \cdot \text{CrossEntropy}(\hat{\mathbf{c}}, \mathbf{c}) + \lambda_{bbox} \cdot \ell_1(\hat{\mathbf{b}}, \mathbf{b}) + \lambda_{giou} \cdot \text{GIoU}(\hat{\mathbf{b}}, \mathbf{b}) \tag{2.29}$$

where $\hat{\mathbf{c}}$ and $\hat{\mathbf{b}}$ are the predicted class and bounding box, respectively. $\mathbf{c}$ and $\mathbf{b}$ are the ground truth class and bounding box, respectively. λ_{class}, λ_{bbox}, and λ_{giou} are hyperparameters that balance the contributions of each loss.

2.4.2 Bounding box Loss

An ordinary Complete Intersection Over Union loss (CIoU) works with four coordinate points and calculates two additional losses besides the IoU and is expressed as follows:

$$\mathbf{L}_{CIoU} = 1 - IoU + \frac{d^2}{C^2} + av \tag{2.30}$$

$$v- = \frac{4}{\pi^2}(arctan\frac{w^{gt}}{h^{gt}} - arctan\frac{w}{h})^2 \tag{2.31}$$

$$\alpha = \frac{v}{(1 - IoU) + v} \tag{2.32}$$

where the fraction $\frac{d^2}{C^2}$ calculates the center offset and av calculates the aspect ratio discrepancy.

This loss function is extension of the generalized IoU loss function used in the original DETR.

2.5 MT3v2 Architecture

The MultiTarget Tracking Transformer v2 network introduced by [34] and later refined in [5] is a transformer network designed to predict states from a set of radar readings. MT3v2 was developed to compete with SOTA radar tracking methods and has demonstrated competitive performance in testing. The architecture uses a standard encoder, modified decoder and a selection mechanism employed to enhance training speed and performance, partly inspired by the deformable DETR [35]. The loss functions are employed at multiple stages such that the model iteratively trains on intermediary object queries at each decoder block. Additionally a contrastive auxiliary loss function is used to train the embedding outputs as well.

Subsequent sections give an account for the MT3v2 network processes used in this research. Most components remain unchanged with modification made mainly to replace Multi-Bernoulli densities with bounding box prediction and adapt the networks ability to work with measurements from CARLA.

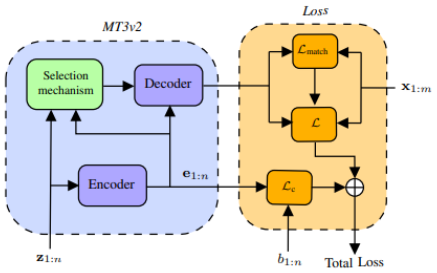


Figure 10: MT3v2 architecture taken from the paper [5].

2.5.1 Input data

The network uses radar measurements which include range (r), radial velocity (v_r), angle (ϕ) and time-step (t). For each time-step the radar may detect a varying amount of measurements n_t. These measurements are concatenated, to form a time-step block with measurements. Lastly, several time-step blocks are concatenated in temporal order and fed to the network as a sequence $\mathbf{x_{1:t}} \in \mathbb{R}^{m \times N}$ where N is the total number of measurements and m is the measurement dimension.

2.5.2 Preprocessing measurements

The measurements are encoded both in the domain of time and space to align the measurements with the dimensional requirements of the transformer model and ensure consistency across varying scales and to facilitate subsequent transformations by the network.

A learned positional encoder (similar to [6] which uses sinusoidal components) is applied to the temporal sequence in the measurements, projecting each time stamp into a high dimensional vector and in turn transforming the sequence into the matrix $\mathbf{q_{1:n}} \in \mathbb{R}^{d_{model} \times N}$. The spatial components of the measurements are similarily projected to a matrix $\mathbf{e_{1:n}} \in \mathbb{R}^{d_{model} \times N}$ using a FFN.

2.5.3 MT3v2 Encoder

The MT3v2 network employs a standard encoder architecture with a self attention module and several linear and normalization layers. The aforementioned encodings are passed to the self-attention module as query, key and value where

$$query = key = \mathbf{q}_{1:n} + \mathbf{e}_{1:n}$$
$$value = \mathbf{e}_{1:n}$$

After the self-attention module the embeddings are passed through several layers of normalization and linears as illustrated by figure 11. Note that mask padding is applied with different sized sequences batch inputs.

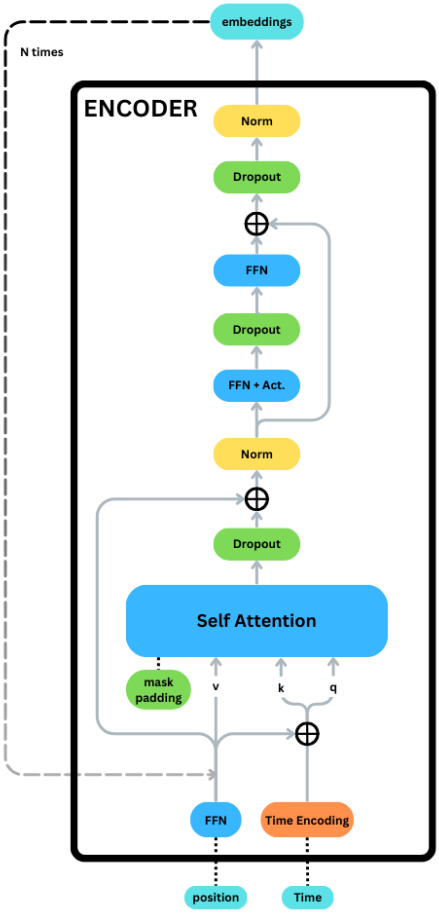


Figure 11: MT3v2 encoder illustrated in detail.

2.5.4 MT3v2 Selection Mechanism

The selection mechanism employed in the MT3v2 network takes inspiration from the two-stage deformable DETR. Where the original DETR uses learned parameters to

produce proposed object queries, the two-stage deformable DETR uses the encoder embeddings as input to a FFN when proposing the initial object queries.

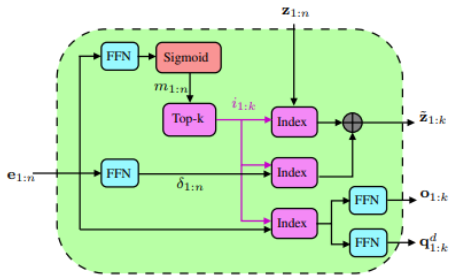


Figure 12: Selection mechanism architecture taken from [5].

The MT3v2 decoder uses two prediction heads for this process; the Object Classifier which ranks all measurements and scores them according to the probability of containing an alive object and the Position and Velocity Predictor which outputs the position and velocity for all object queries.

The selection mechanism initially generates projected embeddings, or state estimates $\mathbf{z}_{1:k}$, from the encoder feature map, denoted embeddings in figure 13. The measurement mask, which filters out any invalid measurements, is applied to the projected embeddings as well as to the measurements. The decoders object classifier processed the state estimates outputting a score for the and saving the N_q (number of object queries) Top-K scoring indices. These indices are used to select which state estimates that are to be passed to the two linears to produce the final object queries and positional encodings for the decoder.

Similarly the proposed projected embeddings are fed to the second prediction head, producing initial measurement reference points and the filtered radar measurements are added after passing a log function to convert measurements into suitable range of $0.0 - 1.0$. The Top-K indices are then chosen as the initial N_q decoder reference points used later in the iterative refining process.

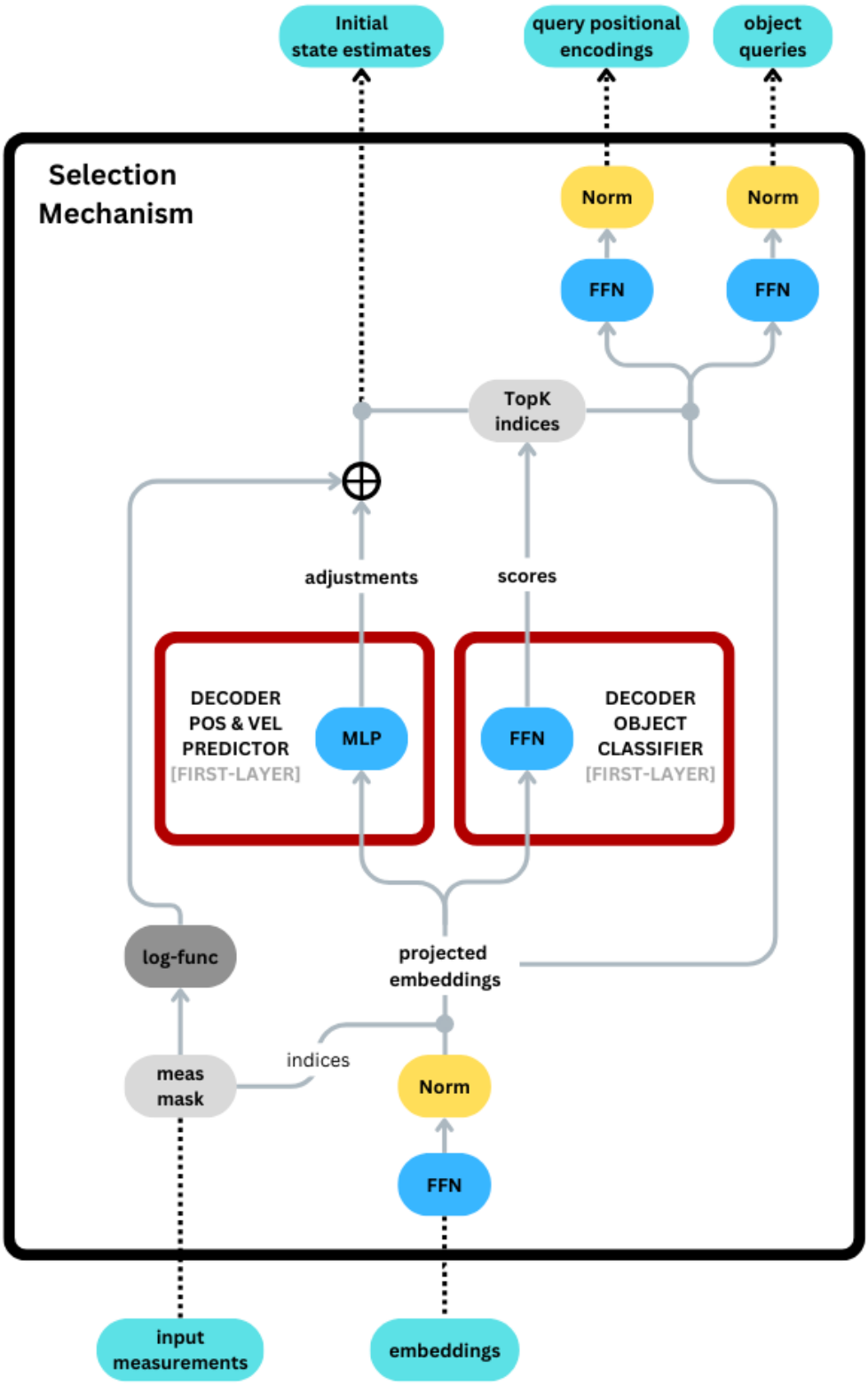


Figure 13: MT3v2 selection mechanism illustrated in detail. The object queries are of size $N_q \times d_{model}$

2.5.5 MT3v2 Decoder

The MT3v2 uses a standard transformer decoder with iterative refinement. The previously computed object queries and positional encoding are initially processed by self attention mechanism and further by cross attention, which is then applied to between the encoder outputs and the N_q object queries. The feature output is the processed by several layers of normalization and linears to output a final set of N_q object queries according to Figure 14. The decoder head in every decoder layer yields the logits and state estimates from the object queries in the refinement with the help of a learnable multi layer perceptron (MLP) or FFN. These object queries are then passed to the next decoder in line and to the prediction heads to save intermediate predictions and perform iterative refinement.

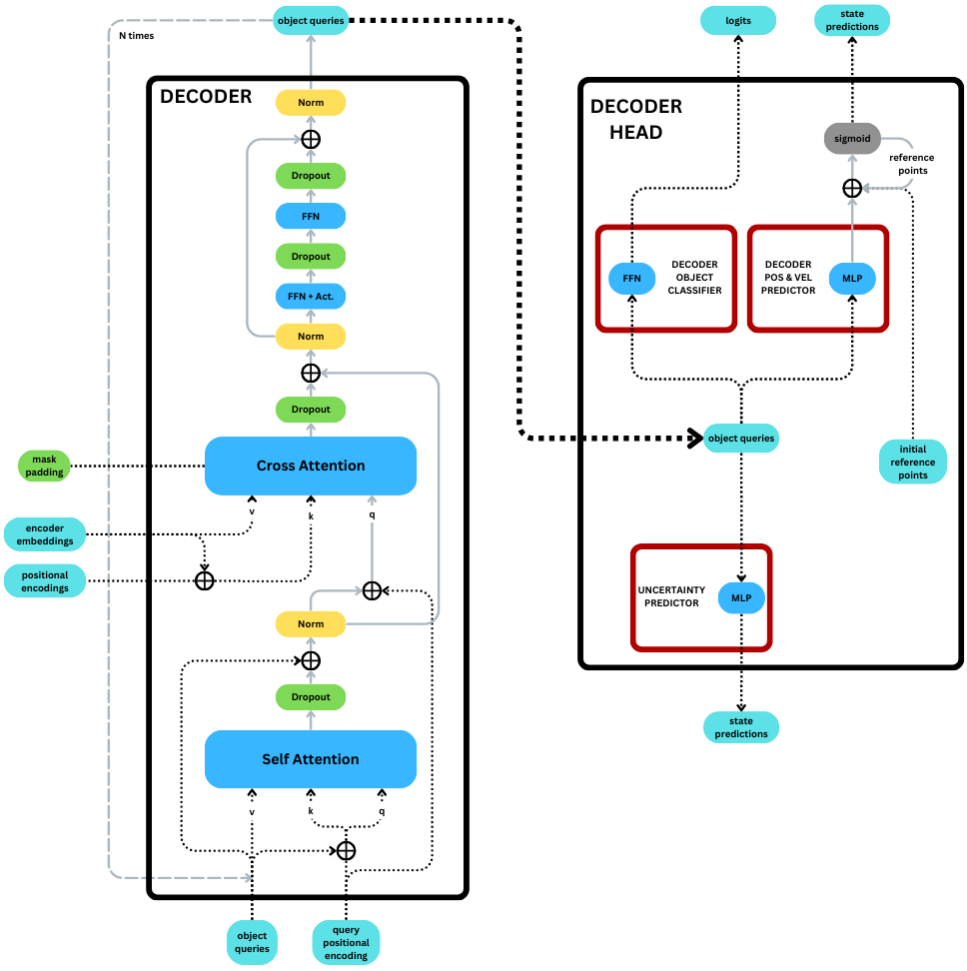


Figure 14: MT3v2 decoder and head illustrated in detail.

Iterative refinement

The iterative refinement approach is aimed at enhancing the accuracy of the model, instead of directly predicting the final state means from the decoder's last output layer. As

previously described, the selection mechanism produces initial state estimates, denoted as $z_{1:k}$, which serve as a baseline for further refinement. Each layer in the decoder then generates adjustments $\Delta^l_{1:k}$ to these estimates at the decoder head, incrementally adjusting the state predictions at every decoder head. This layered approach progressively refines predictions, leading to better accuracy [5]. While the state prediction undergo iterative refinement, the existence probabilities and uncertainties are computed directly at each decoder layer by the classifier heads.

