Institute of Imaging and Computer Vision | RWTH AACHEN UNIVERSITY

Transformer-Based Robust Multi-Object Tracking using Historical Trajectories

Master Thesis

presented by

Jan Frederik Meier

Supervisor:
Prof. Dr.-Ing. Johannes Stegmaier

Institute of Imaging & Computer Vision
Prof. Dr.-Ing. Johannes Stegmaier
RWTH Aachen University

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Meier Jan Frederik**	Personal ID number:	**516393**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Control Engineering**		
Study program:	**Cybernetics and Robotics**		

II. Master’s thesis details

Master’s thesis title in English:

Transformer-Based Robust Multi-Object Tracking using Historical Trajectories

Master’s thesis title in Czech:

Robustní sledování mnoha objektů pomocí transformeru s využitím historických trajektorií

Guidelines:

1) Literature research on transformer-based end-to-end Multi-Object Tracking
2) Investigate new ways to include temporal-visual cues in transformer-based end-to-end Multi-Object Tracking
3) Investigate new ways to faciliate long-term robust association
4) Evaluate the model on current benchmarks

Bibliography / sources:

[1] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, Sergey Zagoruyko: End-to-End Object Detection with Transformers, https://doi.org/10.48550/arXiv.2005.12872
[2] Tim Meinhardt, Alexander Kirillov, Laura Leal-Taixe, Christoph Feichtenhofer: TrackFormer: Multi-Object Tracking with Transformers TrackFormer,https://doi.org/10.48550/arXiv.2101.02702
[3] Fangao Zeng, Bin Dong, Yuang Zhang, Tiancai Wang, Xiangyu Zhang, Yichen Wei: End-to-End Multiple-Object Tracking with Transformer, https://doi.org/10.48550/arXiv.2105.03247

Name and workplace of master’s thesis supervisor:

Prof.Dr.-Ing. Johannes Stegmaier RWTH AAchen University

Name and workplace of second master’s thesis supervisor or consultant:

Ing. Martin Hlinovský, Ph.D. Department of Control Engineering FEE

Date of master’s thesis assignment: **28.02.2024** Deadline for master's thesis submission: **17.12.2024**

Assignment valid until:
by the end of summer semester 2024/2025

Prof.Dr.-Ing. Johannes Stegmaier	doc. Ing. Zdeněk Hurák, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt	Student’s signature

Erklärung nach §18 Abs. 1 ÜPO

Hiermit versichere ich, dass ich die vorgelegte Master Thesis selbständig angefertigt habe. Es sind keine anderen als die angegebenen Quellen und Hilfsmittel benutzt worden. Zitate wurden kenntlich gemacht.

I hereby confirm that I have written this Master Thesis independently using no sources or aids other than those indicated. I have appropriately declared all citations.

[illegible]

Jan Frederik Meier
Aachen, 17.12.2024

Contents

List of Figures

List of Tables

List of Listings

List of Abbreviations

AssA	Association Accuracy
BN	Batch Normalization
CAL	Collective Averaging Loss
CDN	Contrastive Denoising
CMC	Camera Motion Compensation
CNN	Convolutional Neural Network
COLA	Coopetition Label Assignment
DetA	Detection Accuracy
DETR	Detection Transformer
DN	Denoising
DNN	Deep Neural Network
E2E	End-to-End
FC	Fully Connected
FFN	Feed-Forward Network
FN	False Negative
FNA	False Negative Association
FP	False Positive
FPA	False Positive Association
GC	Gradient Coordination
gIoU	generalized Intersection over Union
GPU	Graphics Processing Unit

HOTA	Higher Order Tracking Accuracy
HSV	Hue-Value-Saturation
IDF$_1$	Identity F$_1$-score
IDFN	Identity False Negative
IDFP	Identity False Positive
IDP	Identity Precision
IDR	Identity Recall
IDSW	Identity Switches
IDTP	Identity True Positive
IoU	Intersection over Union
LFT	Look Forward Twice
LSTM	Long Short-Term Memory
MHA	Multi-Head Attention
MLP	Multi-Layer Perceptron
MOT	Multiple Object Tracking
MOTA	Multiple Object Tracking Accuracy
MOTP	Multiple Object Tracking Precision
MQS	Mixed Query Selection
NMS	Non-maximum Suppression
NN	Neural Network
PLD	Pseudo Label Distillation
PSD	Pseudo Label Distillation
QIM	Query Interaction Module
Re-ID	Re-Identification
ReLU	Rectified Linear Unit
RFS	Release-Fetch-Supervision
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SOT	Single Object Tracking
TAN	Temporal Aggregation Network
TbD	Tracking-by-Detection
TGD	Track Group Denoising
TP	True Positive
TPA	True Positive Association
TPM	Trajectory Prediction Module
VRAM	Video Random Access Memory

1 Introduction

Multiple Object Tracking (MOT) represents a fundamental computer vision task, which consists of locating multiple objects, maintaining their identities, and yielding their individual trajectories within a given video sequence. It is employed for the tracking of a diverse array of objects, including pedestrians [1,2], dancers [3], sport players [4], vehicles [5] and animals [6] (Figure 1.1). Furthermore, Multiple object tracking also serves as a preprocessing step for other high-level tasks such as pose estimation [7] and action recognition [8] in video data.

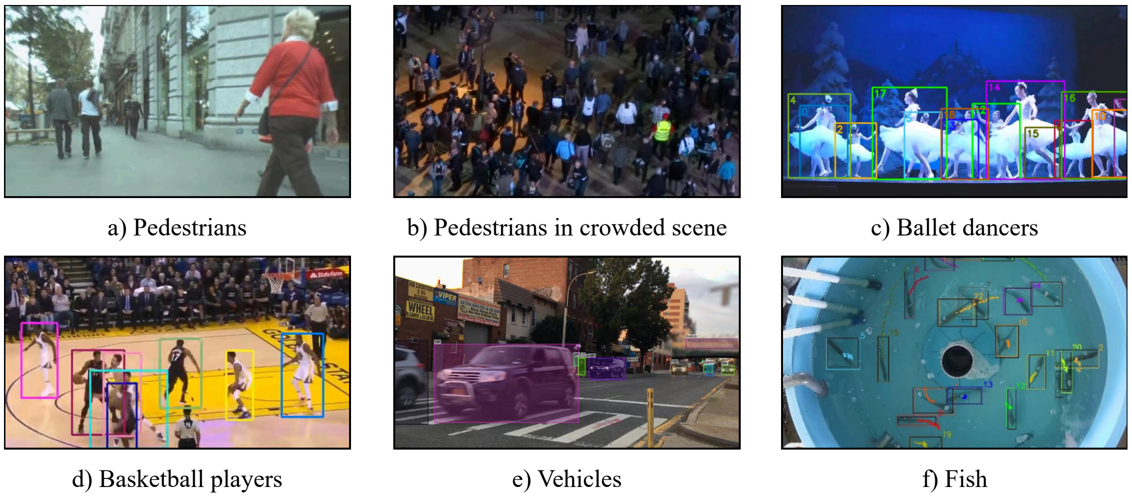


Figure 1.1: Exemplary tracking targets for Multiple Object Tracking. a),b) show the tracking of pedestrians, which is one of the most common targets [1,2]. In c), ballet dancers are tracked during a performance [3]. d) features the tracking of basketball players during a match, which can be used to calculate player metrics [4]. e) contains tracked vehicles in a common traffic scenario [5]. Vehicle tracking can be utilized in autonomous driving applications. In f) an example of animal tracking is given [9].

The foundation for all of these applications is a robust Multiple Object Tracking model. At present, the Tracking-by-Detection (TbD) paradigm represents the most prevalent Multiple Object Tracking approach. In this two-stage approach, objects are first detected in each frame and then associated to form trajectories [10]. Although existing object detection models lead to a good detection performance, Tracking-by-Detection (TbD) models encounter difficulties in establishing robust associations in sequences characterized by complex motion and occlusion [3]. New End-to-End approaches based on Detection Transformer (DETR) [11] are capable of performing well in such cases. They are iteratively updating tracked object representations on a frame-to-frame basis to handle changing motion and appearance [12, 13]. End-to-End models are well suited for scenarios with uniform appearances and complex motion patterns, such as dance performances [3] or animal tracking [9].

Nevertheless, the current End-to-End (E2E) approaches are confronted with a plethora of challenges. As illustrated in Table 1.1, these models have prolonged training times and require a considerable amount of Video Random Access Memory (VRAM), which constrains their practical applications. Furthermore, numerous models incorporate intricate training schedules comprising separate pretraining phases [12, 14] [TrackFormer, DN-MOT] and the incorporation of supplementary training data, a consequence of the data-intensive nature of transformer models [12–15].