2.5.6 Loss Functions

The final output of the network produces 3 different results. The state prediction, $N_q \times 4$ containing position and velocity. The logits prediction, $N_q \times 1$ containing alive objects. Lastly the uncertainty prediction containing $N_q \times 4$ corresponding to position and velocity uncertainties. The logits, and states together with their uncertainty corresponds to the parameters of a multi-bernoulli density (state distribution and existence probability for each component) [5].

Using a Hungarian matching algorithm it computes cost matrix based on the Euclidean distance between predicted and target states and furthermore incorporating predicted probabilities to penalize unlikely matches. The optimal assignment of predictions to targets is then determined by minimizing this cost matrix. Following this matching, the network calculates three loss components: logits loss, state loss, and an uncertainty loss.

Hungarian Matching

This function handles predictions of states, which include position and velocity, matching them to their respective ground truth counterparts. A cost matrix is computed using the pairwise Euclidean distance between the predicted states and the target states, assigning each predicted state to each target state. This cost matrix is further adjusted by subtracting the log of the predicted probabilities, incorporating the confidence of the logits predictions into the cost. The function then performs optimal matching on the adjusted cost matrix finding optimal assignment of predicted states to target states that minimizes the total cost:

$$C_{ij} = \|\mathbf{p}_i - \mathbf{s}_j\|_2 - \log(\sigma(\mathbf{l}_i)) \tag{2.33}$$

where the sigmoid function σ is defined as:

$$\sigma(\mathbf{x}_i) = \frac{1}{1 + e^{-\mathbf{x}_i}} \tag{2.34}$$

where $\mathbf{p}_i$ and $\mathbf{l}_j$ are the predicted state and logit for query i and $\mathbf{s}_j$ the ground truth state for target j. The goal is to find the permutation π that minimizes the total cost:

$$\min_{\pi} \sum_{i} C_{i,\pi(i)} \tag{2.35}$$

The algorithm importantly also returns the indices $(i, \pi(i))$ which define the optimal

matching assignments which will be used for further loss computation and model training.

Logits Loss

The logits loss converts the raw logits from the object classifier into probabilities, comparing them against binary ground truth labels with binary cross-entropy loss. This loss penalizes incorrect predictions, driving the model to improve its accuracy in detecting the presence or absence of objects. It further balances the predicted probabilities against the actual object presence, ensuring robust and reliable performance in tracking tasks.

The total binary cross-entropy loss is the sum of the individual losses is expressed as:

$$\text{BCE}(\mathbf{l}, \mathbf{t}) = -\frac{1}{N} \sum_{i=1}^{N} \Big(\mathbf{t}_i \log(\sigma(\mathbf{l}_i) + (1 - \mathbf{t}_i) \log(1 - \sigma(\mathbf{l}_i)) \Big) \tag{2.36}$$

where $\mathbf{l}_i$ is the predicted logit for the i-th query and $\mathbf{t}_i$ is the binary ground truth label for the i-th query logit, with each entry taking a value of either 1 (object present) or 0 (object absent). σ is defined as in 2.35.

State Loss

The state loss ensures that the model predicts states closely with the ground truth states by use of a the negative log-probability. This is done by treating the model's predictions are as distributions, giving each predicted state (position and velocity) an associated uncertainty. This allows the model to express a confidence and adjust the loss accordingly.

With the indexing from the Hungarian Matching the most likely states (position and velocity: $[\mathbf{x}, \mathbf{y}, \mathbf{v_x}, \mathbf{v_y}]$) to represent an object are matched with the output from the Uncertainty Predictor. These outputs correspond to a mean value (the predicted states) μ_i and an associated uncertainty (standard deviation) σ_i for each dimension. A normal distribution of predicted states can thus be represented:

$$\mathbf{P}_i \sim \mathcal{N}(\mu_i, \sigma_i) \tag{2.37}$$

where μ_i is thee predicted mean for the i-th dimension of the state and σ_i is the predicted standard deviation for the i-th dimension of the state. Given the predicted distribution for each state dimension, the next step is to calculate how probable the actual observed (target) state T_{ij} is under this distribution. This is done using the log-probability of the target state under the predicted normal distribution:

$$\log p(T_{ij} \mid \mu_{ij}, \sigma_{ij}) = -\frac{(T_{ij} - \mu_{ij})^2}{2\sigma_{ij}^2} - \log(\sigma_{ij}\sqrt{2\pi}) \tag{2.38}$$

where: T_{ij} is the ground truth value for the j-th dimension of the i-th state. μ_{ij} is the predicted mean for that dimension. σ_{ij} is the predicted standard deviation (uncertainty).

This expression captures two key components:

- **Squared Error:** $(T_{ij} - \mu_{ij})^2$ represents the squared difference between the predicted state and the actual state. The closer the prediction μ is to the true value T, the higher the probability.

- **Uncertainty Weighting:** The squared error is divided by $2\sigma_{ij}^2$, which means that larger uncertainties σ_{ij} will result in a smaller penalty for the same error, reflecting less confidence in the prediction. However uncertainties σ below 1 will exponentially increase the loss.

- **Negative Log:** $\log(\sigma_{ij}\sqrt{2\pi})$ is meant to handle this reversed exponential increase as the standard deviation *sigma* becomes smaller than 1.

The model's objective is to maximize the likelihood that its predictions match the actual observed states. In practice, this is done by minimizing the negative log-likelihood, which is equivalent to minimizing the negative log-probability:

$$\text{State\_loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{d}\log p(T_{ij} \mid \mu_{ij}, \sigma_{ij}) \tag{2.39}$$

This loss function penalizes predictions that are far from the target values, with the penalty being adjusted by the predicted uncertainties:

- **High Confidence, Large Error:** If σ_{ij} is small (high confidence), even a small error ($(T_{ij} - \mu_{ij})^2$) results in a large loss.

- **Low Confidence, Large Error:** If μ_{ij} is large (low confidence), the same error results in a smaller loss.

This is better dicernable in the following plots of the log probability:

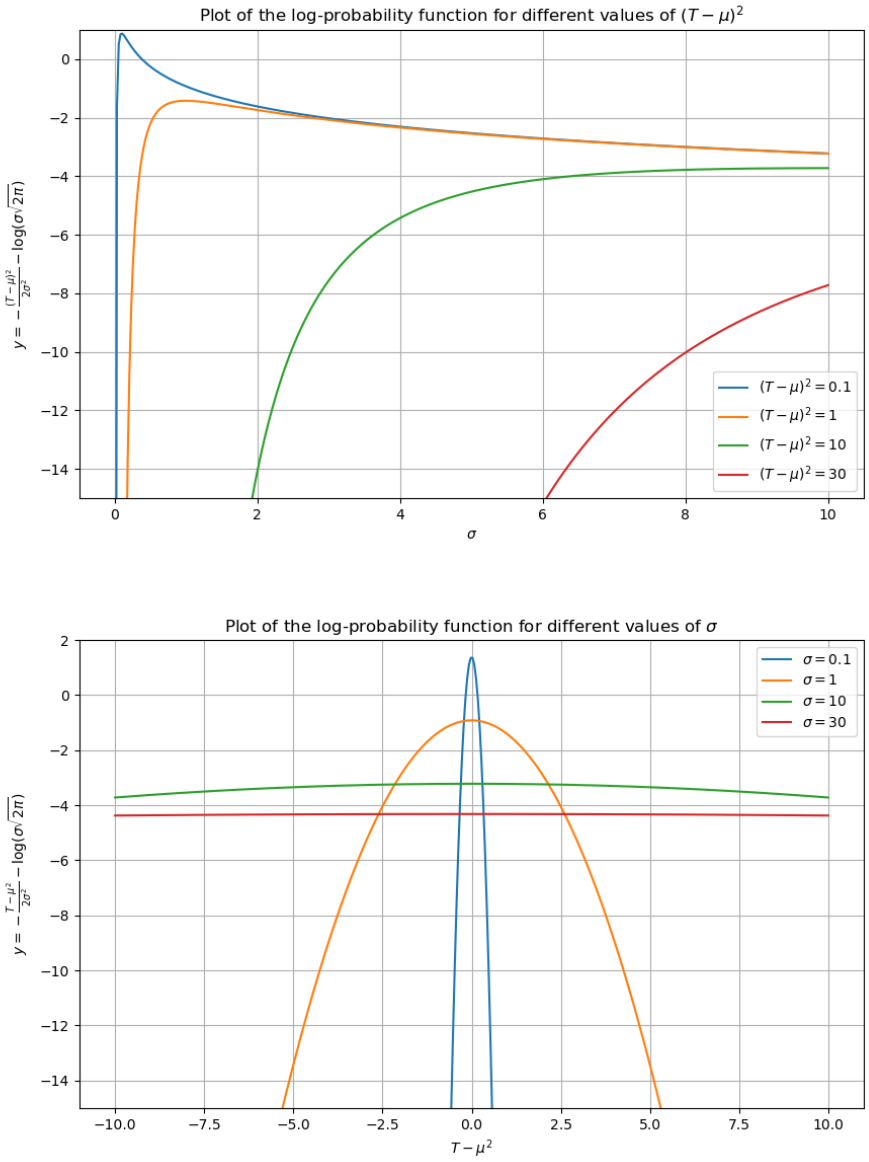


Figure 15: Log probability function illustrating loss for different values of μ and σ

2.5.7 Contrastive Auxiliary Module

The contrastive auxiliary module is tasked with classifying the embeddings directly from the encoder module. Linear transformation projects each input measurement vector into a new space through a linear layer, extracting and learning feature representations.

$$\text{Linear transformation:} \quad z_{1:n} = \text{FFN}(e_{1:n}) \tag{2.40}$$

Following this, the vectors undergo normalization, scaling them to a unit hypersphere.

$$\hat{z}_{1:n} = \frac{z_{1:n}}{\|z_{1:n}\|} \tag{2.41}$$

Once normalized, dot products between all pairs of vectors are computed to measure the degree of similarity between them.

$$D_{ij} = \hat{z} \cdot \hat{z}^T \tag{2.42}$$

This computation creates a similarity matrix that highlights how each vector relates to every other vector. The diagonal elements, denoted δ_{kron} for Kronecker-Delta, are masked in this matrix to avoid self-similarity. Additionally, any padding elements, denoted M_{pad}, are also masked when dealing with different length measurements in a batch.

$$D'_{ij} = \begin{cases} -\infty & \text{if } \delta_{ij} \text{ or } M_{pad} \text{ is true} \\ D_{ij} & \text{otherwise} \end{cases} \tag{2.43}$$

Finally, the log softmax function is applied to the similarity matrix, converting the scores into log probabilities.

$$\mathbb{P}_{ij} = \log\left(\frac{e^{D'_{ij}}}{\sum_{k=1}^{N} e^{D'_{ik}}}\right) \tag{2.44}$$

2.5.8 Contrastive Loss function

The Contrastive Loss function calculates a loss that measures the difference between predicted log probabilities and actual unique identifiers for objects (1) and clutter (0).

Let $\mathbf{u} \in \mathbb{R}^N$ be the vector of unique IDs for each measurement, where N is the number of measurements. The dimensions are expanded by performing an outer product $\mathbf{U} \in \mathbb{R}^{N \times N}$ as follows:

$$\mathbf{U} = \mathbf{u} \otimes \mathbf{1}^T \tag{2.45}$$

where $\mathbf{1} \in \mathbb{R}^N$ is a vector of ones. The resulting matrix $\mathbf{U}$ has each element:

$$\mathbf{U}_{ij} = \mathbf{u}_i \tag{2.46}$$

The ID matrix $\mathbf{I}$ is then created by comparing elements:

$$\mathbf{I}_{ij} = \begin{cases} 1 & \text{if } \mathbf{u}_i = \mathbf{u}_j \\ 0 & \text{otherwise} \end{cases} \tag{2.47}$$

The diagonal of this matrix is masked to ignore self-similarities, and the matrix is normalized row-wise to ensure it forms a valid similarity distribution.

$$\mathbf{I}'_{ij} = \begin{cases} 0 & \text{if } i = j \\ \mathbf{I}_{ij} & \text{otherwise} \end{cases} \tag{2.48}$$

$$\hat{\mathbf{I}}_{ij} = \frac{\mathbf{I}'_{ij}}{\sum_{k=1}^{N} \mathbf{I}'_{ik}} \tag{2.49}$$

where $\hat{\mathbf{I}}$ is the row-wise normalized ID matrix; each row sums to 1.

The log probabilities are then scaled by these normalized similarities by computing the product of log probabilities and the normalized ID matrix denoted as $\mathbb{P} \in \mathbb{R}^{N \times N}$. The final loss product is computed by summing these scaled log probabilities $\mathbb{L} \in \mathbb{R}^{N \times N}$, and averaging over measurements that have non-zero contributions:

$$\ell_i = \sum_{j=1}^{N} \mathbf{L}_{ij} \tag{2.50}$$

Determine the number of measurements with non-zero loss contributions.

$$N_{\text{eligible}} = \sum_{i=1}^{N} \mathbf{1}_{\{\ell_i \neq 0\}} \tag{2.51}$$

Compute the average loss across all eligible measurements.

$$\mathcal{L} = \frac{\sum_{i=1}^{N} \ell_i}{N_{\text{eligible}}} \tag{2.52}$$

This approach ensures that the model learns to distinguish between different objects and clutter effectively, focusing on meaningful differences in the data.

2.6 CARLA

Car Learning to Act (CARLA) is an open-source simulation platform introduced in 2017 for autonomous driving research. Developed by the Computer Vision Center at the Universitat Autònoma de Barcelona, CARLA was designed to generate realistic data from various driving scenarios, providing a safe and controlled environment for testing and developing new algorithms [36]. The simulation tool has gained widespread adoption in both academia and industry, particularly for tasks that require large amounts of training data, such as machine learning models for autonomous vehicles [37],[38]. CARLA's versatility allows for the simulation of diverse driving conditions, making it an invaluable resource for advancing autonomous driving technology.

2.6.1 Sensors

CARLA offers sensors in many modalities including radar, which can be mounted on different parts of the vehicle or environment. However, for our purposes an in-house sensor, developed by the Safe-Radar company, will be used, providing the following measurements in a single detection:

- Range (Distance): The distance to the target.
- Velocity (Doppler): The relative velocity of the target.
- Azimuth (Angle): The horizontal angle to the target.

- Elevation (Theta): The vertical angle to the target
- Radar cross section (RCS): The intensity of the returned signal, often referred to as the Radar Cross-Section, gives an indication of the target's reflectivity and size.
- Signal to noise ratio (SNR): Signal to noise ratio in a detection.
- Tag: The object type eg. Vehicle, building, sign etc.
- ID: Every object in the simulation has its own ID number.
- t: The time step ($\Delta t = 0.05$) measured in seconds.

It is important to note that during a single time-step, multiple detections can occur. Although the simulated FMCW-radar operates at a frequency of 20 Hz, there is no fixed limit on the number of detections per time-step. Typically, the radar registers around 15 detections per time-step, depending on the complexity of the environment and the number of objects within the radar's range and with randomness inherent to the problem. Furthermore it is worth noting that radar modules are usually set at angle (upwards) such that reflections from the road surface itself are as far as possible omitted. Conclusively these measurements provide a comprehensive understanding of the target's position, speed, and characteristics.