Table 1.1: Training Time of different End-to-End Multiple Object Tracking models. Despite the utilization of multiple high-end GPUs, the training process for a small dataset, such as MOT17, spans multiple days.

Model	GPU	Training Time
TrackFormer	$7 \times$ 32 GB GPUs	2 days
MOTR	$8 \times$ NVIDIA V100	2.5 days

The objective of this thesis is to design a lightweight End-to-End Multiple Object Tracking model with a simple training routine. We refrain from utilizing supplementary data during the training phase and instead employ generic initialization weights in lieu of task-specific pretrained models. To further capitalize on the robust association performance observed in E2E models, we are integrating a motion prediction module based on historical trajectory data. In the initial chapter, we present a comprehensive overview of existing literature on the subject, commencing with an examination of the Multiple Object Tracking (MOT) task in its entirety, followed by an analysis of its associated metrics, an investigation of the various deep learning architectures employed, an exploration of the different Detection Transformers, and a detailed account of the existing E2E MOT models. In the Methods chapter, we present our own architecture. Subsequently, we discuss the experimental setup and implementation details. We then compare our results with those of other trackers on the MOT17 [1], DanceTrack [3], and SportsMOT [4] datasets. Finally, we summarize our results and provide possible directions for future research.

2 Related Works

This chapter discusses previous work relevant to this thesis. First, the MOT task itself is considered. For this purpose, the various approaches and metrics are analyzed. Next, a brief introduction to Convolutional Neural Networks (CNNs) is given. The focus will be on the ResNet [16] architecture. Subsequently, the Transformer [17] architecture is explained in detail. Based on this, the DETR [11] and modifications of it are explained. Finally, various E2E MOT models are analysed in detail.

2.1 Multiple Object Tracking

Multiple Object Tracking is a basic computer vision task that aims to detect and track objects (of one or more classes) over a video sequence. The task is closely related to object detection, as it also returns a bounding box for each detected object, while additionally assigning an ID to each detection. This ID allows objects of the same class to be distinguished and tracked over several frames. In Single Object Tracking (SOT), the appearance of the object to be tracked is a priori knowledge. In MOT this prior information about the number of objects and their appearance is missing, therefore a detection step is necessary [18].

The main challenge in MOT is occlusion, which refers to the situation where an object is partially or fully covered by another object in the same frame. The effect is more severe in crowded scenes. Inaccurate detections present an additional challenge, as they result in significant tracking difficulties. Detection is made even more difficult by changing backgrounds, lighting and motion blur [19].

2.1.1 Approaches

Most Multiple Object Tracking algorithms can be divided into two different approaches. TbD is currently the predominant approach and is based on the separation of detection and association. In the first step, the objects are detected in the frame and then assigned an ID in the second step. E2E MOT, on the other hand, is a new approach and does not separate detection and association. Both tasks are performed in the same step. The two approaches are discussed in more detail below.

Tracking by Detection

In Tracking-by-Detection, the targets in the frame are detected first. The boxes are associated by assigning IDs over several frames. For this reason, the problem is also described as an assignment problem. Due to the rapid progress in object detection, currently strong and fast object detection models are employed in the detection stage. As a result, most TbD algorithms only try to improve the association. This is also reflected in the MOT datasets, which often contain a standard set of detections. Due to this, the association performance of different algorithms can be compared independently of the detection performance [18]. TbD approaches can be classified according to the type of information used for association.

Motion-based Motion-based methods are using motion models to predict future locations of the tracks and then apply a heuristic to match the predictions with the detection. SORT [20] uses a Kalman filter as a linear motion model and then matches based on Intersection over Union (IoU) using one-to-one matching. Other approaches are using Camera Motion Compensation (CMC) in addition to handle cases with camera motion [21, 22].

Appearance-based Other models are focusing on appearance features to associate detection. The authors in [23] propose to extract features from detection patches for Re-Identification (Re-ID). Based on similarity between features, new detections are then assigned to existing tracks using one-to-one matching. These approaches are robust to motion and occlusion, as the extracted appearance features are temporally stable [24].

End-to-End Tracking

End-to-End Multiple Object Tracking, also known as Tracking-by-Query [25] or Tracking-by-Attention [12] is a new approach based on the DETR [11]. The DETR architecture makes it possible to perform detection and association in one step. DETR detects objects by processing ‘object queries’ in the decoder. In E2E MOT, additional ‘track queries’ are added to the object queries, each of which tracks a specific object. These queries are then passed from one time step to the next and thus track the same object without separate association [25]. DETR, as well as the models inspired from it, are described in chapters section 2.4 and section 2.5 in detail.

2.1.2 Metrics

In order to evaluate a model and enable comparability between different models, a group of metrics must be defined. However, unlike other tasks, MOT is difficult to evaluate. While the performance of classification and object detection is comparatively easy to describe with a few metrics, MOT is about balancing the importance of detection and association. As a result, there are many different metrics. At the beginning of this section the classical MOT metrics [26] are considered. Then the CLEAR [27] and Identity [28] metrics will be discussed. Finally, the Higher Order Tracking Accuracy (HOTA) [29] metrics are examined.

Classical MOT Metrics

These metrics were introduced by [26] to quantitatively evaluate the performance of their algorithm. They were designed to cover most typical errors which were observed during their experiments. A visualization of the different metrics is given in Figure 2.1.

- **Mostly Tracked (MT) trajectories:** More than 80% of the trajectory is tracked
- **Fragments:** Less than 80% of a ground truth trajectory
- **Mostly Lost (ML) trajectories:** More than 80% of the trajectory is lost
- **False trajectories:** A trajectory corresponding to no real object
- **ID switches:** Identity Switches (IDSW) between two trajectories

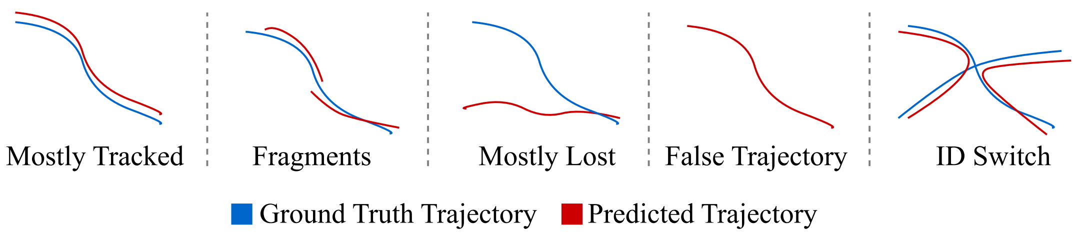


Figure 2.1: Visualization of the classical MOT metrics. Blue denotes the ground-truth trajectory and red the predicted ones (adapted from [26]). (for the final figure false alarm will be left out and all cases will be separated by vertical lines)

CLEAR MOT Metrics

The CLEAR MOT metrics [27] were developed to enable an objective comparison between different multi-object trackers. The focus is on the precision of the object localization, the accuracy of the configuration recognition and the consistency of the IDs over time. To achieve this, two metrics were introduced: Multiple Object Tracking Accuracy (MOTA) and Multiple Object Tracking Precision (MOTP). These metrics are each made up of other simpler metrics. MOTA & MOTP are designed so that they can be applied 2D as well as 3D MOT problems with a single or multiple cameras. In this thesis, only the 2-dimensional case with a single camera is considered [27].

To determine the performance of multi-object trackers, correspondences between the predictions and the ground-truth must first be established. Afterwards, the simple metrics, which MOTA and MOTP are summarizing, can be calculated. The procedure for this is described below.

Let $\{\hat{y}_1, ..., \hat{y}_m\}$ be the prediction of the tracker in each frame and $\{y_1, ..., y_n\}$ the real object. Let $M_t = (y_i, \hat{y}_j)$ be the set of correspondences made up to time t and initialize M_t with $\emptyset$.

1. Check every correspondence in M_{t-1}. If the corresponding object still exists in frame t and if their IoU is above a certain threshold τ, keep the correspondence for frame t.

2. Check all unmatched objects and try to establish a correspondence whose IoU is higher than τ. The matching should minimize the total object-hypothesis error. Since only one-to-one matches are allowed, the problem can be solved with the Hungarian Algorithm [30]. If a correspondence $(y_i, \hat{y}_k)$ is established that contradicts a correspondence $(y_i, \hat{y}_j)$ from the previous frame $t-1$, replace $(y_i, \hat{y}_j)$ with $(y_i, \hat{y}_k)$. This is counted as a mismatch error. The number of mismatch errors in frame t is mme_t.

3. c_t is the number of matches in frame t. Calculate the distance d_t^i between the object and its corresponding prediction for each match.

4. All remaining predictions are considered false positives and all remaining objects misses. Let fp_t be the number of false positives, m_t the number of misses and g_t the number of objects present in frame t.

5. Repeat the procedure from step 1 for the next frame.

Based on the number of misses m_t, false positives fp_t and mismatches mme_t for frame t, MOTA and MOTP can be calculated.

Multiple Object Tracking Accuracy MOTA can be derived from the error rate E_{tot} which is the sum of the false positive ratio, the miss ratio and the mismatch ratio. The formula is given in Equation 2.1. It can be seen as an intuitive measure of the tracker's performance at detecting objects and keeping their trajectories, independent of the precision with which the object locations are estimated [27].

$$MOTA = 1 - \frac{\sum_t (m_t + fp_t + mme_t)}{\sum_t g_t} \tag{2.1}$$

Multiple Object Tracking Precision MOTP is the localization error for each match, averaged by the total number of matches. It reflects the tracker's ability to precisely estimate the object position independent of its association performance and other aspects. The calculation is depicted in Equation 2.2 [27].

$$MOTP = 1 - \frac{\sum_{t,i} d_t^i}{\sum_t c_t} \tag{2.2}$$

Identity Metrics

Existing performance metrics, such as the CLEAR metrics, indicate the frequency with which a tracker makes erroneous decisions and the types of them. However, in certain instances, the user may be more interested in rewarding a tracker that is able to follow an object for the longest possible duration. Therefore, the Identity metrics [28] are trying to measure performance not by how often mismatches occur, but by how long the tracker correctly identifies targets.

Analogous to the calculation of the CLEAR metrics, ground-truth and predictions must first be matched. However, to calculate the Identity metrics, the predicted trajectories are matched with the ground-truth trajectories via one-to-one matching rather than frame-wise matching. Standard metrics such as Precision, Recall and F_1-score are then calculated based on this matching. The matching is calculated such that the number of False Negative (FN) and False Positive (FP) is minimized. Then Identity False Negative (IDFN), Identity False Positive (IDFP) and Identity True Positive (IDTP) can be determined. These metrics are the extension of FN, FP and True Positive (TP) for matched trajectories. Based on them, it is possible to compute Identity Precision (IDP), Identity Recall (IDR), and the corresponding Identity F_1-score (IDF_1) according to Equation 2.3 [28].

$$\begin{aligned} IDP &= \frac{IDTP}{IDTP + IDFP} \\ IDR &= \frac{IDTP}{IDTP + IDFN} \\ IDF_1 &= \frac{2 \cdot IDTP}{2 \cdot IDTP + IDFP + IDFN} \end{aligned} \tag{2.3}$$

IDP and IDR are the proportion of correctly identified detections out of all predicted or ground-truth detections. While IDP and IDR highlight the trade-offs in tracking, the IDF_1 score provides a means of ranking trackers by calculating their harmonic mean [28].

Higher Order Tracking Accuracy

Previous MOT metrics overestimate the importance of detection or association, which can be seen in Figure 2.2. MOTA overemphasizes detection and IDF_1 association performance. To solve this, HOTA [29] explicitly balances both aspects into one single unified metric, while also taking the localization accuracy into consideration. The HOTA metric is a combination of the Detection Accuracy (DetA) and Association Accuracy (AssA), which are each focusing solely on their respective subtask of MOT. These sub-metrics allow a detailed analysis of the tracking performance [29].

Furthermore, the newly introduced metrics are readily comprehensible. DetA is defined as the percentage of aligning detections and AssA as the average alignment between matched trajectories averaged over all detection. HOTA is then calculated by taking the geometric mean of these two scores and averaging it over different localization thresholds. The detailed computation of HOTA and its sub-metrics is explained below [29].

1. **Matching Predictions and Ground-Truth:** Similar to MOTA, matching is performed bijective frame-wise. A TP is a pair (ground-truth detection,

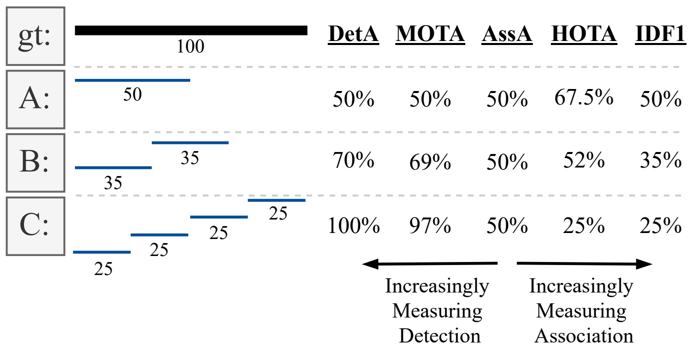


Figure 2.2: Visualization of HOTA metrics. Comparison of HOTA, its sub-metrics DetA and AssA with MOTA and IDF$_1$ for various tracking scenarios. The black line is the ground-truth trajectory and each blue line a predicted trajectory (adapted from [29]).

predicted detection) with a localization similarity $S \geq$ threshold α. A FN is an unmatched ground-truth detection. A FP is an unmatched predicted detection. Since multiple combinations of matches can occur, the matching is performed with the Hungarian algorithm [30] to maximize the HOTA score.

2. **Measuring Association:** To measure the association, the concept of TP, FN and FP is adapted to association. The set of True Positive Association (TPA) is the set of TP for which ground-truth detection and predicted detection have the same ID. For a given TP (referred to as 'c'), the set of False Negative Association (FNA) is the set of ground-truth detections Det$_{gt}$ with the same ID as c, which were either assigned a different or no ID$_{pred}$. At last, for a given c, the set of False Positive Association (FPA) is constituted by the set of Det$_{pred}$ with the same ID$_{pred}$ as c, which were either assigned a different ID$_{gt}$ than c or no ID$_{gt}$. These definitions are visualized in Figure 2.3.

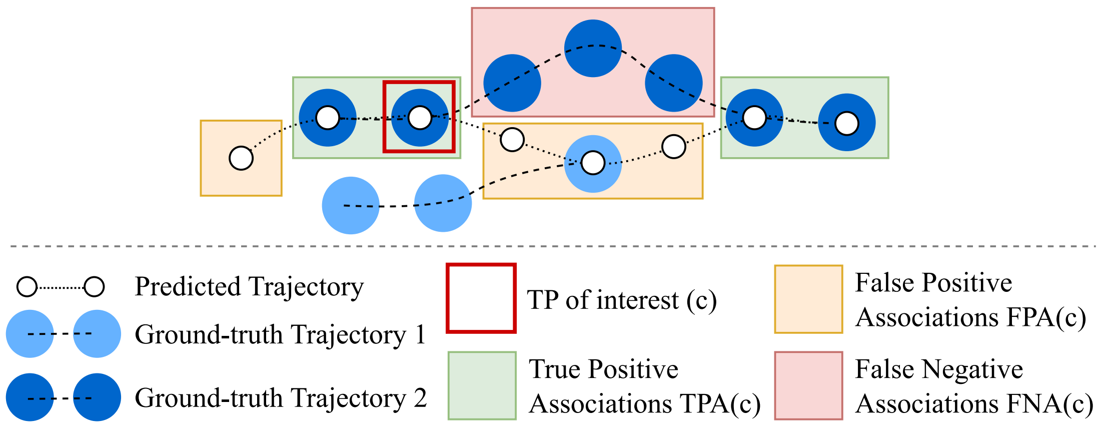

Predicted Trajectory
Ground-truth Trajectory 1
Ground-truth Trajectory 2
TP of interest (c)
True Positive Associations TPA(c)
False Positive Associations FPA(c)
False Negative Associations FNA(c)

Figure 2.3: Overview of different association concepts. TPA, FPA and FNA extend the concept of TP, FP and FN by including the trajectory ID. In this figure they are highlighted for a specific $c \in TP$ (adapted from [29]).