2.6.2 Maps

CARLA features a variety of maps that simulate different driving environments, each designed to test the performance of autonomous systems under diverse conditions. For the scope of this project experiments will be conducted in urban maps, designed to represent complex environments including buildings, traffic signs, intersections, and pedestrian crossings. These settings makes urban maps ideal for testing algorithms in scenarios where obstacle avoidance, traffic management, and pedestrian detection are critical.

CARLA maps can also be augmented with different weather conditions, such as rain, fog, and varying lighting (day/night cycles), however these conditions are irrelevant to a radar sensor.

2.6.3 Vehicles

CARLA offers a wide range of simulated vehicles with varying behaviors and properties, allowing for the creation of diverse traffic scenarios. The platform supports the customization of several vehicle parameters.

Different vehicle models are included in the CARLA library, ranging from compact cars to large trucks. Simulated vehicles in CARLA are equipped with behavioral models that can be customized. This includes setting parameters like speed limits, acceleration, braking behavior, and proximal reaction like min distance to car allowed in any given situation. In essence this enables for simulating various driving styles, from cautious to aggressive. CARLA also supports the simulation of traffic management systems, such as traffic lights and stop signs, allowing for the testing of vehicle interactions with traffic control devices. The number and type of vehicles to be spawned in the simulation can thus be varied, enabling the creation of traffic scenarios with different levels of

congestion. These features make CARLA a powerful tool for simulating realistic driving environments.

3

Method

3.1 Generating data with CARLA

Annotated training data was generated using simulations ran in CARLA, utilizing its Python API to extract radar detections within various traffic scenarios facilitating direct extraction of labeled data from the simulation.

An arguably model free testing environment was established with the help of randomness in driver characteristics (deterministic). This diverse and semi-realistic data was considered important for training the network, as it exposed the model to a wide range of scenarios, thereby aiding to prevent overfitting and enhancing the network's ability to generalize across different situations.

The use of CARLA allowed for large-scale data generation, which was deemed essential for future developments. Moreover, the generated training data is reflective of real-world conditions to some extent, providing a solid foundation for addressing the research questions and advancing the network's testing toward practical applications.

3.1.1 Maps

Town 02 from the CARLA collection was used as a base case with different scenarios intended to simulate a realistic urban setting. The layout of Town02 includes a variation of velocity restrictions with a mix of simple intersections and straights. Town02 also has a decent amount of varying environmental objects such as a diverse array of building types, foliage, poles and fences, delivering a realistic radar backdrop in driving scenarios. The exposure to diverse and realistic training data from Town02 suites the base task description and helps to improves the model's robustness and ability to generalize, reducing the risk of overfitting to specific scenarios.

Though the map includes traffic lights and stop signs to simulate typical urban traffic control measures they were all disabled in order for a more sporadic and unpredictable traffic flow with varying levels of congestion. Cars would in either case still take turns in intersections on a first-come-first-serve basis but this measure was taken to increase traffic flow and avoid longer periods of stand still traffic. Pedestrian spawns were also disabled in the scope of this project.

Figure 1: Town02 in the CARLA simulator with the radar field of view shown as orange lines

3.1.2 Vehicles

The simulation focused on modifying two specific parameters for each driver, which remained constant throughout the entire simulation. These parameters were the adherence to speed limits, set between 90% and 130%, and the following distance to the leading vehicle, which was randomly assigned between 4 and 8 meters. While it is possible to incorporate more parameters these were the easiest to implement and deemed sufficient in the scope.

During any simulation conducted, 50 cars were spawned, omitting trucks, motorcycles, and pedestrians. This was due to a bug in the map of Town02 where larger vehicles could not be used unfortunately. The remaining cars were deemed enough to incorporate a decent variability in car size and bounding box for our purposes.

3.1.3 Radar spawn points

In order to have diverse data from different angles and sections of traffic the 12 different radar spawn points, each with a fixed angle, were chosen to generate data from. As the CARLA traffic is stochastic the data generated was deemed sufficiently diverse and generalized.

The different spawn points allows for the network to be able to associate behaviours dependant on the scenery, such as if there is a junction present in the Field Of View (FOV) or if there are buildings or fences, resulting in more clutter. Also rotation relative to the road, giving a more dynamic object birth behaviour that can enter from anywhere in the FOV.

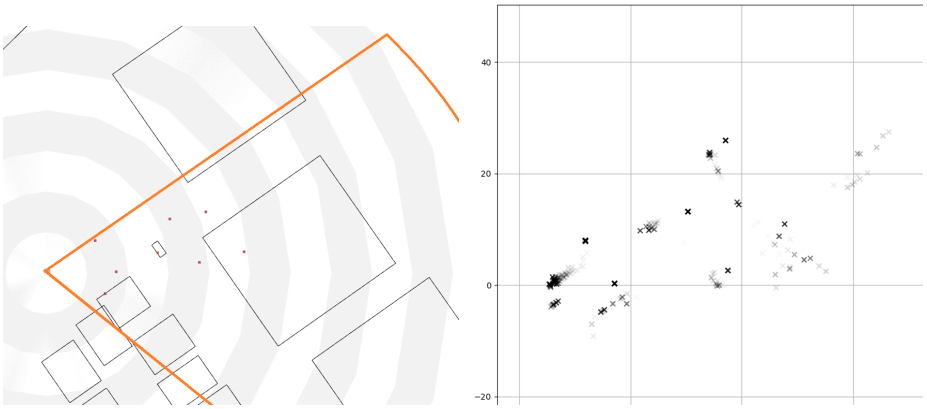


Figure 2: Left image shows radar reading in one time step while. Right image shows all readings in 20 timesteps with opacity representing older readings.

3.1.4 The radar module

Utilizing the in-house developed radar model developed by SafeRadar realistic radar detections were extracted with CARLA APIs. Though CARLA provides radar sensors, the model utilized was developed with the aim to resemble a FMCW-radar akin to the one that SafeRadar themselves use and developed in real-life. The in-house developed radar is capable of outputting complex and realistic radar detections, however, more complex signals were omitted to establish a base case. This entailed simplifying object detection information and setting certain boundaries in line with the scope of this project.

[{"ids_truth": [2928], "bboxes_truth": {"2928": [[5.547796282970877, 2.031017600161507], [-8.473318601855508, 5.975268706959914], [4.717525005340576, 1.894826889038086]]}},

{"t": 1716635824.762922, "pointcloudx": 14.280449999999998, "pointcloudy": 1.68045, "tag": 6, "id": 2830, "r": 14.378983427384565, "vx": 0.0, "vy": 0.0, "vr": 0.0, "phi": -0.11713616926017317, "snr": 31.381053569996094, "rcs": 2.9890479810219857},

{"t": 1716635824.762922, "pointcloudx": 29.70045, "pointcloudy": -1.4095499999999999, "tag": 14, "id": 2967, "r": 29.733879017124554, "vx": -2.1457550288210117, "vy": -4.733994297380112, "vr": -1.9242455075664906, "phi": 0.04742329370608641, "snr": 29.23431743851887, "rcs": 14.265875920103364},

{"t": 1716635824.762922, "pointcloudx": 23.88045, "pointcloudy": -3.0595499999999998, "tag": 14, "id": 2932, "r": 24.075646167964006, "vx": 4.400380210077230 5, "vy": 5.027470791890269, "vr": 3.900976003209116, "phi": 0.1274252596476579, "snr": 32.32304923829085, "rcs": 12.077216693934925},

{"t": 1716635824.762922, "pointcloudx": 2.70045, "pointcloudy": 0.87045, "tag": 14, "id": 2928, "r": 2.8372721767571045, "vx": -8.473318601855508, "vy": 5.975268706959914, "vr": -6.094008960053298, "phi": -0.31181976336824074, "snr": 66.1221700505356, "rcs": -24.098433825756437},

{"t": 1716635824.762922, "pointcloudx": 39.90045, "pointcloudy": -19.469549999999998, "tag": 14, "id": 2929, "r": 44.39717657019419, "vx": -0.012546207557879628, "vy": 0.008767681591900783, "vr": -0.015130286669571434, "phi": 0.4539637609976931, "snr": 13.84454383617128, "rcs": 14.370769278746623}]

Figure 3: Set of raw simulated FMCW-radar detections in a single time-step. The top row consists of one ground truth object, located inside the FOV.

The radar module was set to operate at a 20Hz frequenzy giving a $\Delta t = 0.05$ seconds. While the frequency and amount of time steps were set, the amount of measurements collected per time step were random due to the nature of radar with an average of roughly 15 measurements per Δt.

For a single time step $\Delta t = 0.05$ the radar measurements yields a point cloud $\mathbf{z}_t \in \mathbb{R}^{m \times n}$ defined:

$$\mathbf{z}_t = \sum_{i}^{n} \mathbf{m}_i \tag{3.1}$$

Where $\mathbf{z}_t$ is a sequence of n measurements received in window of time Δt and $\mathbf{m}$ is the received dimension of a single radar point measurement:

$$\mathbf{m} = [r, doppler, azimuth, t] \tag{3.2}$$

Where r is the range from the radar to the measurement, *doppler* the radial velocity, *azimuth* the angle, and t the time-stamp. Table 3.1 illustrates an example of an entire training sequence containing 20 time steps with 187 measurements. It is worth to note that the sequence has a varying amount of measurements per time step.

This data given from the radar model is then compared to the ground truth to see if there are any vehicles that are occluded or hit multiple times during a time step.

3.1.5 Ground Truth Extraction

Radar detections are initially collected in 3D space and then collapsed into a 2D plane. This transformation is performed to simplify the problem, reducing it to the analysis

τ	$\mathbf{t}_0$		$\mathbf{t}_1$						$\mathbf{t}_n$	$\mathbf{t}_{20}$		
M	0	1	2	3	4	5	6	7	$\mathbf{m}_i$	185	186	187
r	25.2	57.5	11.0	20.3	80.9	19.5	25.2	57.5	...	25.3	18.2	51.1
v_r	1.0	8.9	2.5	3.7	1.6	6.7	1.0	8.3	...	0.7	5.0	0.1
φ	-0.34	-0.22	0.012	0.45	0.024	-0.35	-0.34	-0.22	...	0.45	0.02	-0.35
t	0.0	0.0	0.0	0.0	0.0	0.05	0.05	0.05	...	1.95	1.95	1.95

Table 3.1: Example of 20 time steps containing 187 measurements equalling 1 second of measurements

of areas rather than volumes, which would present a more complex challenge for the network. This approach aligns with the scope of the project, ensuring that the network's task remains manageable and focused.

The ground truth labels, consist of three sets of measurements for each vehicle ID and time-stamp, as can be seen in Figure 3. For every set of measurement $\mathbf{z_t}$ the ground truth states for any object within the FOV is recorded:

$$\mathbf{y}_{gt} = [\mathbf{ID}] : [[x, y], [v_x, v_y], [b_l, b_w]] \tag{3.3}$$

Vehicles are included in the ground truth labeling if they are within the radar's detection area which is a cone with a range of 100 meters spanning $\pm$ 70 degrees from center. An object occlusions is here determined if any of the bounding box lines spanned by the vertices (in 3D) are blocked from the radar's direct line of sight, rendering it invisible. From a tracking perspective it means that the network will be dealing with object births when any part of the vehicle enters the FOV and is visible in a direct line of sight, and with deaths when occluded or leaving the FOV permanently.

In reality radar detections do not necessarily reflect directly from an object of interest but could bounce of various other nearby surfaces carrying information from multiple reflections. This kind of signal processing has not been expanded upon within simulator and deemed to have nearly negligible significance on the result.

3.2 Training data

3.2.1 Datasets

Two different types of datasets were created for training the network, one with a specific type of vehicle, resulting in every vehicle having the same bounding box and movement constraints. The other dataset consists of vehicles with varying sizes and movement constraints resulting in a dataset where it is harder to accurately predict the velocity and correct size of individual bounding boxes.

The **homogeneous** dataset used an Audi A2 type vehicle for all objects and further constrained the behaviour of each individual object to have the same movement profile in terms of following speed limits, accelerations and leading distance to front cars. This dataset is considered to be the baseline standardized task that aims to create a scenario where the network can specialize at giving accurate predictions without needing to

account for different behaviours.

The **heterogeneous** dataset used 20 kinds of different cars brands with various sizes, but each with a different, permanent, driving characteristic as described in section 3.1.2.

3.2.2 Transformer training data

Data was then collected for each simulation modality as mentioned in the the previous section 3.2.1 and divided into training and evaluation data. The training data was collected from 10 different radar spawn points while the validation data was collected from 2 different radar spawn points and used to periodically evaluate its current state with the GOSPA measurement. Each dataset contains approximately 150,000 time steps, where one second corresponds to 20 time steps, each including their respective radar spawn points.

For training, a time step was randomly selected, and a sequence of n consecutive time steps was extracted. Each measurement m used for training contains the subset z_{train}

$$z_{train} = [[r, v_r, \varphi, t], [ID, x, y, v_x, v_y, b_h, b_w, t]] \tag{3.4}$$

derived from the radar point cloud and corresponding ground truth at each time step. 3.1 is meant to illustrate the order of any given sequence. These sequences are concatenated and fed to the transformer as training data, represented as:

$$\mathbf{z}_{1:n} = \mathbf{z}_{t_0} \quad ... \quad \mathbf{z}_{t_n} \tag{3.5}$$

Given this data generation approach, the number of possible unique sequences from a dataset containing N time steps with radar readings is $N - n$.

The labels are only stored for the last time step of the batch, containing the center point, velocity vector and extent of all the objects present in the state space. The loss functions that are penalizing the network are using this data to change the networks weights during training.

To evaluate the network over the course of training, sequences from the evaluation datasets are fed to the network in order to estimate the predictions on data that is previously unseen. The estimates made are scored with a GOSPA score as described in the section 2.2.4 and are used to keep track of convergence during the training process. This forces the trained network make predictions in a scenery that it is not specifically trained to do and requires the network to be general in its predictions regarding the task at hand.

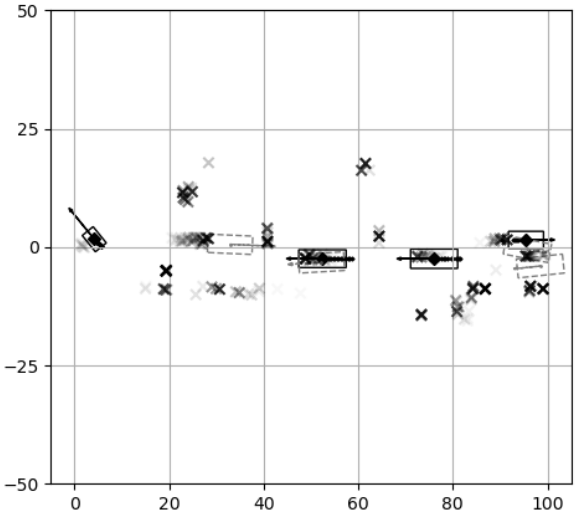


Figure 4: An illustration of the final prediction (gray) and lables in the final output of the model (black)

3.3 Transformer network MET3v2

Adaptions were made to the MT3v2 network to enable tracking for the data that was simulated in the CARLA simulations. The adaptation to the original architecture resulted in the new network Multi Extended Target Tracking Transformer v2 (MET3v2) which is trained on a sequence of $\mathbf{Z}$ readings and predicts the center point, velocity vector and the bounding box of an object in the state space.

3.3.1 Selection mechanism

The selection mechanism chooses N_q object queries to address the set-prediction problem. These object queries serve as the initial predictions in the first decoder layer and act as "educated guesses" on the encoder embeddings, distinguishing objects from clutter. The likelihood scores are ranked, and the top K selections are used as object queries. This approach has been tested only in scenarios where each object produces a single point per time step, leaving its effectiveness in EOT uncertain. In EOT, an object can generate multiple readings, which are individually scored without correlation. To adapt the mechanism for EOT, a gating approach was introduced, masking nearby readings likely to come from the same object to reduce association errors.

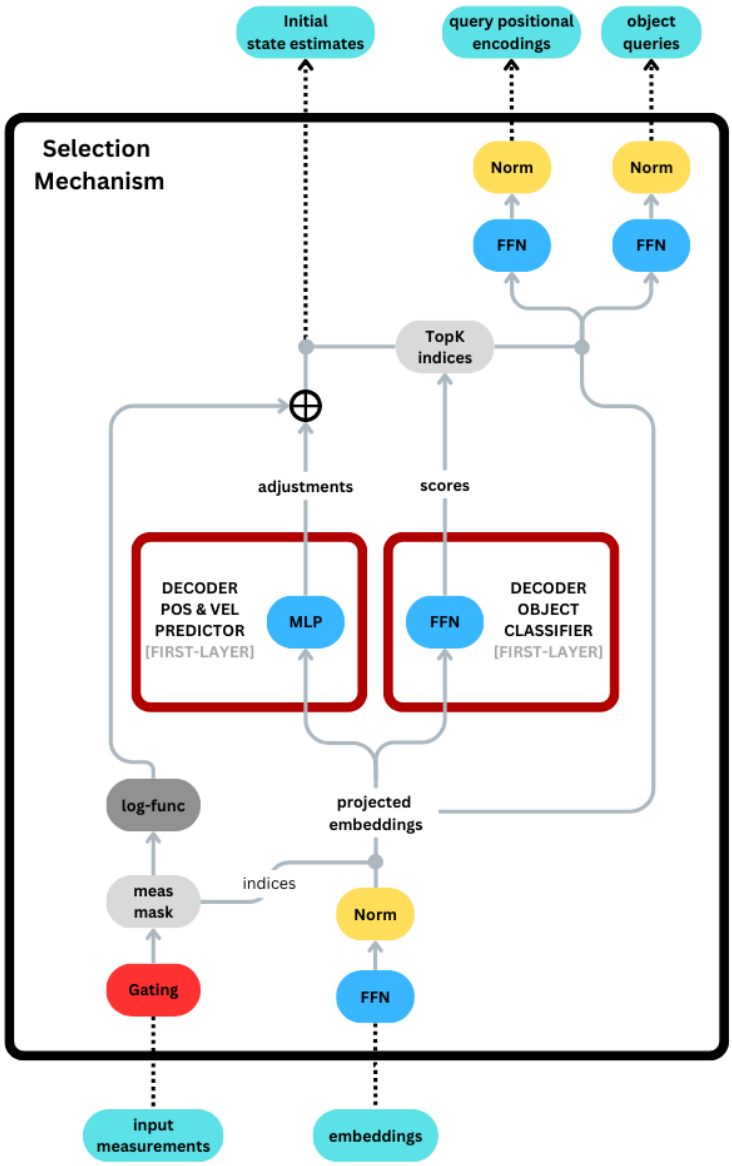


Figure 5: The new selection mechanism architecture with the gating present.

The gating was set up as for each time step in the sequence, the points are evaluated one by one and the distance to each of the other points are calculated based on a Euclidian distance in the [x,y]-plane with a threshold which could be set as a parameter. If the k points closest to the chosen point met the gating condition, the center point of those points is calculated and the point closest to the center point is kept while the other $k-1$ points are masked. An illustration of how the gating mechanism works in a single step can be seen in figure 7.

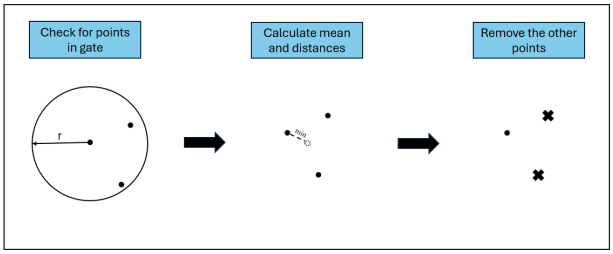
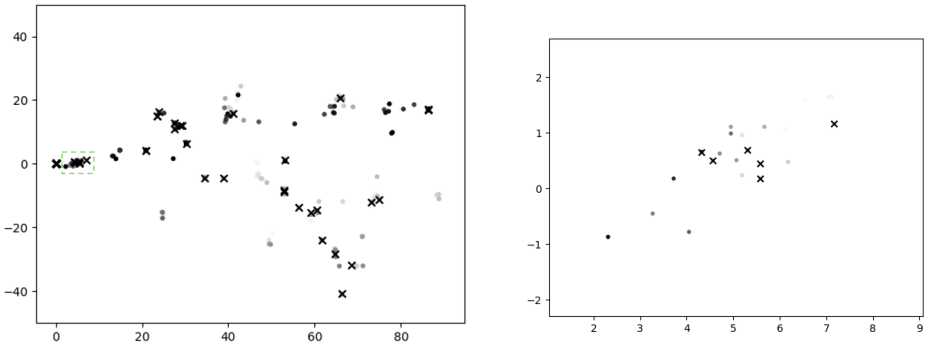


Figure 6: An illustration showing the gating process.

The gating and resulting removals in an example point cloud can be seen in figure 7. This enables a behaviour where clusters generated from the same object are simplified in order for the selection mechanism proposals to have less points to chose from. The idea is to simplify the point cloud from points that stems from the same object and accelerates the learning since the object queries will be more spread out and the initial attention more divided in the early stages of the training.



(a) Point cloud with gated measurements. (b) Zoomed area as indicated in (a).

Figure 7: How the gating changes a point cloud.

3.3.2 Extended Objects

Since the networks new task is to predict positions of extended objects, a novel loss function was introduced in the hopes of solving this problem. The contribution to the loss function is the prediction and labeling of the width and height of the vehicles, that together with the position and velocity can span a bounding box and its rotation. There are many ways to formulate such a loss and has been extensively researched, mainly in the image tracking domain.

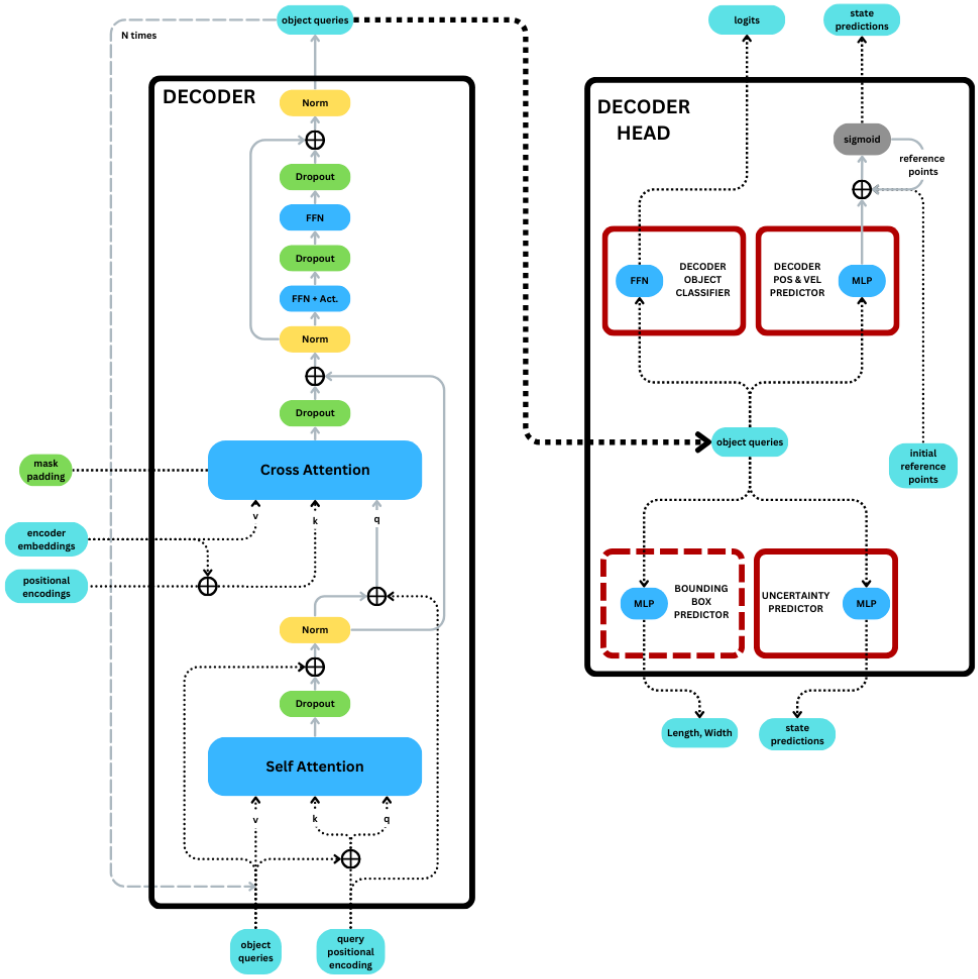


Figure 8: MET3v2 decoder and head illustrated with the added bounding box MLP.

Bounding box loss

A new MLP was added to take vehicle sizes into account, predicting the vehicle length and width. A PyTorch CIoU loss function was implemented which uses a pair of vertices spanning the prediction and ground truth bounding boxes to determine the loss. While it does take into account the aspect ratio it does not handle rotation definitively.

The initial idea was for the MLP to predict bounding-box in any given rotation and anywhere on the map, spanning $\pm$ values on the y-axis. However, the loss function only works with positive vertices on a plane and strictly with vertices such that $x_1 \leq x_2$ and $y_1 \leq y_2$. Some efforts were made to get around these issues but eventually put on hold to progress in other areas. Thus the PyTorch CIoU remains and used in a simplified manner with prediction and ground truth vertices spanning the plane as follows: $[[10, 10], [h, w]]$, $[[10, 10], [h_{gt}, w_{gt}]]$. Further discussion about this topic ensues later on.

3.3.3 Extended GOSPA

An extended GOSPA module were added to the original GOSPA module present in the MT3v2 network. In addition to the distance between the means of the objects, the extended module also include distance scoring depending on the difference in predicted shapes and orientations. As described in section 2.2.5 the methodology builds on using the Gaussian Wasserstein distance when dealing with extended objects. Since the shapes in the ground truth and the predictions in this case are rectangular, the covariance matrix and mean are calculated in order to continue using the distance measurement. The error is then capped with the cut-off distance which otherwise would consider the track as not ascribed to a target.

$$d_{GWD} = \begin{cases} c & \text{if } d_{GWD} > c \\ d_{GWD} & \text{if } d_{GWD} \leq c \end{cases} \tag{3.6}$$

3.4 Training

In order to see the MET3v2 networks potential as a tracker and evaluate its performance, several networks architectures were trained and validated on the datasets that was generated from CARLA. A total of six network configurations were trained, three of them were trained on the homogeneous dataset and the other three used the heterogeneous dataset. In these three networks there were one architecture without the selection mechanism as described in 2.5.4, one with the selection mechanism as described in [5] and one with the new selection mechanism as described in 3.3.1. This enables the comparison between the different architectures and also allows to choose the best one for benchmarking and evaluation.

3.5 Comparison to a model based approach

To evaluate the performance of the trained networks, the GOSPA error of the networks were compared to a GGIW-PMBM filter. In [5] it was stated that a PMBM filter was the highest performing one - after outperforming the δ-GLMB tracking filter - and was therefore chosen as a benchmark for the model based comparison. Since it was tested in a POT scenario however, the addition of the GGIW for EOT was necessary. The GGIW-PMBM filter was set up in MATLAB and fed with the same evaluation data as the MET3v2 network was evaluated with during training.

Batches of the evaluation data were fed to the network and to the tracking filter. A total of 12 files from 2 spawn points containing a total of roughly 30000 time steps of data were possible for selection. The files and the sequence start are chosen by random when evaluating one tracking sequence. In order to get an overall performance value from the trackers trying to determine the state space in different scenarios, 100 tracking simulations were run on randomly chosen sequences from the files. The task is to predict the state space in last time step in every sequence. From these 100 simulations, the total GOSPA error, standard deviation, localization error, missed target error and false target error are saved for comparison.

4

Network training

The MET3v2 network was trained using the data gathered from CARLA. The parameters set regarding network configurations for each of the different tracking tasks can be seen in Appendix A. The architectures were trained on the two different datasets as described in 3.4. All training was done using a Nvidia Tesla P4 GPU on Google Cloud taking roughly 20h to train 100000 time steps per network resulting in roughly a week of uninterrupted training time to train all the network configurations.

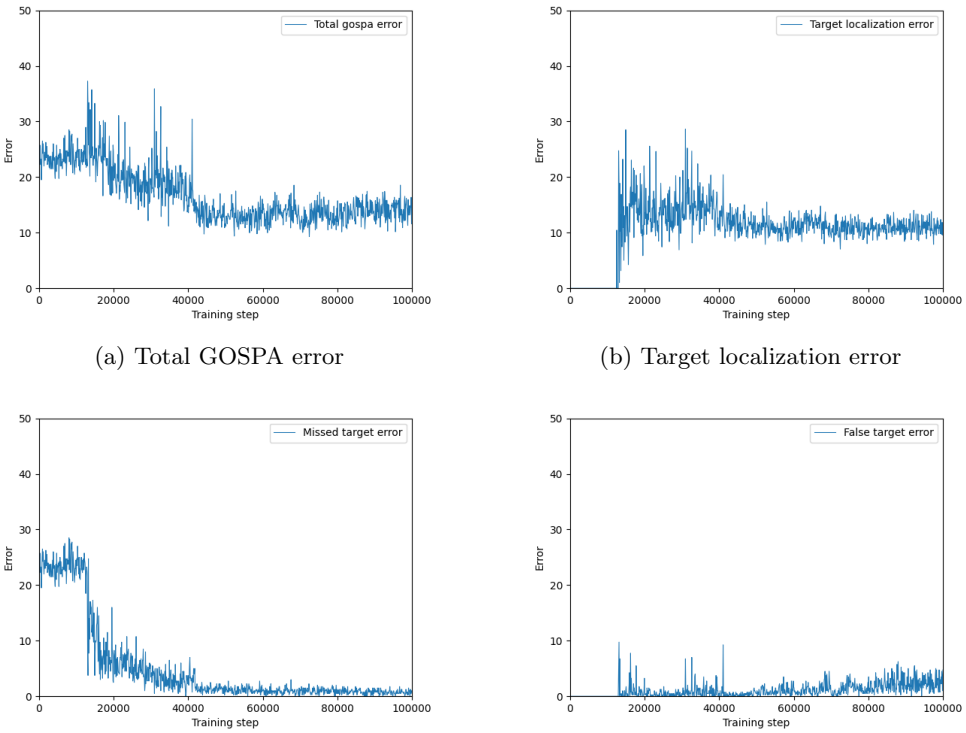
4.1 Training the MET3v2 network

All the training was done on sequences containing 20 time steps equivalent of a time window of 1 second of data fed to the network per prediction and training step.