3. **Scoring Function:** Based on the previously defined metrics, it is now possible to calculate the HOTA$_\alpha$ score for a given localization threshold α. The equation is given in Equation 2.4. Even though A, TP,FN, etc. depend on α, α is left out of the equation for visual clarity.

$$\begin{aligned} HOTA_\alpha &= \sqrt{\frac{\sum_{c\in\{TP\}} A(c)}{|TP|+|FN|+|FP|}} \\ A(c) &= \frac{|TPA(c)|}{|TPA(c)|+|FNA(c)|+|FPA(c)|} \end{aligned} \tag{2.4}$$

4. **Matching to Optimize HOTA:** The matching in HOTA maximizes the HOTA score. To achieve this, the Hungarian Algorithm is used in combination with a suited scoring function. The primary objective is to maximize the number of TP. The secondary objective is the maximization of the association score and the third objective is maximizing the similarity S. The scoring function between each potential match Det$_{gt}$ i and Det$_{pred}$ j is shown in Equation 2.5.

$$\text{Score}(i,j) = \left\{ \frac{1}{\epsilon} + A_{max}(i,j) + \epsilon \cdot S(i,j) \quad \text{,if } S(i,j) \geq \alpha \right\} \tag{2.5}$$

5. **Integrating over Localization Thresholds:** The HOTA$_\alpha$ score (Equation 2.4) does not take the localization accuracy into consideration. To include the localization, the HOTA$_\alpha$ score is averaged over a distinct set of α values (Equation 2.6). This functions as an approximation of the integral of the HOTA$_\alpha$ score across α.

$$HOTA = \int_0^1 HOTA_\alpha \, d\alpha \approx \frac{1}{19} \sum_{\alpha=0.05}^{0.95} HOTA_\alpha \tag{2.6}$$

6. **Decomposing HOTA into sub-metrics:** The decomposition of HOTA into different sub-metrics has the further advantage that it enables users to select algorithms or tune algorithms' hyper-parameters based on the nuances of their particular use-case. DetA and AssA are measuring detection and accuracy respectively. They can be calculated according to Equation 2.7 [29].

$$\begin{aligned} DetA_\alpha &= \frac{|TP|}{|TP|+|FN|+|FP|} \\ AssA_\alpha &= \frac{1}{|TP|} \sum_{c\in\{TP\}} A(c) \\ HOTA_\alpha &= \sqrt{\frac{\sum_{c\in\{TP\}} A(c)}{|TP|+|FN|+|FP|}} \\ &= \sqrt{AssA_\alpha \cdot DetA_\alpha} \end{aligned} \tag{2.7}$$

It can be seen that HOTA_α is equal to the geometric mean of DetA_α and AssA_α. This ensures that both detection and association performance are evenly balanced in HOTA [29].

2.2 Convolutional Neural Networks

CNNs made impressive achievements in many areas, including computer vision and are in most tasks still state of the art. The term 'convolution' was first used in this context by LeCun et al. [31] in the original version of LeNet, which is a CNN for handwritten zip code recognition. In 2012, CNNs made a significant advancement in computer vision with the release of AlexNet [32], which achieved a new record score in the ImageNet LSVRC-2010 [33] image classification challenge. Since then, numerous variations of CNN models have been developed. A representative sample of these models is presented in Figure 2.4 [34].

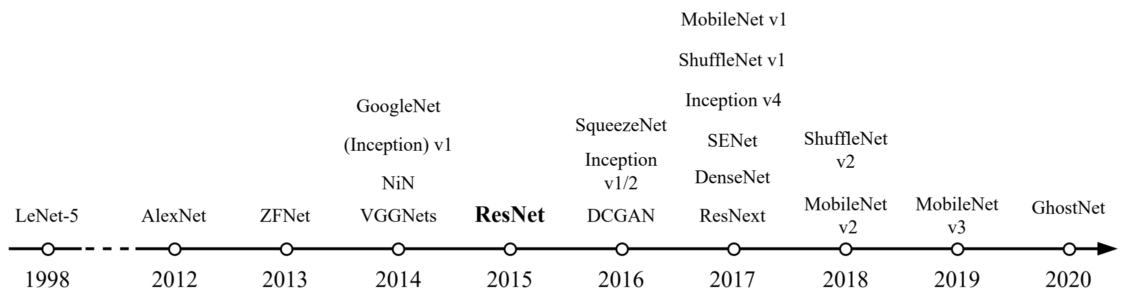


Figure 2.4: Timeline of influential CNNs. This thesis exclusively considers ResNet (adapted from [34]).

This thesis only provides a cursory examination of CNN and focuses on the ResNet [16] architecture, which is employed as a fundamental component of the DETR. The ResNet architecture is presented in the following subchapter. In particular, ResNet-50 is discussed.

2.2.1 ResNet

Deep CNNs have been the source of a number of breakthroughs in various computer vision tasks. The idea is that deep networks naturally extract low-, mid- and high-level features and that the features can be enriched by stacking more layers. When learning Deep Neural Network (DNN) problems like vanishing/exploding gradients which counteract convergence occur. This has however already been addressed by methods such as normalized parameter initialization, Batch Normalization (BN) and Stochastic Gradient Descent (SGD). Another problem is that when the network depth increases, the performance saturates and then degrades rapidly. This degradation is not due to overfitting as stacking more layers also leads to an increased training error. This implies that deeper models are more difficult to optimize. ResNet addresses this degradation problem by introducing residual connections. These connections add the input of a module to its output via identity mapping. A block featuring these residual connections is visualized in Figure 2.5 [16].

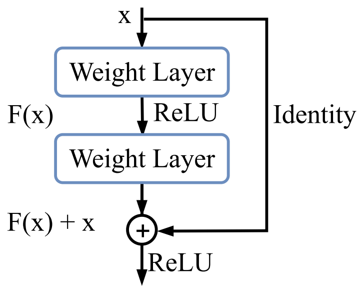


Figure 2.5: Structure of a residual block. A residual connection is added which adds input of the module to the output via identity mapping. This addresses the degradation problem which occurs in CNNs with increasing depth (adapted from [16]).

ResNet-50

The ResNet family comprises five CNNs, each with a distinct depth (ranging from 18 to 152 layers). Given the multifaceted applications of ResNet-50, this subchapter will focus on an in-depth examination of ResNet-50 as a specific case study. The ResNet-50 model is composed primarily of a series of standard blocks, also referred to as 'bottleneck blocks'. These blocks comprise three convolutional layers, each followed by a batch normalization and a Rectified Linear Unit (ReLU). Subsequently, a residual connection is employed to integrate the input of the block with its output. The structural configuration of a bottleneck block is illustrated in Figure 2.6 [16].

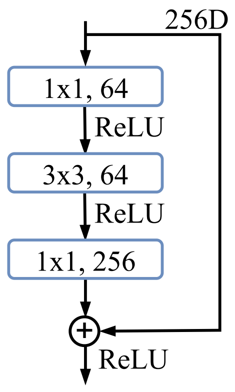


Figure 2.6: Structure of ResNet-50s Bottleneck block. Resnet-50 is build from multiple of those blocks with changing kernel sizes, channel dimensions and strides (adapted from [16]).

The Resnet-50 network is composed of five fundamental building blocks. The Conv1 block constitutes the initial module of the network. The block comprises a convolutional layer, a BN layer, a ReLU, and a max-pooling layer. The subsequent building blocks, Conv2 to Conv5, are simply repeated bottleneck blocks with distinct kernel sizes and channel numbers. Subsequently, an average-pooling layer is introduced, followed by a feed-forward layer. Afterwards, the output of this layer is fed into a softmax layer, which outputs the probabilities for each class. The initial portion, including the average-pooling layer, learns a general representation of the image. The subsequent Fully Connected (FC) layer is the classification head of the network. The architectural design of ResNet-50, including the kernel sizes and channel numbers, is displayed in Figure 2.7 [16].

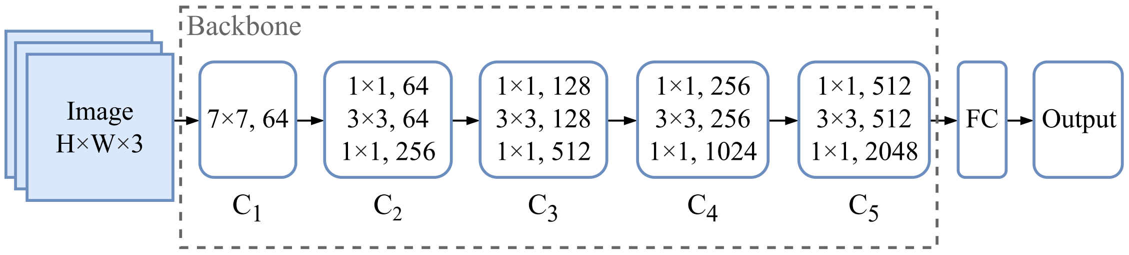


Figure 2.7: Overview of the ResNet-50 architecture. Each block contains the kernel size of its convolutional layers and their output channel dimension (adapted from [35]).

2.3 Transformers

Prior to the emergence of the transformer architecture, the prevailing approaches for sequence modeling and transduction problems were Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks [36], and gated RNNs [37]. Recurrent models are typically constructed in a sequential manner, which precludes the possibility of parallelization. This presents challenges for longer input sequences due to the limitations of memory resources. The transformer architecture deliberately abstains from the use of recurrence and instead relies entirely on an attention mechanism to establish dependencies between the input and the output. This results in a higher degree of parallelization and state-of-the-art performance on a range of benchmarks. Furthermore, the transformer requires considerably less training data than previous state-of-the-art models. In previous models, the number of operations required to relate signals from two arbitrary input or output positions increases with the distance between the positions. In the Transformer, the needed number of operations is independent from the distance. [17] The following subsections provide a comprehensive description of the transformer model, starting with an overview of attention and the proposed Multi-Head Attention (MHA) module.