The training is monitored by measuring the GOSPA error as described in 2.2.5 every 100 training steps to see if the network manages to make more accurate predictions on the evaluation data over time. This to see if the network is general enough to make predictions in new scenarios. The GOSPA error from evaluation is calculated on 10 separate sequences from the evaluation dataset, keeping the mean values as the final score increase reliability. For stability purposes over the course of training, only sequences that had between 3-7 vehicles present in the prediction time step were included in the evaluation datasets during training. The goal of the comparison is to find a modality that shows low GOSPA error after being fully trained and prefarably quick convergence.

4.2 Training with the original selection mechanism

The change in GOSPA error over the course of training when using the original selection mechanism on homogeneous data can be seen in Figure 1.



(a) Total GOSPA error

(b) Target localization error

(c) Missed target error

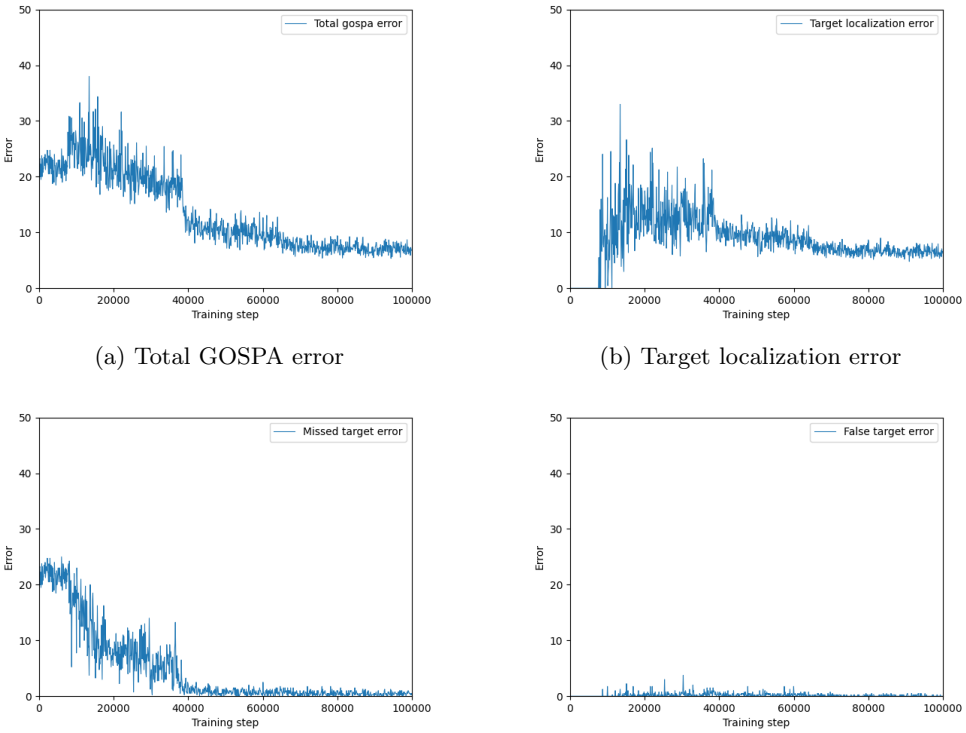
(d) False target error

Figure 1: Total and split GOSPA error using the original selection mechanism training on homogeneous data.

As can be seen from the figures, the total GOSPA error drops at around 20000-50000 steps almost halving its GOSPA scoring compared to the early stages of the training. The main contributor to the total GOSPA error in the early stages is the missed target error which accounts to the logits gating in the model having low confidence in the existence of an object where the object query is chosen. At around 15000 steps, the logits gating stabilizes and the model starts making predictions on the location of the vehicles in the state space. This leads to a more unstable behaviour in the initial part as the localization error takes over the main part of the total error since the model is still predicting the locations and extents quite poorly. As the logits stabilize, the false target error also starts giving scores since some of the queries were falsely activated.

In summary, the GOSPA scoring shows that after training, the model receives a lower score as training progresses and the model learns how to predict the state space on data that it is not introduced to in the training stage. The scoring also shows a logical progression of how the network learns the task.

By using the same selection mechanism and parameters but changing to the heterogeneous dataset we get the results presented in figure 2.



(a) Total GOSPA error

(b) Target localization error

(c) Missed target error

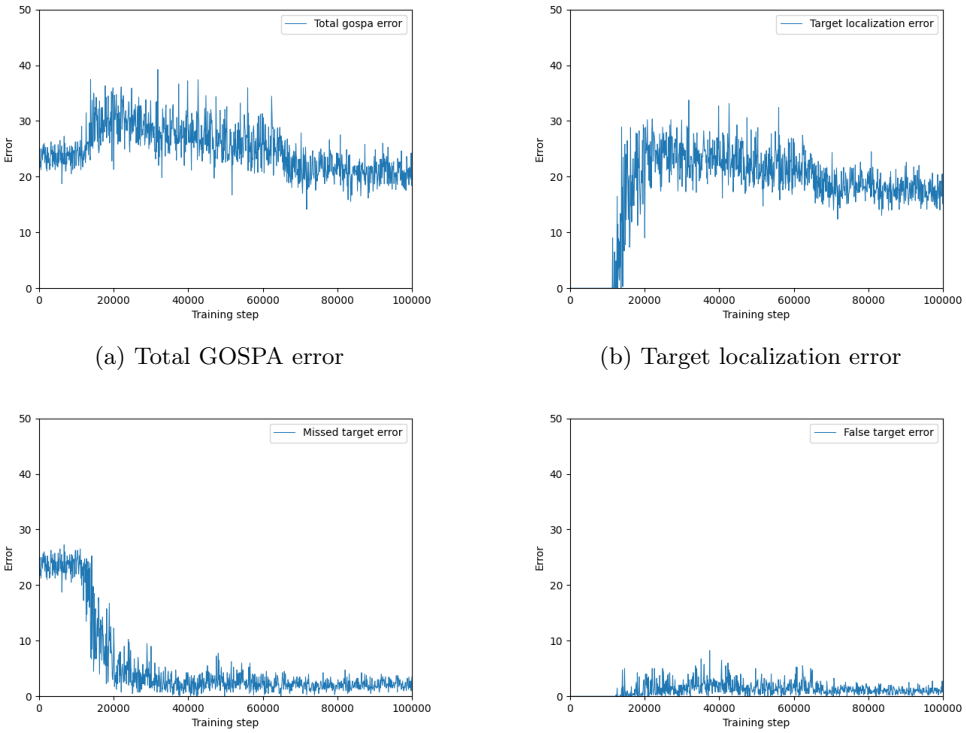
(d) False target error

Figure 2: Total and split GOSPA error using the original selection mechanism training on heterogeneous data.

By analyzing the GOSPA scoring, we can see in similar fashion as with the homogeneous dataset that the network learns the task given over time. Something that is surprising however is that the network receives a lower GOSPA score after training than when using the homogeneous dataset although it takes longer for the network to show improvement. The major difference between the end performance being the false target being considerably lower at later stages.

4.2.1 Without the selection mechanism

The modality with the selection mechanism deactivated is equivalent to using the architecture as described in the DETR architecture. The network does not shift its attention in initial steps as the methodology introduced in the deformable DETR [35] described where the measurements that seem to contain a high information value for the task at hand gets focused on initially. This entails longer training times in general as presented in [35]. The training procedure for the homogeneous dataset can be seen in figure 3.



(a) Total GOSPA error

(b) Target localization error

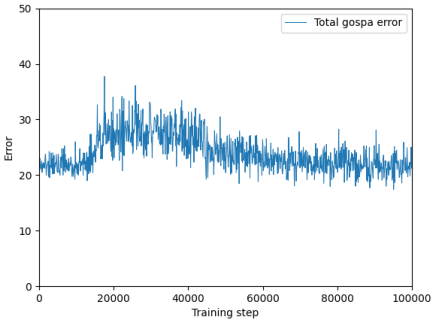
(c) Missed target error

(d) False target error

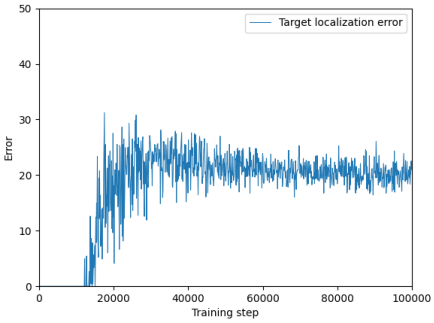
Figure 3: Total and split GOSPA error without using the selection mechanism training on homogeneous data.

The results show that the GOSPA error stabilizes with regards to the logits gating in a similar fashion as before but remains considerably higher in the localization error than when training the networks using the selection mechanism. This confirms that using the selection mechanism, even though not designed to be used for extended objects, also is beneficial for gaining good predictions in an EOT setting.

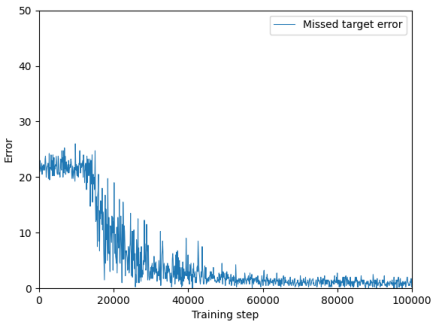
The different GOSPA errors from training without the selection mechanism on the heterogeneous dataset shows the following.



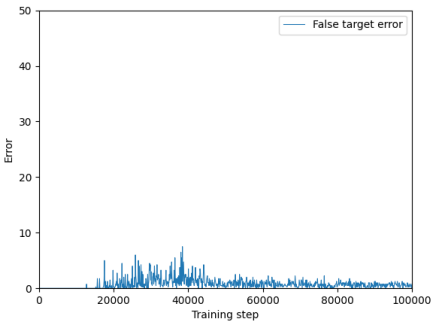
(a) Total GOSPA error



(b) Target localization error



(c) Missed target error



(d) False target error

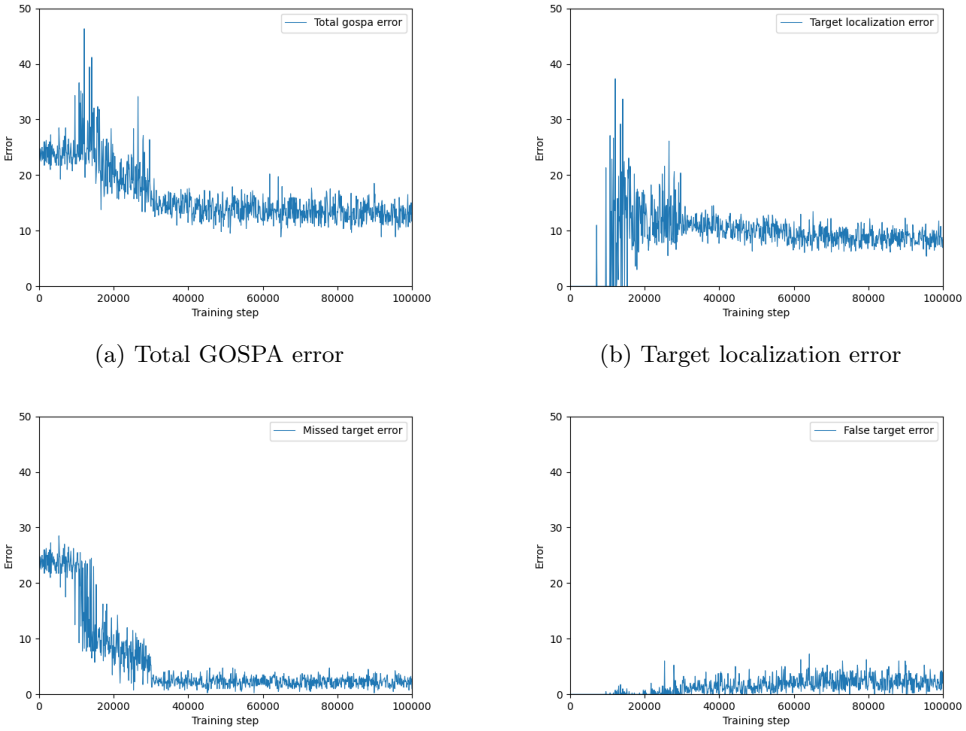
Figure 4: Total and split GOSPA error without using the selection mechanism training on heterogeneous data.

As can be seen in figure 4, the localization error is, alike in the case with the homogeneous data, the major contributing factor to the total GOSPA error while the false and missed target errors still remains low.

4.2.2 Using the altered selection mechanism

This modality as described in section 3.3.1 uses a gating approach to account for multiple measurements coming from the same object in every time step. The idea is to reduce the problem through the gating and that the network faster can focus its attention on the readings that have a high information value.

As in the earlier cases we begin by training the network on the homogeneous dataset shown in figure 5 to see how well it manages to converge over time.



(a) Total GOSPA error

(b) Target localization error

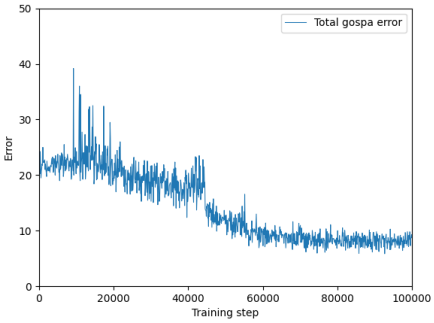
(c) Missed target error

(d) False target error

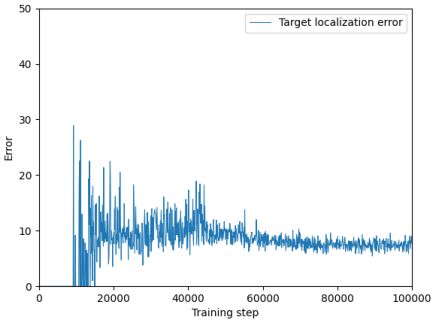
Figure 5: Total and split GOSPA error using the original selection mechanism with the added gating training on homogeneous data.

The results show that by using the altered selection mechanism during training the network performs better over time at predicting the data in the evaluation dataset. The false and missed target errors however do not converge to low levels and shows little improvement from earlier selection mechanism modalities. An improvement was expected as a result of the gating approach since the network did not have to deal with the same amount of readings that could be eligible for being an object.

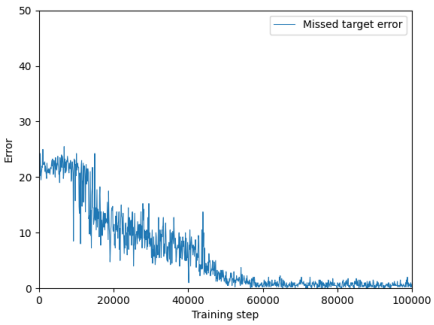
Figure 6 shows how the training procedure looks for the network training on the heterogeneous case.



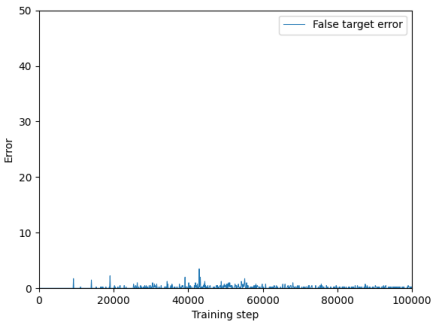
(a) Total GOSPA error



(b) Target localization error



(c) Missed target error



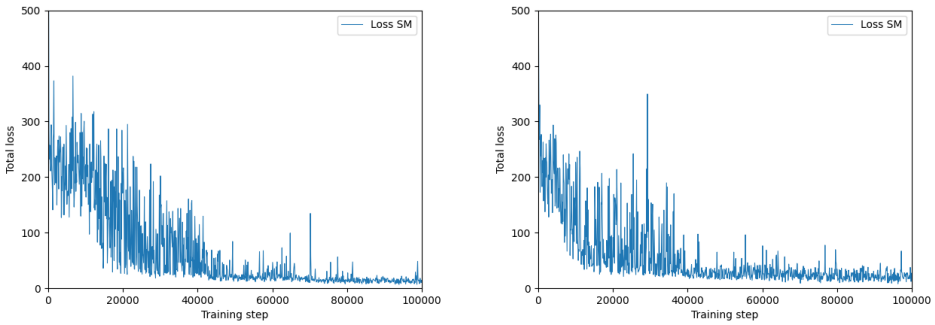
(d) False target error

Figure 6: Total and split GOSPA error using the original selection mechanism with the added gating training on heterogeneous data.

Once again the network learns the task at hand by showing that the evaluation tasks gets a lower GOSPA error over time showing that the architecture is viable. In contrary to the homogeneous case the false and missed target error stays at low levels throughout the later stages of the training, which as in the earlier case when training with the original selection mechanism is attributed to the vehicles having different properties.

4.3 Training loss

Additional to the GOSPA error, the loss function that penalizes the network during training can be monitored to see how the network loss is progressing over time.



(a) Loss over time homogeneous dataset. (b) Loss over time heterogeneous dataset.

Figure 7: Total loss over time during training using the original selection mechanism.

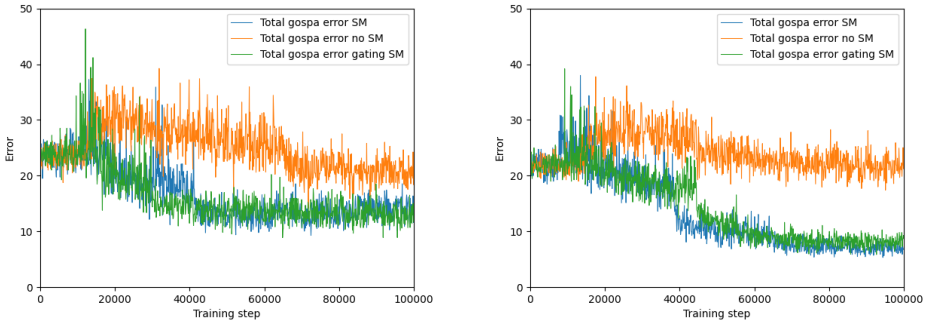
As can be seen in the figure 7, the total loss and the total GOSPA error shares similar attributes with a more or less steady decline over time. Major drops in the loss later on in the training are often due to learning rate adjustments as the learning rate decreases over training time when the loss plateaus. A similar loss behaviour over time were shown throughout all the different training sessions.

Since the loss over time and the GOSPA error over time are showing similar behaviour, the loss function used can be considered effective for penalizing the network during training to effectively lead to a desired behaviour.

4.4 Task comparison

By observing the networks ability to lower its GOSPA scoring when evaluated on the validation datasets it can be concluded that the network can be trained to handle tracking tasks that include both homogeneous vehicles and heterogeneous vehicles.

To compare the networks to one another, the total GOSPA errors over time can be seen in figure 8.



(a) Total GOSPA error over time for the homogeneous case.

(b) Total GOSPA error over time for the heterogeneous case.

Figure 8: Comparison between the different selection mechanism modalities.

The figures show that compared to not using the selection mechanism, both the modalities utilizing the selection mechanism converged faster and gave better predictions after training for 100000 training steps. The architecture not utilizing the selection mechanism showed little to no improvement over the course of training.

A comparison between training using the selection mechanism for the different datasets can be seen in figure 9.

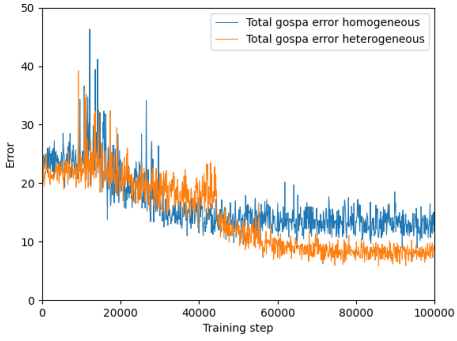


Figure 9: Total GOSPA error over time for the different datasets using the selection mechanism.

The figure shows a training behaviour where the network converges faster for the homogeneous dataset but ends up at a higher GOSPA score than in the case of the het-

erogeneous dataset which takes more time to converge but manages to receive a lower GOSPA score after having trained during the full training duration.

5

Network evaluation

In order to be able to draw conclusions about how well the trained networks building on the MET3v2 architecture serves as a tracker and to validate their efficacy, comparisons between the trained models and a GGIW-PMBM filter were made. The evaluation setup is explained in section 3.5.

5.1 GGIW-PMBM setting

To maximize the GGIW-PMBM filters tracking efficiency it needs to be uniquely tuned for the task. Tuning involves changing the parameters as introduced in section 2.3.1 and 2.3.2. For these tasks, which are considered to be of the more complex nature, tuning will need to account for both the frequent occlusions, varying amount of measurements from the vehicles per time step and their stochastic behaviour.

The parameters used in each task share some characteristics in the tuning due to the fact that the same radar is used and thus the measurement behaviour is alike in both the dataset containing homogeneous and heterogeneous vehicles. However, due to the more dynamic nature of the heterogeneous dataset, parameters concerning the uncertainty connected of the filter were changed such as the gamma parameters that models the expected extent of the objects. The parameter tuning for the different tasks can be viewed in the Appendix B.

5.2 Comparing GOSPA scoring

The task consists of predicting the bounding boxes of the vehicles at time step Z_{20} given the measurements yielded by the radar compared to the ground truth.

5.2.1 Homogeneous vehicles

The trained networks and the GGIW-PMBM filter were given sequences from the homogeneous validation dataset to make predictions on. A total of 100 sequences were ran through the trackers to yield mean GOSPA errors. This was done in order to increase the robustness and lessen the impact of outliers. The errors can be seen in table 5.1.

Table 5.1: Total and split GOSPA error from the different tackers when estimating the state space on sequences from homogeneous dataset.

Tracker	GOSPA error	Localization	Missed	False
MET3v2 SMgate	12.92±5.68	8.21	1.41	3.30
MET3v2	13.24±5.65	9.24	0.70	3.35
MET3v2 noSM	19.18±6.64	15.67	1.95	1.03
GGIW-PMBM	27.29±11.01	3.91	14.80	8.58

As can be deduced from the numbers in the table the MET3v2 network overall outperforms the GGIW-PMBM filter when making predictions on the homogeneous dataset. As previously shown during the network training, the versions of the network using the selection mechanism are the ones yielding the highest performance in this task, showing the selection mechanisms efficiency and place in the architecture. The difference between the gating approach and the regular selection mechanism architecture are however not very impactful on the overall GOSPA error. The gating approach seems to give better localization estimates while missing more targets which implies that the use of the different modalities may be situational.

5.2.2 Heterogeneous vehicles

The same procedure were followed for the evaluation of the trackers with the heterogeneous dataset.

Table 5.2: Total and split GOSPA error from the different tackers when estimating the state space on sequences from heterogeneous dataset.

Tracker	GOSPA error	Localization	Missed	False
MET3v2 SMgate	7.58±3.15	6.60	0.325	0.65
MET3v2	7.21±2.86	6.43	0.75	0.025
MET3v2 noSM	20.52±6.49	19.15	1.03	0.35
GGIW-PMBM	28.96±10.93	5.11	15.85	8.00

Similarly to the previous results, table 5.2 shows that the model free approaches performs better than the GGIW-PMBM filter when measuring with a GOSPA error. The difference in the selection mechanisms are rougly the same as in the case with the other dataset. Few conclusions about the difference between them can therefore be made since the difference lies within the uncertainty connected to the variance.

5.3 Tracker comparison

A comparison between the network and the GGIW-PMBM filter can be seen below to understand the difference in GOSPA error between the trackers. To visualize the predictions, figure 1 shows the predictions made by the trained network and the GGIW-PMBM filter. In the case of the GGIW-PMBM predictions, the vehicles are presented as ellipses instead of the rectangular shape.

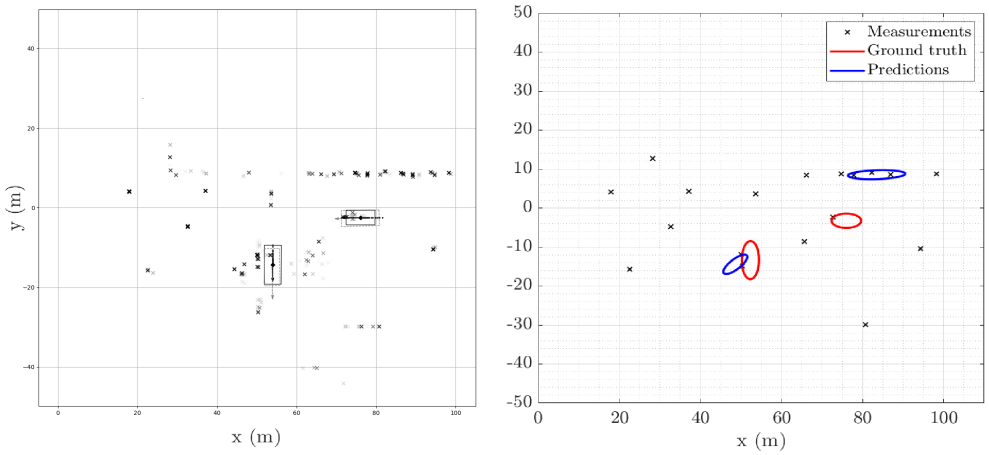


Figure 1: Left: Fully trained network prediction. Right: GGIW-PMBM prediction.

Here one can see how the predictions differ from the actual state space. Two typical behaviours of the GGIW-PMBM filter can be seen in the figure. One where a prediction is only covering a part of a vehicle because of size and orientation error, but is still relatively close with regards to the center point. Since the measurements fed to the tracker are sparse and only cover parts of the vehicle, the PMBM filter have a problem spanning up the true extent of the targets. The other typical behaviour is that the filter predicts measurements that stems from clutter (in this case a wall) to be a vehicle. By more specific tuning and changed motion models this would probably be solvable but may in turn lead to other unwanted tracking side effects.

5.3.1 Inference times

In addition to the GOSPA error measurement, the inference time is another interesting aspect when comparing the tracking approaches. Table 5.3 presents the inference times in the different network architectures and the GGIW-PMBM filter. The time is measured from the point the model is fed the data sequence generated from the file to the time that a prediction is made. The inference times presented are the mean 100 prediction runs and were run on an AMD Ryzen 5 2500U 2.00 GHz CPU.

Table 5.3: Inference times of the different trackers when analyzing a sequence of 20 time steps.

Tracker	Inference time [s]	
	Homogeneous	Heterogeneous
MET3v2 SMgate	0.6011	0.5195
MET3v2	0.4329	0.4071
MET3v2 noSM	0.4609	0.4217
GGIW-PMBM	271.9117	209.5303

Alike the results in [5] where similar tests are conducted, the inference times for these tasks when using the network trackers are considerably lower than the times given by

the GGIW-PMBM filter. The filters inference time is highly variant and dependant on the sequence given to it where the inference time is drastically increased when the filter is having many tracks active because of computational load. This kind of behaviour is not observed in the case of the model free trackers where the amount of objects to be tracked have little impact on the inference times.

5.4 Attention maps

Cross attention for a sequence is illustrated below in Figure 3 in tandem with the network output in Figure 2. These attention heads are taken during inference and show the first and last decoder layer cross attention. The attention between object queries and encoder embeddings seem to be tilted towards the end of the embedding sequence for most attention heads while some showcase attention for intermediary embeddings.

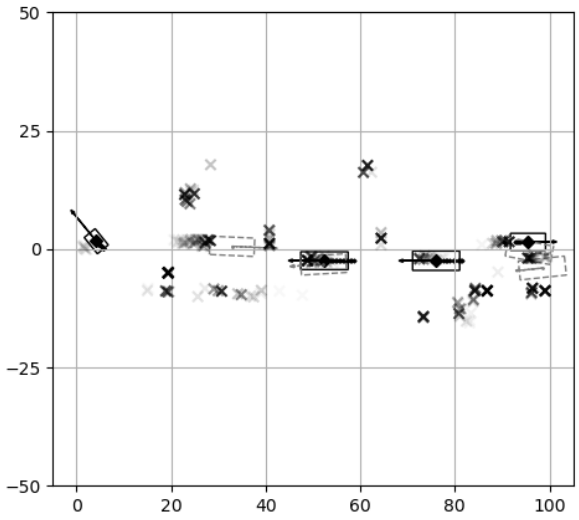
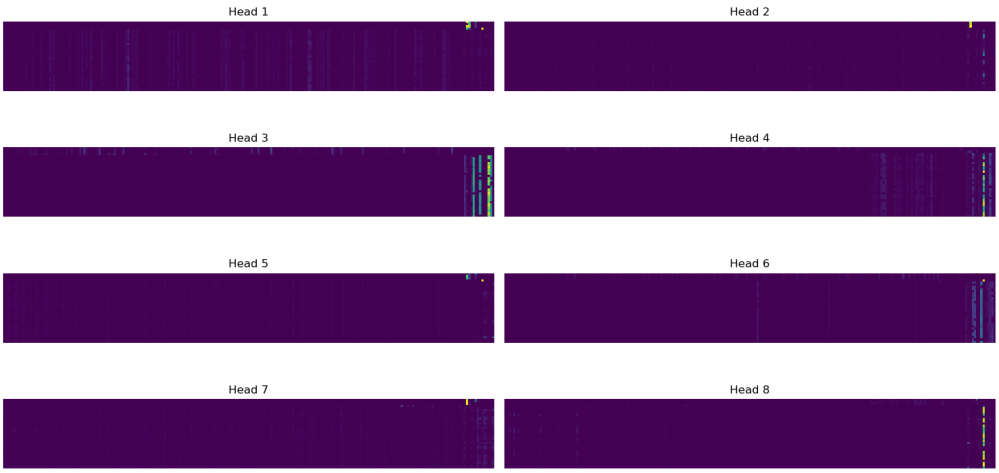


Figure 2: Illustrating the prediction output of attention maps in Figure 3.

(a) Cross attention in Decoder Block 0.



(b) Cross attention in Decoder Block 5.

Figure 3: Comparison of cross-attention in different decoder blocks.

6

Discussion

6.1 Results from the network training and evaluation

As can be seen in chapter 4 & 5 and previously shown in [5], the MET3v2 network is comparable with a tracking filter that is building on the traditionally used mathematical model based trackers. In this case the trained networks even surpasses the conventional tracking filter used as a benchmark. The evaluation was conducted by using data generated and annotated by CARLA and the radar tool which is a setting that the network architecture has not been tested in before.

The GOSPA error during network training in chapter 4 shows that both when using the data with the homogeneous and the heterogeneous objects, the trained models using the selection mechanism as proposed in [5] converges faster than without it. This suggests that the use of it even when tracking extended objects is beneficial although the usage was intended for point objects.

The benefits of using the added gating leaves little to no proof that it is beneficial to use since both of the selection mechanism architectures yielded good results. However, in the simulations that were ran, the radar rarely yields more than a few points from one object in the same time step and reduces the need of such an approach but may see its use in situations with denser point clouds where it can reduce the cloud more efficiently. Another methodology that also builds on the similarities or dissimilarities from measurements in the point cloud alike the gating approach could be to try to tie the measurements together instead of removing them and then attribute them to hypothesized objects. This could be based on other information than the position of the measurement such as the radial velocity but has yet to be incorporated in the network, let alone tested. However a distinction on how much information every data point contains needs to be formulated in order to see if removing or classifying data is the best way to aid the network.