2.3.1 Attention

A significant attribute of human perception is the tendency to process information in a selective manner, rather than attempting to process the entirety of the information at once. This enables humans to focus their attention on a specific region. Attention mechanisms in deep learning have been designed to reflect this concept. In general, the implementation process of the attention mechanism can be divided into two steps. The first is to compute the attention distribution on the input information, and the second is to compute the context vector according to the attention distribution. In computing the attention distribution, the Neural Network (NN) initially encodes the source data feature as K, which is referred to as the key. The key can be expressed in a variety of representations, depending on the specific task and architecture in question. Furthermore, it is typically required to incorporate a task-specific representation vector, denoted as query Q, into the process. Subsequently, the NN employs a score function to compute the correlation between queries and keys, thereby obtaining the energy score that reflects the importance of queries with respect to keys in determining the subsequent output. The score function plays a pivotal role in the attention model, as it determines the manner in which keys and queries are matched or combined. The energy score is then often used to weight the elements of a new feature representation V, called values. The two most commonly utilized attention mechanisms are additive attention and multiplicative (dot-product) attention [38].

In summary, the authors in [17] describe the attention mechanism: "as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.”

Scaled Dot-Product Attention

The attention employed in the transformer is referred to as 'scaled dot-product attention'. This attention mechanism is identical to the previously presented dot-product attention, except for a scaling factor. It takes queries and keys of dimension d_k and values of dimension d$_v$ as input. Then it takes the dot-product of query and keys and multiplies it with the scaling factor $\sqrt{d_k}^{-1}$. Afterwards, softmax is applied and the result is multiplied with the values. This process is visualized in Figure 2.8 [17].

The intuition behind the scaling is that the dot products magnitude grows for higher query dimensions, which pushes the softmax function into saturation. This results in vanishing gradients. The scaling is used to counteract this effect. The attention function is implemented in the form of matrix multiplications and is computed over a set of queries simultaneously. The vectorized computation is shown in Equation 2.8.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.8}$$

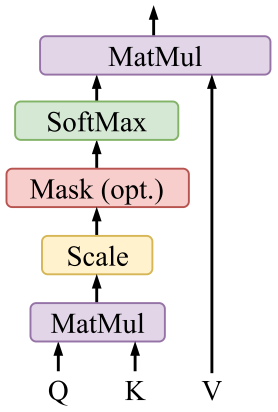


Figure 2.8: Visualization of the scaled dot-product attention. This attention mechanism is used in the transformer architecture. The scaling factor $\sqrt{d_k}^{-1}$ is introduced to keep the values from pushing the softmax function into saturation (adapted from [17]).

Multi-Head Attention

Instead of directly applying the scaled dot-product attention on the d$_{model}$-dimensional K,V and Q, the transformer applies h different learned linear projections on the inputs. The attention function is then applied on each of these projected inputs separately. Each separate attention function is referred to as a single head of the MHA. The outputs of each attention function are then concatenated and projected. The calculations are summarized in Equation 2.9. This multi-head attention is depicted in Figure 2.9. The idea behind this approach is that the model is able to attend to information from different representation subspaces [17].

$$\begin{aligned} MultiHead(Q,K,V) &= Concat(head_1, ..., head_n)W^{Out} \\ \text{with} \;\; head_i &= Attention(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{2.9}$$

,where W^{Out} refers to the output projection weights.

2.3.2 Architecture

The transformer employs an encoder-decoder structure, wherein both the encoder and decoder are constructed from stacked attention and feed-forward layers. The input tokens are initially transformed into vectors through the input embedding process. Subsequently, positional encoding is added to the vectorized tokens. The vectors are then processed by N stacked encoder layers. Thereafter, the (shifted) outputs are vectorized in a manner analogous to the inputs and fed into the decoder. The decoder consists of N subsequent decoder layers and generates the output sequence based on the vectorized outputs and the encoder output [17]. The model architecture is illustrated in Figure 2.10.

The transformer uses multi-head attention in three different ways:

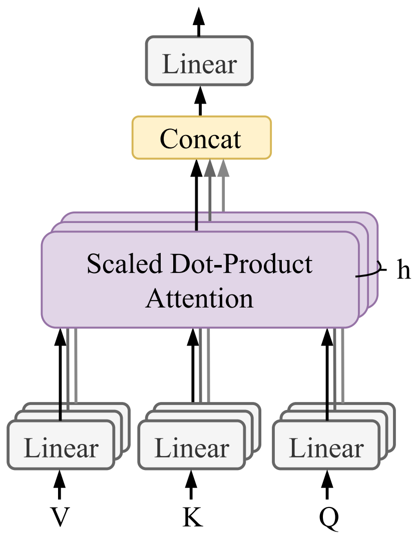


Figure 2.9: Architecture of the multi-head attention module. The query, key and value are projected by h different linear layers and then processed by the scaled dot-product attention. The output is concatenated and then projected to obtain the output (adapted from [17]).

- **Encoder Self-Attention.** Queries, keys and values are the outputs from the previous encoder layer. In the first encoder layer, the input embeddings are used. Each position in the encoder can attend to all positions from the previous layer.
- **Decoder Self-Attention.** Similar to encoder self-attention, except that each position in the decoder can only attend to all positions in the decoder up to and including that position.
- **Decoder Cross-Attention.** The queries are from the previous decoder layer, while the keys and values are from the output of the last encoder layer. Due to this, every position in the decoder can attend over all positions in the encoder.

All components will be explained in detail in the following subsections.

Encoder

The transformer encoder is comprised of six identical encoder layers, each of which contains a MHA layer followed by a feed-forward layer. A residual connection is present around both layers, followed by a layer normalization. The input shape of each encoder layer is identical to the output shape of each layer [17].

Decoder

Analogous to the encoder, the decoder is build from $N = 6$ identical layers. The structure of the decoder layer is very similar to that of the encoder layer, with the

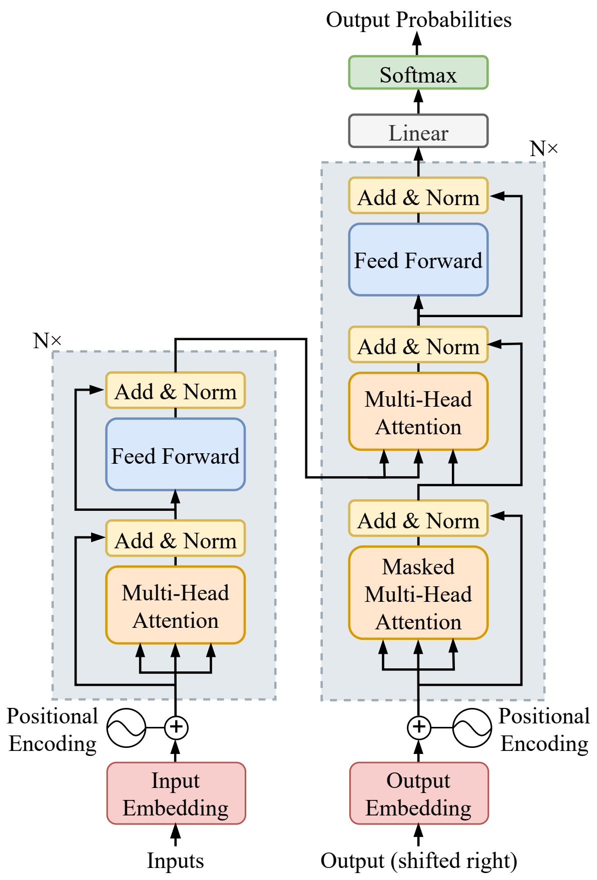


Figure 2.10: Architecture of the transformer model. The input embeddings are refined in the encoder. The final layer encoder output is then passed to the decoder. The decoder applies attention between the output embeddings and the encoder output to generate the output sequence. A positional encoding is necessary as the MHA is translation invariant (adapted from [17]).

exception of the insertion of an additional block between the MHA layer and the feed-forward layer. This additional block is also a MHA layer, but it takes both the previous sub-layers' output and the output from the last encoder layer as an input. It performs what is known as cross-attention between these two inputs. The first MHA layer, which performs self-attention, is also modified in the decoder. It uses a mask to ensure that the predictions for position i only depend on the known outputs at positions $< i$ [17].

Embeddings

Learned embeddings are used to vectorize the input and output tokens. Both embedding layers are sharing the same weights [17].