When looking at the results from the evaluation tasks in chapter 5 the DL approach shows very good results compared to the GGIW-PMBM filter. When breaking down the different components in the GOSPA measurement, it shows that the major contributors are the missed and false target errors. The evaluation of the original MT3v2 network conducted in [5] shows a similar trend when using the setup described in the paper.

Questions may be raised about the sample size of 100 sequences used during evaluation

and that it may be to small to be able to draw accurate conclusions. Furthermore, all the evaluation runs are trained on unique sets of sequences which may skew the results somewhat between the evaluation runs if a tracker have been fed sequences that are considered more difficult than others. Multiple evaluation runs were done in order to see the variance of each of them and showed a difference in GOSPA error between $5 - 10\%$ between the runs. This indicates that the 100 sequences are enough to make the evaluation run general.

In the case of the homogeneous dataset it can be seen that the false target error in the GOSPA measurement is significantly higher than that of the heterogeneous dataset. This is interesting since the heterogeneous dataset was originally seen as the more challenging dataset because of the increased dynamics. This is most likely due to the velocity and size profiles of each vehicle that helps the network to distinguish between readings coming from different vehicles close to each other. It is however still worth mentioning that the false target error accounts for less than one false target per prediction and that the cardinality predictions of all the network settings performed over expectation even when dealing with MEOT tasks.

6.2 GGIW-PMBM

The GGIW-PMBM filter, although proven powerful as a tracker, is intricate and dependant on the many parameters and settings that are connected to it. This leads to challenges in how to tune the tracker suitably for the radar data yielded by the radar module used in CARLA and how general the tracker have to be able to handle all the unique sequences that it may encounter. The data used in this work can be considered to be complex both in terms of the dynamic behaviour of the tracked objects but also considering clutter, occlusions and variety of possible spawn points.

Since the vehicles in the simulations have a dynamic velocity profile that can accelerate and decelerate as well as changing direction, the GGIW-PMBM filters linear motion model of the velocity profile resulted in the tracks having a tough time to keep up and made it difficult for the associations while still not getting stuck on static objects that are undesirable to track. To handle this, though difficult to be done effectively, a more accurate motion model is probably beneficial for the overall tracking performance of the filter. Ways of making the motion model more dynamic and realistic exists such as interacting multiple models or the coordinated turn model [39], [40].

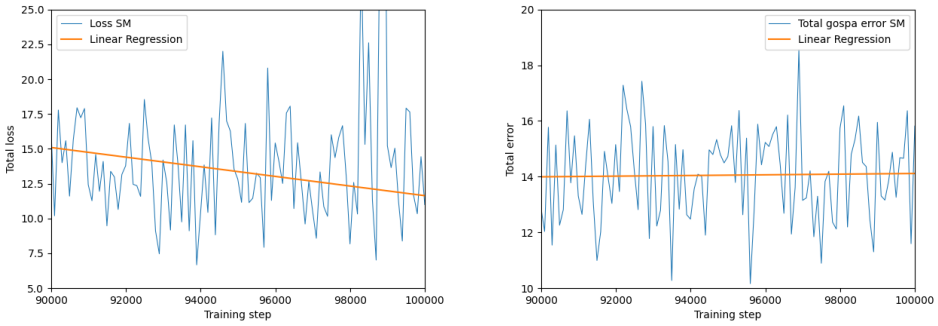
Another problem when tracking extended objects with filters is that measurements usually appear along the edge of the vehicles which gives the filter a hard time estimating the extent since it is only aware of a small part of the whole extent. This is a known problem when working with extended objects in the radar field [18]. The filter also had problems keeping up with targets further away from the radar spawn point which were getting frequent but fewer hits than the ones close to the sensor. This implies that the environment and measurement model that has been used is not ideal for tracking with the GGIW-PMBM filter. With more specific tuning, the filter would probably be able to perform better but due to the high ratio of clutter in the point clouds it would be difficult to make the filter perform on par with the DL methods without adding or reshaping the filter in any way to tailor the needs.

6.3 Network training

Since the training times were long and resources limited there were only time to train one of each network. Since DL is a stochastic process, it is difficult to know if the resulting trained networks are optimised in a way such that they have found a global optimum.

Another question to ask is if 100000 training steps are enough to train a network of this calibre. An ever actual question when working with ML of any sort is to know how many training steps are required to train a network and when it is trained enough to be considered fully trained or trained enough. Another question to ask is how big the network needs to be as well as how much diverse data is needed to not train the network to become inbred. As have been shown before, transformer networks are usually benefiting from scaling in the form of more data and more parameters to perform better [41], [42]. This implies that the architecture used could perform better and that the size is something that limits the performance from the training.

Figure 1 shows the later stages of the training when training the network on the homogeneous data with the selection mechanism architecture with a linear regression to see the trend of the data.



(a) Regression loss.

(b) Regression GOSPA error.

Figure 1: Linear regression for loss and GOSPA error over the last 10000 training steps.

The figure suggests that the loss is still decreasing in the later stage of the training while the GOSPA scoring seem to have stagnated and perhaps even increasing slightly which suggests that the network is fully trained to its maximal performance given the dataset that is available. To further enhance performance, exploring the use of a larger model or expanding the dataset to include more varied examples may be beneficial. Additionally, implementing early stopping could be considered to prevent the model from overfitting.

6.4 CARLA and the radar module

The radar module in CARLA that was used to generate the point clouds has limitations. Therefore, the question of how alike it reassembles an actual FMCW-radar that is used in a real life setting is relevant. Some of the major points that makes the readings from CARLA less realistic are:

- No noise in the measurements
- Only yields measurements from edges
- Cannot go through window panes and other penetrable objects

These are key behaviours that needs to be addressed in the CARLA simulations if it is to be used in a domain adaptation kind of setting. This measuring behaviour also possibly skews the results in a way that makes the data easier to work with since it is more deterministic than data containing a lot of noise.

However, the fact that the measurements are only registered from the edges might restrain the ability to track from this data since there does not exist any information concerning the extent of the objects. When having the ability to receive data points uniformly over the objects extent it also gives the network a clue of the size differences.

6.5 Future work

The MET3v2 network have in this work shown potential in the radar tracking domain but there are ways to explore its capabilities further in possible future work.

6.5.1 Attention

The cross-attention between the decoder object queries and encoder embeddings suggest that that the network has learned to create object queries that inquire about what point-measurements are most likely to be objects. The network usually finds these correlations at the last instances of the sequence but they are sometimes also shifted to earlier measurements as well. Of further note is also that the attention is a vertical line suggesting that a single attention head finds correlation between all (or most) object queries and one/few measurements. This behaviour gives rise to questions:

- Does the network mainly attend to the last measurements, and if true could the network be trained on even fewer time steps?
- Can the network training be improved such that object queries inquire more generally, attending better to object extents and finding correlations between measurements. In simpler terms what changes would be necessary to make the network attend between measurements like illustrated in 5.

To change the way the network attends means to change the way it is penalized in loss functions and perhaps also change how and what it predicts. In the MET3v2 and MT3v2 network the object queries are essentially encoder embeddings that have been projected and transformed by a single FFN and into "projected embeddings" as illustrated in Figure 13. The projected embeddings are essentially derived from measurements, though transformed multiple times, and thus the question arises if the object queries should be something else.

6.5.2 Modifying the loss function

The implementation of a CIoU was done in hopes of adapting it to take into account rotation and more importantly modifying it to allow for direct bounding box prediction

onto the $[x, y]$-plane. After some considerable thought and time modifying the CIoU loss function it is deemed simpler and perhaps less computationally expensive to simple employ a Mean Squared Error loss function for our purposes testing similar sized cars or even Relative Error loss function when dealing with larger variations in vehicle sizes.

$$\text{Relative Error} = \frac{1}{2}\left[\left|\frac{l - l_{gt}}{l_{gt}}\right| + \left|\frac{w - w_{gt}}{w_{gt}}\right|\right] \tag{6.1}$$

It might however also be possible to do away with the entire Bounding Box predictor and loss function altogether, simply expanding the original Pos&Vel Predictor (State Predictor) head to predict the height and width of each object query. This way the model head will predict the entire state in one MLP and perhaps find better association between all the states (position, velocity and extent) within predicted objects. The Hungarian Matching algorithm and state loss would be extended to incorporate object extent.

Extended objects

To properly manage extended targets, insofar as attention goes, one might need to revise or add a loss function such that object identities are maintained and attended to properly in training. As of now the network predicts N_q object queries which are suitable guesses for objects and as can be seen in the figure 3 of attention heads. The focuses between encoder embeddings and decoder object queries however to mainly lie at the end of the sequence.

Though the network proves it can handle object detection the network seems to not utilize all of the measurements for its prediction. Thus finding a way to incorporate identities into the training sequence and making use of them in a proper loss functions might be helpful.

6.5.3 Tracking Identities

The current architecture demonstrates the ability to track objects within a specific frame of reference, corresponding to the input data spanning a certain time. However, to maintain object identities consistently over longer periods and across consecutive time steps, modifications to the network architecture would be required. Specifically, this would involve incorporating an auto-regressive component into the model, accounting for object identities across time, ensuring that each object is consistently identified and tracked throughout the sequence. Identities predicted would need to influence the predictions in subsequent steps. This would allow the network to maintain continuity in object tracking.

While these modifications would enhance the network's ability to track objects consistently, they might increase the complexity of the model and potentially increase inference times. For applications focused solely on obstacle avoidance, especially for the domains which employ radar sensors, the precise tracking of individual object identities may be less critical and this added complexity might be unnecessary.

There already exists several implementations to account for tracking identities throughout multiple time-frames. Facebook AI Research teams proposed the Trackformer [8], building on their previous DETR architecture [9] which manages to solve this problem in an end-to-end manner. It solves this problem by also including track queries

that are propagated across frames and used regressively. These queries maintain object identities over time by associating detected objects in the current frame with those in previous frames. Incorporating these structures to the current MET3v2 network might be a viable option, since it shares many structures with DETR, if maintaining identities regressively is deemed necessary.

6.5.4 Training on higher resolution

Training on higher resolutions yields more information which as so far been sparse and in general might yield even better results. However the direct correlation between more data and physical radar size might make this impractical. Because of limited resources this work does not include training on denser radar clouds, where using the gating approach modality is expected to perform better. This would require the network to be bigger which in turn takes longer to train.

6.5.5 Tracking 3D-objects

Although the detections in this work were captured in the 3D space, the network only predicted the 2D bounding box. Tracking in the 3D space gives rise to a multitude of new challenges but also further use for the network. Extending the network to a 3D tracker will require some alterations similar to the ones that have been done in this work.

6.5.6 Deployment on real data

The end goal of the network is to be able to be handle and yield accurate predictions from radar data that have been collected in reality. As briefly mentioned in 6.4, this entails several new challenges that have not been relevant to investigate in this work working with data stemming from CARLA.

Measurement noise

The uncertainty of the data is increased with the measurement noise. The noise from real data that is not present in the CARLA software introduces an uncertainty in the data and adds to the complexity of the problem. How the network would handle this is hard to say but it would most probably make the associations from measurement to object more difficult.

SNR & RCS

An idea of using the SNR in the selection process to alter the scoring of the points that was received was discussed but not implemented since the handling of the SNR in CARLA was deemed to simplistic and unrealistic to be able to be used effectively in the network implementation. The SNR is not deemed to be robust in most radars however so the probability of successfully utilizing this is rather low.

RCS is a direct property of a targets reflectivity and is different between various objects. It can indeed give a hint as to what type of reflection surface it has. Metal objects are naturally more reflective than other materials such as concrete. Which might aid when doing the first assessment of which measurements are interesting in a tracking setting.

6.6 Sustainability and ethics

There are two main concerns regarding sustainability and ethics when dealing with DL for radar detections. The environmental impact of AI usage and the possible misuse of the tracker.

Depending on the problem one wants to solve and the magnitude of it, it can take a lot of power both in training networks and also in usage, this leads to an increase in power consumption. Recent reports have shown that the energy consumption connected to AI has increased steadily over time and that these effects needs to be taken account for in future legislation to make AI usage more sustainable [43],[44],[45].

Relying solely on DL solutions and consider them to be robust may sometimes depending on the use case cause harm and in some situations even be fatal. Recent reports have shown different kinds of DL solutions that shows serious flaws if not addressed properly and those that have failed, while most of the failures are harmless and does not cause issues other failures can lead to serious injury [46],[47]. Developing and using AI with this in mind is crucial to minimize the possible societal impacts that it may entail. For this work which touches both tracking and ADAS it is important to make sure that the system is not used in ways that can cause harm to the user.

7

Conclusion

From the results in chapter 5, we can conclude that the MET3v2 network and the DL approach to radar tracking are comparable and even outperforms the GGIW-PMBM SOTA tracking method that was used as a benchmark when evaluated with a GOSPA metric. The evaluation were set in situations where the data were complex with many different kinds of objects present with non-linear velocity profiles. This suggests that an approach that builds on DL when working with MEOT problems for radar detections are of interest to find a solution for optimal tracking performance. Although the usage of DL tracking solutions in radar applications in actual real-life use cases are limited, this work gives implications that when wanting to track extended objects in cluttered environments, the new SOTA methodology could be model free in a near future.

The MT3v2 transformer architecture serves as a good base for learning from dynamic stochastic data such as the one generated by CARLA simulations. The data can be considered complex with lots of clutter, dynamic motion, extended targets and stochastic behaviour and can to some extent be considered alike data that are taken from real life in an urban environment. During training, the network managed to increase its performance over the course of time when periodically evaluated on a evaluation dataset. This implicates that the network is able to be trained on complex data and potentially could be trained with real life data and get deployed in a commercial system as a tracker.

With this said, the reality is often way more complex than what any simulation tool can produce. In accordance with what can be seen in other AI domains, a fool proof AI solution is almost an impossibility since the amount of edge cases that can occur in any situation is infinite and impossible to account for while creating a suitable dataset for network training and evaluation.

When assessing the networks ability to perform MEOT tasks compared to the GGIW-PMBM filter it shows great performance when evaluating with a GOSPA measurement. When analyzing the tracking behaviour it becomes apparent that the clutter that the environment produces in combination with unpredicted births, deaths and dynamical movement profiles makes it very hard to tune the GGIW-PMBM filter used to match the performance of the network. One of the major strengths of the model free approach had that the filter had issues with was its ability to discern clutter from actual measurements which simplifies the problem immensely. With the resulting low inference times with using the DL approach less computational power there are also a economical benefit in not needing expensive components in order to predict in real-time making it widely accessible.

A drawback with using the network as a tracker is the amount of data needed for it to become efficient and not overfitted, as with all kinds of ML tasks. This leads to a shift in the work methodology which leads to the major part of the work is in the data labeling and annotation instead of tuning the filters to its optimal performance.

This work shows that the end-to-end DL approach for MEOT for radar applications yields promising results in a scenario that is closer to a domain adaption than previously tested. After some alterations to the architecture of the existing network MT3v2, resulting in the new network MET3v2, results shows that it can handle the MEOT tasks provided well.

Bibliography

[1] Kemiao Huang and Qi Hao. *Joint Multi-Object Detection and Tracking with Camera-LiDAR Fusion for Autonomous Driving*. 2021. arXiv: 2108.04602 [cs.CV]. URL: https://arxiv.org/abs/2108.04602.

[2] Giacomo D'Amicantonio, Egor Bondarau, and Peter H. N. de With. *uTRAND: Unsupervised Anomaly Detection in Traffic Trajectories*. 2024. arXiv: 2404.12712 [cs.CV]. URL: https://arxiv.org/abs/2404.12712.

[3] Jasmine Khairunissa et al. "Detecting Poultry Movement for Poultry Behavioral Analysis using The Multi-Object Tracking (MOT) Algorithm". In: *2021 8th International Conference on Computer and Communication Engineering (ICCCE)*. 2021, pp. 265–268. DOI: 10.1109/ICCCE50029.2021.9467144.

[4] Su Pang and Hayder Radha. *Multi-Object Tracking using Poisson Multi-Bernoulli Mixture Filtering for Autonomous Vehicles*. 2021. arXiv: 2103.07783 [cs.RO]. URL: https://arxiv.org/abs/2103.07783.

[5] Juliano Pinto et al. *Can Deep Learning be Applied to Model-Based Multi-Object Tracking?* 2022. arXiv: 2202.07909 [cs.LG]. URL: https://arxiv.org/abs/2202.07909.

[6] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html (visited on 02/02/2024).

[7] Colin Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2023. arXiv: 1910.10683 [cs.LG]. URL: https://arxiv.org/abs/1910.10683.

[8] Tim Meinhardt et al. *TrackFormer: Multi-Object Tracking with Transformers*. 2022. arXiv: 2101.02702 [cs.CV]. URL: https://arxiv.org/abs/2101.02702.

[9] Nicolas Carion et al. *End-to-End Object Detection with Transformers*. 2020. arXiv: 2005.12872 [cs.CV]. URL: https://arxiv.org/abs/2005.12872.

[10] Merrill I. Skolnik, ed. *Radar Handbook*. en. 3rd Edition. McGraw-Hill Education, 2008. ISBN: 978-0-07-148547-0. URL: https://www.accessengineeringlibrary.com/content/book/9780071485470 (visited on 08/22/2024).

[11] Dipl-Ing (FH) Christian Wolff. *Radartutorial*. en. Publisher: Dipl.-Ing. (FH) Christian Wolff. URL: https://www.radartutorial.eu/02.basics/Frequency%20Modulated%20Continuous%20Wave%20Radar.en.html (visited on 08/22/2024).

[12] Ziqiang Tong Ziqiang Tong, R. Reuter, and M. Fujimoto. "Fast chirp FMCW Radar in automotive applications". en. In: *IET International Radar Conference 2015*. Hangzhou, China: Institution of Engineering and Technology, 2015, pp. 7 .–7 . ISBN: 978-1-78561-038-7. DOI: 10.1049/cp.2015.1362. URL: https://digital-

library.theiet.org/content/conferences/10.1049/cp.2015.1362 (visited on 08/22/2024).

[13] Peter Nimac et al. "Pedestrian Traffic Light Control with Crosswalk FMCW Radar and Group Tracking Algorithm". en. In: *Sensors* 22.5 (Jan. 2022). Number: 5 Publisher: Multidisciplinary Digital Publishing Institute, p. 1754. ISSN: 1424-8220. DOI: 10.3390/s22051754. URL: https://www.mdpi.com/1424-8220/22/5/1754 (visited on 08/22/2024).

[14] Shahzad Ahmed, Junbyung Park, and Sung Ho Cho. "FMCW Radar Sensor Based Human Activity Recognition using Deep Learning". In: *2022 International Conference on Electronics, Information, and Communication (ICEIC)*. ISSN: 2767-7699. Feb. 2022, pp. 1–5. DOI: 10.1109/ICEIC54506.2022.9748776. URL: https://ieeexplore.ieee.org/document/9748776/?arnumber=9748776 (visited on 08/22/2024).

[15] S. Haykin et al. "Classification of radar clutter in an air traffic control environment". In: *Proceedings of the IEEE* 79.6 (June 1991). Conference Name: Proceedings of the IEEE, pp. 742–772. ISSN: 1558-2256. DOI: 10.1109/5.90155. URL: https://ieeexplore.ieee.org/document/90155/?arnumber=90155 (visited on 08/22/2024).

[16] Pia Addabbo et al. "Learning Strategies for Radar Clutter Classification". In: *IEEE Transactions on Signal Processing* 69 (2021). Conference Name: IEEE Transactions on Signal Processing, pp. 1070–1082. ISSN: 1941-0476. DOI: 10.1109/TSP.2021.3050985. URL: https://ieeexplore.ieee.org/document/9321174/?arnumber=9321174 (visited on 08/22/2024).

[17] Jihong Shen et al. "Evaluation of Unscented Kalman Filter and Extended Kalman Filter for Radar Tracking Data Filtering". In: *2014 European Modelling Symposium*. Oct. 2014, pp. 190–194. DOI: 10.1109/EMS.2014.49. URL: https://ieeexplore.ieee.org/document/7153997/?arnumber=7153997 (visited on 08/22/2024).

[18] Karl Granstrom, Marcus Baum, and Stephan Reuter. *Extended Object Tracking: Introduction, Overview and Applications*. en. arXiv:1604.00970 [cs, eess]. Feb. 2017. URL: http://arxiv.org/abs/1604.00970 (visited on 08/22/2024).

[19] Abu Sajana Rahmathullah, Ángel F. García-Fernández, and Lennart Svensson. "Generalized optimal sub-pattern assignment metric". en. In: *2017 20th International Conference on Information Fusion (Fusion)*. arXiv:1601.05585 [cs]. July 2017, pp. 1–8. DOI: 10.23919/ICIF.2017.8009645. URL: http://arxiv.org/abs/1601.05585 (visited on 08/22/2024).

[20] Shishan Yang, Marcus Baum, and Karl Granström. "Metrics for performance evaluation of elliptic extended object tracking methods". In: *2016 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*. Sept. 2016, pp. 523–528. DOI: 10.1109/MFI.2016.7849541. URL: https://ieeexplore.ieee.org/document/7849541/?arnumber=7849541 (visited on 08/22/2024).

[21] Branko Ristic, Sanjeev Arulampalam, and Neil Gordon. *Beyond the Kalman Filter: Particle Filters for Tracking Applications*. en. Artech House, Dec. 2003. ISBN: 978-1-58053-851-0.

[22] Ángel F. García-Fernández et al. "Poisson multi-Bernoulli mixture filter: direct derivation and implementation". en. In: *IEEE Transactions on Aerospace and Electronic Systems* 54.4 (Aug. 2018). arXiv:1703.04264 [cs, stat], pp. 1883–1901. ISSN: 0018-9251, 1557-9603, 2371-9877. DOI: 10.1109/TAES.2018.2805153. URL: http://arxiv.org/abs/1703.04264 (visited on 08/23/2024).

[23] Nicolas Taba. *Understanding the Role of Modern Technology in Environmental Sustainability*. https://www.ida.liu.se/~732A64/info/BetygA/NicolasTaba.pdf. Accessed: 2024-08-22. 2023.

[24] Audun G. Hem, Martin Baerveldt, and Edmund F. Brekke. "PMBM Filtering With Fusion of Target-Provided and Exteroceptive Measurements: Applications to Maritime Point and Extended Object Tracking". In: *IEEE Access* 12 (2024). Conference Name: IEEE Access, pp. 55404–55423. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2024.3389824. URL: https://ieeexplore.ieee.org/document/10500836/?arnumber=10500836 (visited on 08/23/2024).

[25] Xingxiang Xie et al. "PMBM Filter for Multiple Extended Targets With Unknown Clutter Rate and Detection Probability". In: *IEEE Sensors Journal* 23.15 (Aug. 2023). Conference Name: IEEE Sensors Journal, pp. 17133–17147. ISSN: 1558-1748. DOI: 10.1109/JSEN.2023.3285885. URL: https://ieeexplore.ieee.org/document/10154588/?arnumber=10154588 (visited on 08/23/2024).

[26] Sen Wang, Qinglong Bao, and Jiameng Pan. "Multi-Bernoulli Mixture Filter: Complete Derivation and Sequential Monte Carlo Implementation". en. In: *2021 14th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*. Shanghai, China: IEEE, Oct. 2021, pp. 1–5. ISBN: 978-1-66540-004-6. DOI: 10.1109/CISP-BMEI53629.2021.9624421. URL: https://ieeexplore.ieee.org/document/9624421/ (visited on 08/23/2024).

[27] Karl Granstrom, Maryam Fatemi, and Lennart Svensson. "Poisson multi-Bernoulli conjugate prior for multiple extended object filtering". en. In: *IEEE Transactions on Aerospace and Electronic Systems* 56.1 (Feb. 2020). arXiv:1605.06311 [cs, stat], pp. 208–225. ISSN: 0018-9251, 1557-9603, 2371-9877. DOI: 10.1109/TAES.2019.2920220. URL: http://arxiv.org/abs/1605.06311 (visited on 08/23/2024).

[28] Seung-Hwan Bae et al. "Automated Multi-target Tracking with Kinematic and Non-Kinematic Information". In: *IET Radar, Sonar and Navigation* (Apr. 2012).

[29] Yuxuan Xia et al. *Poisson Multi-Bernoulli Approximations for Multiple Extended Object Filtering*. en. arXiv:1801.01353 [eess]. Aug. 2021. URL: http://arxiv.org/abs/1801.01353 (visited on 08/23/2024).

[30] Michael Beard et al. "Multiple Extended Target Tracking with Labelled Random Finite Sets". en. In: *IEEE Transactions on Signal Processing* 64.7 (Apr. 2016). arXiv:1507.07392 [stat], pp. 1638–1653. ISSN: 1053-587X, 1941-0476. DOI: 10.1109/TSP.2015.2505683. URL: http://arxiv.org/abs/1507.07392 (visited on 08/23/2024).

[31] Xingjing Mao et al. *Toward Fact-aware Abstractive Summarization Method Using Joint Learning*. Oct. 2022. DOI: 10.21203/rs.3.rs-2206382/v1.

[32] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG]. URL: https://arxiv.org/abs/1502.03167.

[33] Ch Murthy et al. "Investigations of Object Detection in Images/Videos Using Various Deep Learning Techniques and Embedded Platforms—A Comprehensive Review". In: *Applied Sciences* (May 2020). DOI: 10.3390/app10093280.

[34] Georg, William Hess, and Ljungbergh. *Transforming the Field of Multi-object Tracking*. 2021. URL: https://odr.chalmers.se/server/api/core/bitstreams/734cc4c1-7892-4b7b-9f0b-4ac604348061/content (visited on 07/29/2024).

[35] Xizhou Zhu et al. *Deformable DETR: Deformable Transformers for End-to-End Object Detection*. 2021. arXiv: 2010.04159 [cs.CV]. URL: https://arxiv.org/abs/2010.04159.

[36] Alexey Dosovitskiy et al. *CARLA: An Open Urban Driving Simulator*. 2017. arXiv: 1711.03938 [cs.LG]. URL: https://arxiv.org/abs/1711.03938.

[37] Jean-Emmanuel Deschaud. *KITTI-CARLA: a KITTI-like dataset generated by CARLA Simulator*. 2021. arXiv: 2109.00892 [cs.CV]. URL: https://arxiv.org/abs/2109.00892.

[38] Rodrigo Gutiérrez-Moreno et al. "Reinforcement Learning-Based Autonomous Driving at Intersections in CARLA Simulator". In: *Sensors* 22.21 (2022). ISSN: 1424-8220. DOI: 10.3390/s22218373. URL: https://www.mdpi.com/1424-8220/22/21/8373.

[39] Sebastian Dingler. *State estimation with the Interacting Multiple Model (IMM) method*. en. arXiv:2207.04875 [cs, eess]. July 2022. URL: http://arxiv.org/abs/2207.04875 (visited on 08/23/2024).

[40] X. Rong Li and V.P. Jilkov. "Survey of maneuvering target tracking. Part I. Dynamic models". In: *IEEE Transactions on Aerospace and Electronic Systems* 39.4 (Oct. 2003). Conference Name: IEEE Transactions on Aerospace and Electronic Systems, pp. 1333–1364. ISSN: 1557-9603. DOI: 10.1109/TAES.2003.1261132. URL: https://ieeexplore.ieee.org/document/1261132/?arnumber=1261132 (visited on 08/23/2024).

[41] Xiaohua Zhai et al. *Scaling Vision Transformers*. en. arXiv:2106.04560 [cs]. June 2022. URL: http://arxiv.org/abs/2106.04560 (visited on 08/27/2024).

[42] Jared Kaplan et al. *Scaling Laws for Neural Language Models*. en. arXiv:2001.08361 [cs, stat]. Jan. 2020. URL: http://arxiv.org/abs/2001.08361 (visited on 08/27/2024).

[43] Carole-Jean Wu et al. *Sustainable AI: Environmental Implications, Challenges and Opportunities*. en. arXiv:2111.00364 [cs]. Jan. 2022. URL: http://arxiv.org/abs/2111.00364 (visited on 09/03/2024).

[44] Adrien Berthelot et al. "Estimating the environmental impact of Generative-AI services using an LCA-based methodology". In: *Procedia CIRP*. 31st CIRP Conference on Life Cycle Engineering 122 (Jan. 2024), pp. 707–712. ISSN: 2212-8271. DOI: 10.1016/j.procir.2024.01.098. URL: https://www.sciencedirect.com/science/article/pii/S2212827124001173 (visited on 09/03/2024).

[45] Philipp Hacker. "Sustainable AI Regulation". en. In: *SSRN Electronic Journal* (2023). ISSN: 1556-5068. DOI: 10.2139/ssrn.4467684. URL: https://www.ssrn.com/abstract=4467684 (visited on 09/03/2024).

[46] Luke Oakden-Rayner et al. "Hidden stratification causes clinically meaningful failures in machine learning for medical imaging". en. In: *Proceedings of the ACM Conference on Health, Inference, and Learning*. Toronto Ontario Canada: ACM, Apr. 2020, pp. 151–159. ISBN: 978-1-4503-7046-2. DOI: 10.1145/3368555.3384468. URL: https://dl.acm.org/doi/10.1145/3368555.3384468 (visited on 09/03/2024).

[47] Andrew J. Hawkins. *Tesla's Autopilot and Full Self-Driving under NHTSA investigation after deadly crashes*. Accessed: 2024-08-27. Apr. 2024. URL: https://www.theverge.com/2024/4/26/24141361/tesla-autopilot-fsd-nhtsa-investigation-report-crash-death.

A

Network configurations

Table A.1: ME3v2 model parameters used

$n_{\mathbf{params}}$	$n_{\mathbf{layers}}$	$d_{\mathbf{model}}$	$n_{\mathbf{head}}$	$d_{\mathbf{head}}$	**Batch Size**	**Learning Rate**
≈ 18.7 million	12	256	8	32	2	$2.5 \times 10^{-5} - \ 2.5 \times 10^{-8}$

B

PMBM-GGIW parameters

Table B.1: Parameters for the homogeneous dataset task.

a	**b**	**P**	**v**	V
600	50	[1,1;1,1]	50	600·[1/2,3/4;3/4,1/2]

Table B.2: Parameters for the heterogeneous dataset task, where r denotes a random float between 0 and 1.

a	**b**	**P**	**v**	V
450	50	[1,1;1,1]	50	600·[1/2+r,3/4;3/4,1/2+r]

CHALMERS