Positional Encoding

The MHA is position-invariant, it does not preserve any information about the order of the input sequence. Therefore, a positional encoding is added to the input and output embeddings. Both embeddings and positional encodings have the same dimensions. In the transformer, sine and cosine waves of different frequencies are used as relative positional encodings (Equation 2.10). '*pos*' refers to the position of the element in the sequence and i to the dimension [17].

$$\begin{aligned} PE(pos, 2i) &= sin(\frac{pos}{10000^{2i/d_{model}}}) \\ PE(pos, 2i+1) &= cos(\frac{pos}{10000^{2i/d_{model}}}) \end{aligned} \tag{2.10}$$

2.4 Detection Transformer Models

This section discusses different Detection Transformer [11] models, which constitute the fundamental component enabling E2E MOT. Since the publication of the original DETR paper in 2020, a number of different models based on it have been developed. Figure 2.11 presents a chronological overview of select DETR-like models.

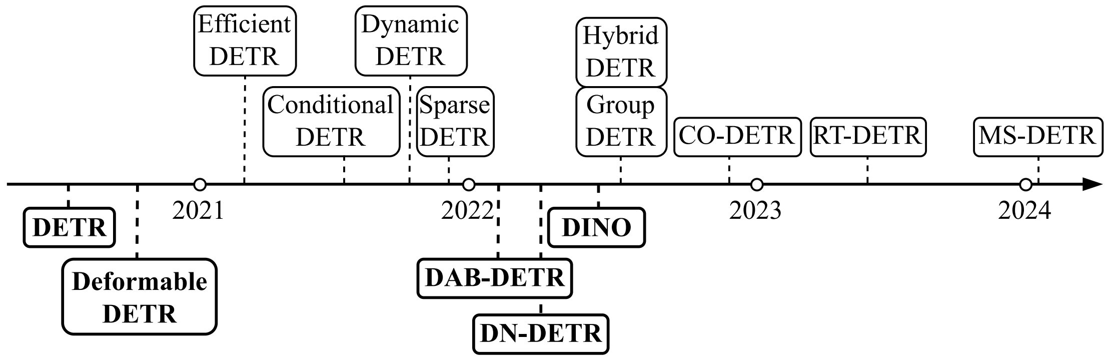


Figure 2.11: Timeline of selected DETR models. Only supervised learning models are considered. This thesis focuses on the standard, deformable, DAB-, DN- and DINO DETR.

The following subsections provides a comprehensive analysis of the original DETR and influential follow-up models. It starts with a review of the original DETR. Afterwards, the Deformable DETR [39] is covered, which introduces a significant speed-up and the use of multi-scale features. Subsequently, DAB-DETR [40] is considered, which reinterprets the role of so-called object queries in DETR models. Instead of learning queries as a high-dimensional embeddings, the authors propose to learn 4-dimensional anchor boxes directly. Based on this, DN-DETR [41] is examined, which introduces an additional reconstruction task. This leads to better performance and convergence speed. Finally, a consideration of DINO [42] follows.

DINO builds on all previously considered DETR-like models and further improves performance. Given that DINO represents the gold standard for DETR-like models, we do not consider any more recent models.

2.4.1 DETR

DETR represents an object detection architecture that regards object detection as a direct set prediction problem. It simplifies the object detection pipeline by eliminating manually crafted components, such as Non-maximum Suppression (NMS) and anchor generation. DETR is leveraging the transformer encoder-decoder structure in combination with a bipartite matching strategy and a suited loss function to achieve this goal. The general architecture, which is illustrated in Figure 2.12, shows this process. DETR extracts features using a CNN and feeds them through the transformer encoder-decoder to predict all objects. During training, a bipartite matching loss is then used to train the model in an End-to-End manner [11].

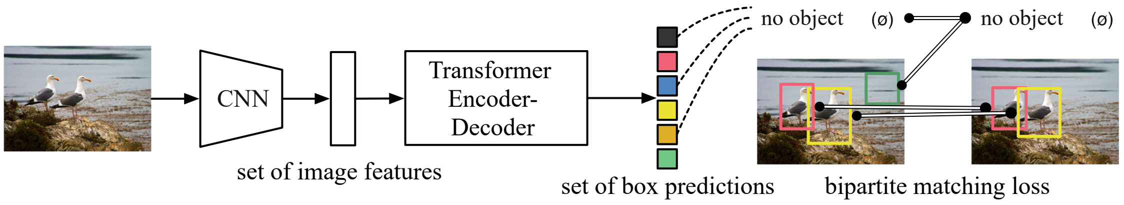


Figure 2.12: Overview of the basic DETR model structure. The model utilizes a transformer encoder-decoder structure to always predicts a fixed number N of detections. During training the predictions are matched with the ground-truth using bipartite matching (adapted from [11]).

In order to achieve a direct set prediction in object detection, two components are essential. The first of these is an appropriate architecture that predicts a set of objects. The second is a prediction loss which forces the model to learn a bipartite matching between the predicted and ground truth boxes [11].

Architecture

The fundamental structure of DETR is straightforward and is illustrated in Figure 2.13. The figure depicts the three primary components of the model: the CNN backbone, which extracts a representation of the image; the transformer encoder-decoder structure; and the prediction heads, which generate the final prediction based on the decoder output. The following sections provide a comprehensive description of these components [11].

Backbone A conventional CNN backbone (e.g. ResNet-50 without its classification head) is used to generate a feature map $f \in \mathbb{R}^{C \times H \times W}$ from the input image $I \in \mathbb{R}^{3 \times H_0 \times W_0}$ [11].

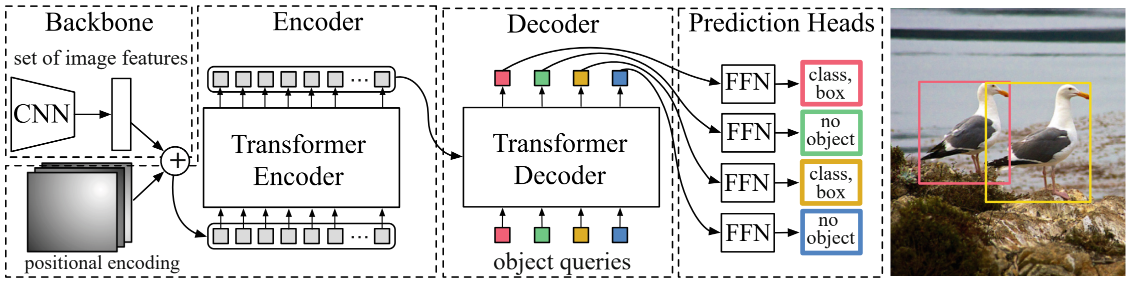


Figure 2.13: The DETR architecture and its main components. The backbone extracts features from the image, which are then processed in the transformer encoder-decoder. The output generated by the decoder is then employed as input to FFNs, which yield the final predictions (adapted from [11]).

Encoder The extracted feature map from the backbone is modified by a 1×1 convolutional layer, reducing its channel dimension from C to d. Given that the encoder expects a sequence as input, the spatial dimensions of the feature map, represented by $f \in \mathbb{R}^{d\times H\times W}$, are flattened. This results in $z_0 \in \mathbb{R}^{d\times HW}$. The encoder structure itself is similar to the original one described in subsection 2.3.2. It consists of 6 identical encoder layers, each containing a MHA module and a Feed-Forward Network (FFN). Due to the permutation-invariance of the encoder, a position-encoding is added to the input of each layer [11].

Decoder Similarly to the encoder, the decoder is also analogous to the standard transformer decoder. However, in contrast to the original transformer, all input embeddings (referred to as 'object queries' in DETR) are decoded in parallel, rather than in series. Given that the decoder is also permutation-invariant, the output embeddings must be distinct in order to generate differing results. These so called object queries are learned positional encodings. They are added to the input of each attention layer in the decoder. The N object queries are transformed into an output embedding by the decoder. Afterwards, each object query is independently decoded into box coordinates and class labels by two FFNs [11].

Feed-Forward Networks As previously stated, each object query is fed into two distinct FFNs, each comprising a Multi-Layer Perceptron (MLP) with three layers, a ReLU activation function, a hidden dimension d_{FFN}, and a linear projection layer. One FFN predicts the bounding box (x, y, h, w) of the detected object, while the other predicts the class label of the prediction using a softmax function. Additionally, a class label for the background class (no detected object) is utilized. As the model predicts N bounding boxes, where N is typically greater than the number of actual objects, a 'no object' class, denoted as $\emptyset$, is essential [11].

Spatial Positional Encoding A d-dimensional fixed position encoding is added to the encoder input. Given that the input in DETR is a 2-dimensional feature map, rather than a 1-dimensional sequence as in the original transformer, the positional encoding needs to be modified accordingly. To achieve this, two independent $\frac{d}{2}$ sine and cosine functions with different frequencies are generated and then concatenated [43].

Loss

The number of objects predicted by the DETR model is typically significantly larger than the actual number of objects present in the image. During training those predictions need to be matched with the ground truth objects based on class, position and size. This is accomplished through an optimal bipartite matching between the predicted and ground truth objects which minimizes a specific loss function [11].

The set of ground truth objects is denoted as y and the set of predictions as $\hat{y} = \{\hat{y}_i\}_{i=1}^{N}$. Under the assumption that there are more predictions than ground-truth objects, the set y is padded with no-class objects. The optimal matching which minimizes a cost function can also be seen as finding the permutation of N elements $\sigma \in \mathcal{P}_N$ with minimal cost. The pair-wise matching cost function between the ground truth y_i and the prediction with index $\sigma(i)$ is denoted $L_{match}(y_i, \hat{y}_{\sigma(i)})$. The matching problem is formulated in detail in Equation 2.11 and can be solved efficiently using the Hungarian algorithm [11, 30].

$$\hat{\sigma} = \underset{\sigma \in \mathcal{P}_N}{\arg\min} \sum_{i}^{N} \mathcal{L}_{match}(y_i, \hat{y}_{\sigma(i)}) \tag{2.11}$$

The matching cost is comprised of two components: A class loss and a bounding box loss. Each ground truth object is build as follows $y_i = (c_i, b_i)$ with target class label c_i and bounding box $b_i \in [0, 1]^4$. The bounding box is normalized to the image size and consists of the center point coordinates (x, y), the height h and the width w. For the prediction with index $\sigma(i)$, the probability of class c_i is defined as $\hat{p}_{\sigma(i)}(c_i)$ and the predicted bounding box as $\hat{b}_i$. With these definitions, it is possible to define the bounding box loss $\mathcal{L}_{box}(b_i, \hat{b}_{\sigma(i)})$ [11].

Bounding Box Loss The bounding box loss is used to evaluate the quality of the bounding boxes. It is a linear combination of the l_1 loss and the generalized Intersection over Union (gIoU) loss $\mathcal{L}_{gIoU}$. This is done to balance the scale-variant l_1 loss with the scale-invariant $\mathcal{L}_{gIoU}$. The bounding box loss is defined in Equation 2.12 with hyper-parameters $\lambda_{L1}, \lambda_{gIoU} \in \mathbb{R}$. The loss is then normalized by the number of objects inside the batch [11].

$$\mathcal{L}_{box}(b_i, \hat{b}_{\sigma(i)}) = \lambda_{gIoU} \cdot \mathcal{L}_{gIoU}(b_i, \hat{b}_{\sigma(i)}) + \lambda_{l_1} \left\| b_i - \hat{b}_{\sigma(i))} \right\|_1 \tag{2.12}$$

Matching Cost Using the definition of the bounding box loss and including a cost term depending on the classification score, the matching cost $L_{match}(y_i, \hat{y_{\sigma(i)}})$ can be defined (Equation 2.13). It is notable that the matching cost between an object and $\emptyset$ is independent of the prediction, which implies that in such a scenario, the cost is a constant [11].

$$\mathcal{L}_{match}(y_i, \hat{y}_{\sigma(i)}) = -[c_i \neq \emptyset] \cdot \hat{p}_{\sigma(i)}(c_i) + [c_i \neq \emptyset] \cdot \mathcal{L}_{box}(b_i, \hat{b}_{\sigma(i)}) \tag{2.13}$$

Hungarian Loss The actual training loss is then computed using the matching from Equation 2.11. It is defined as the sum of the negative log-likelihood of the class prediction and $\mathcal{L}_{box}$. In the implementation the class loss term is down-weighted by a factor 10, when $c_i = \emptyset$. This is done to counter class imbalance. The loss is depicted in Equation 2.14 [11].

$$\mathcal{L}_{Hungarian}(y, \hat{y}) = \sum_{i=1}^{N} \left[-log\hat{p}_{\hat{\sigma}(i)}(c_i) + [c_i \neq \emptyset] \cdot \mathcal{L}_{box}(b_i, \hat{b}_{\hat{\sigma}(i)}) \right] \tag{2.14}$$

Auxiliary Losses DETR uses auxiliary losses in the decoder during training. Prediction FFNs with shared weights are added after each decoder layer to generate the predictions. Subsequently, the Hungarian loss is applied to the predictions of each layer to compute the auxiliary losses [11].

2.4.2 Deformable DETR

DETR suffers from slow convergence speed and limited feature spatial resolution, due to the computational complexity of the attention modules. To solve this problems, [39] proposes the use of the deformable attention module. Deformable attention only attends to small set of sampling locations instead of all tokens. The better computational and memory efficiency enables the use of higher resolutions and multi-scale feature maps. Furthermore, Deformable DETR utilizes an iterative bounding box refinement mechanism to improve detection performance [39].

Deformable Attention

Instead of looking over all possible input tokens, deformable attention only attends to a set of key sampling points around a reference point. Only a small fixed number of keys is assigned to each object query. This boosts the convergence speed and mitigates the resolution limitation of DETR [39].

For a given input feature map $x \in R^{C \times H \times W}$, let q index a query element with content query $z_q \in R^d$ and a 2-dimensional reference point p_q, the deformable attention is calculated by

$$\text{DefAttn}(z_q, p_q, x) = \sum_{m=1}^{M} W_m \left[\sum_{k=1}^{K} A_{mqk} \cdot W'_m \cdot x\left(p_q + \Delta p_{mqk}\right) \right] \tag{2.15}$$

with M attention heads and K sampled keys ($K << M$). $\Delta p_{mqk} \in R^2$ denotes the sampling offset for the k^{th} key in the m^{th} attention head. The attention weights

$A_{mqk} \in [0,1]$ are normalized by $\sum_{k=1}^{K} A_{mqk} = 1$. Both Δp_{mqk} and A_{mqk} are generated by feeding the query features z_q through a linear layer. To obtain the attention weights a softmax layer is applied to the linear projection outputs. The deformable attention module is illustrated in Figure 2.14 [39].

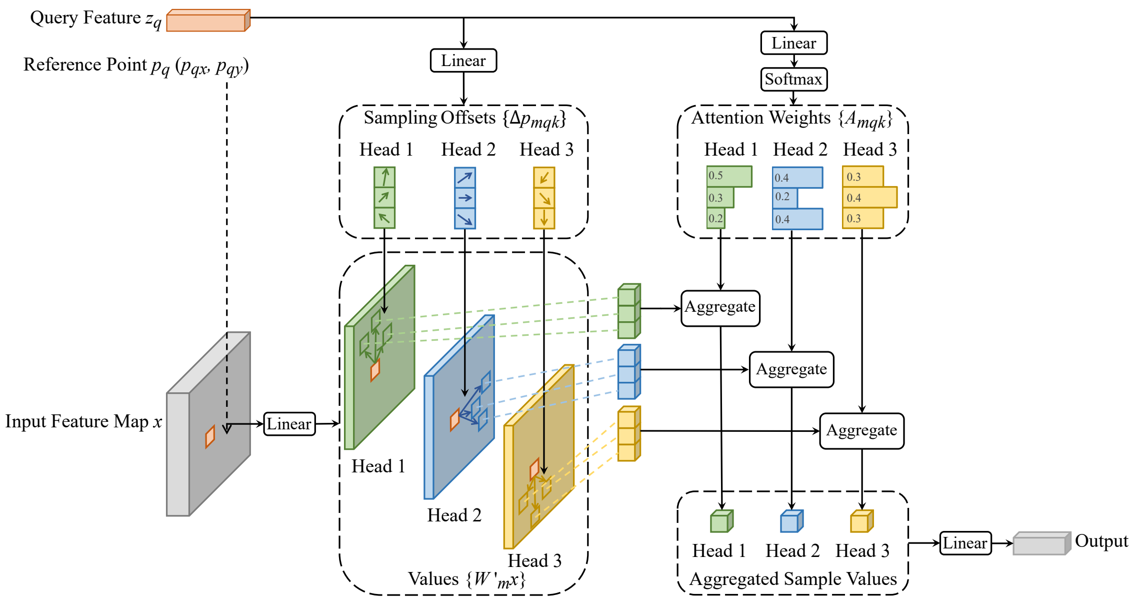


Figure 2.14: Structure of the deformable attention module. The module takes queries, reference points and a feature map as an input. The keys are generated by adding projected offsets to the reference points. The values are a linear projection of the feature map. Deformable attention only attends to a small number of keys (adapted from [39]).

Multi-scale feature maps are commonly used in modern object detection models to improve small object detection performance. The deformable attention module can be extended for multi-scale feature maps [39].

Let $\{x^l\}_{l=1}^{L}$ be the multi-scale feature maps with L scale levels, where $x^l \in \mathbb{R}^{C\times H_l \times W_l}$ and let $\hat{p}_q \in [0,1]^2$ be the normalized reference point for each query element q. The multi-scale deformable attention is then defined by

$$\begin{gathered}\mathrm{MSDefAttn}(z_q, \hat{p}_q, \{x^l\}_{l=1}^{L}) = \\ \sum_{m=1}^{M} W_m \left[\sum_{l=1}^{L} \sum_{k=1}^{K} A_{mlqk} \cdot W'_m \cdot x^l \left(\phi_l\left(\hat{p}_q\right) + \Delta p_{mlqk}\right) \right]\end{gathered} \tag{2.16}$$

Δp_{mlqk} and A_{mlqk} denote the sampling offset and attention weight for the k^{th} sampling point in the l^{th} feature level in the m^{th} attention head. In the multi-scale version, the reference points $\hat{p}_q$ are normalized with respect to the image size. This is done so that the function $\phi_l\left(\hat{p}_q\right)$ can rescale it to the l^{th} level feature map [39].

Deformable Encoder Deformable DETR replaces the self-attention module in the encoder with the proposed multi-scale deformable attention module. It takes $\{x^l\}_{l=1}^{L}$ $(L = 4)$ as input and outputs a multi-scale feature map with the same dimensions. The first 3 scale-level feature maps are the output feature maps from the C_3, C_4 and C_5 stages of ResNet-50 transformed by a 1×1 convolution. The last scale-level feature map is obtained by feeding the C_5 feature map through a convolution layer with kernel size 3×3 and stride 2. The extraction of the multi-scale feature maps is visualized in Figure 2.15. All multi-scale feature maps have $C = 256$ channels. In the encoder the reference points are the feature map pixels themselves. To distinguish between different scale-levels, a learned scale-level embedding $\{e^l\}_{l=1}^{L}$ is added to the positional embedding [39].

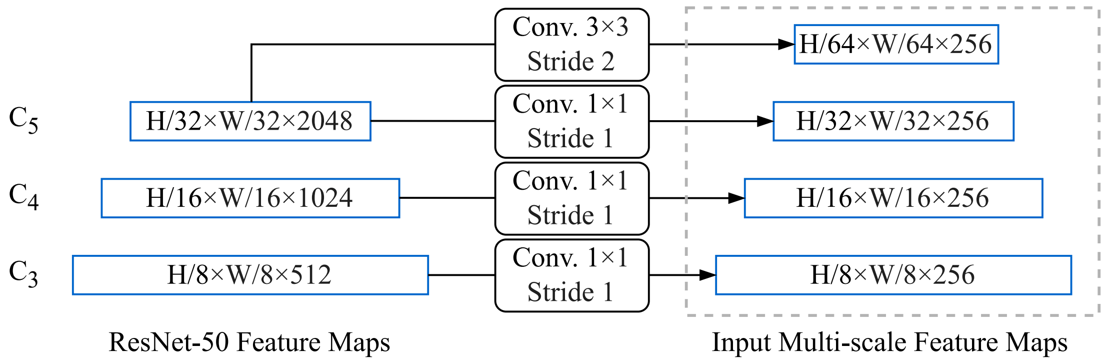


Figure 2.15: Extraction of multi-scale features from ResNet-50. The outputs of the last three layers are convoluted to have 256 channels (adapted from [39]).

Deformable Decoder Deformable DETR only replaces the cross-attention module with the multi-scale deformable attention module. The self-attention modules remain unchanged. For each object query, the reference point is predicted by feeding its query embedding (also called content query) through a linear layer followed by a sigmoid function. The sigmoid function guarantees $\hat{p}_q \in [0,1]^2$. To reduce the optimization difficulty, the detection head predicts the bounding box as a relative offset with respect to the reference point [39].

Complexity

The main argument in favor of the deformable attention module is the lower computational complexity, which allows the use of higher resolutions and multi-scale feature maps. Table 2.1 illustrates the advantage of deformable attention over classical attention for self-attention in the encoder and cross-attention in the decoder, where the input image is $I \in \mathbb{R}^{C \times H \times W}$ and K denotes the number of keys per query. Since $HW \gg C$ and $HW \gg K$, the deformable attention module is significantly more efficient and enables much higher resolutions due to the linear scaling with the number of pixels in the encoder [39].

Table 2.1: Complexity of the different attention modules in DETR and Deformable DETR.

	DETR	Deformable DETR
Encoder (Self-Attention)	$\mathcal{O}(CH^2W^2)$	$\mathcal{O}(C^2HW)$
Decoder (Self-Attention)	$\mathcal{O}(C^2N + CN^2)$	$\mathcal{O}(C^2N + CN^2)$
Decoder (Cross-Attention)	$\mathcal{O}(CHWN)$	$\mathcal{O}(C^2KN)$

Iterative Bounding Box Refinement

The detection head predicts the relative offset with respect to the reference point $\hat{p}_q = (\hat{p}_{q,x}, \hat{p}_{q,y})$. In this process $\hat{p}_q$ is used as an initial guess for the bounding box center. Sigmoid σ and inverse sigmoid σ^{-1} are used to guarantee that $\hat{b}_q \in [0,1]^4$. This process is shown in Equation 2.17 [39].

$$\hat{b}_q = \left\{\sigma\left(b_{q,\{x,y\}}\right) + \sigma^{-1}\left(\hat{p}_{q,\{x,y\}}\right), \sigma\left(b_{q,\{w,h\}}\right)\right\} \tag{2.17}$$

The idea behind the bounding box refinement is that each decoder layer refines the bounding box based on the predictions from the previous decoder layer. Let D be the number of decoder layers and $\hat{b}_q^{d-1}$ be the normalized bounding box predicted from decoder layer $(d-1)$. Then the d^{th} decoder layer refines the bounding box by

$$\hat{b}_q^d = \left\{\sigma\left(\Delta b^d_{q,\{x,y,w,h\}} + \sigma^{-1}\left(\hat{b}^{d-1}_{q,\{x,y,w,h\}}\right)\right)\right\} \tag{2.18}$$

with $d \in \{1, ..., D\}$. The offsets predicted by the d^{th} decoder layer are denoted as $\Delta b^d_{q,\{x,y,w,h\}} \in \mathbb{R}$. The center coordinates of the initial bounding box $\hat{b}^0$ are the reference points. The height and width are set to 0.1. To stabilize training, the gradients are blocked at $\sigma^{-1}\left(\hat{b}^{d-1}_{q,\{x,y,w,h\}}\right)$. In addition, the sampling offset Δp_{mlqk} is modulated by the box size, by multiplying it with the width and height. This makes the sampling locations related to both center and size of the box predictions from the previous decoder layer [39].

Two-Stage

In the normal Deformable DETR, object queries are learned and do not depend on the current image. The Two-Stage Deformable DETR is a variant of Deformable DETR which extracts the object queries from the input image. These extracted queries can be seen as a region proposals. This approach is strongly inspired by two-stage object detectors [39].

In the first stage, all pixels of the encoder output feature map are seen as object query candidates. A detection head consisting of a 3-layer FFN for bounding box regression and a linear layer for classification (foreground/background) is applied to each pixel. The top-K scoring pixels and their corresponding predicted bounding boxes are used as object queries. The pixel channels form the content query and the predicted box forms a 4-dimensional reference point. The reference point can be seen as a region proposal. In the second stage, the extracted object queries are fed into the decoder. The detection head is trained by using the Hungarian Loss [39].

2.4.3 DAB-DETR

Dynamic Anchor Boxes (DAB)-DETR [40] is an alternative approach to mitigate the slow training convergence and improve the performance of the original DETR. This is achieved by a more detailed examination of the cross-attention module and the use 4D anchor boxes (x, y, w, h) as learnable object queries. The primary insight derived from this formulation is that each query in DETR is composed of two distinct parts. A content part (the decoder self-attention input) and a positional part (learnable queries in original DETR). In the cross-attention module, the attention weights are then computed by comparing the queries with a set of keys. These keys also contain a content part (encoded image features) and a positional part (positional embeddings). Therefore, queries in the decoder can be interpreted as pooling features from a feature map based on query-to-feature and positional similarity. This motivates the formulation of queries as anchor boxes, the use of the center position (x, y) of the anchor box to pool features around the center and the use of the box size (w, h) to modulate the cross-attention map. Furthermore, the use of anchor boxes as queries allows them to be updated layer-by-layer. Except for the decoder and its queries, DAB-DETR is structurally identical to the original DETR. The structure of the DAB-DETR decoder is shown in Figure 2.16 [40].

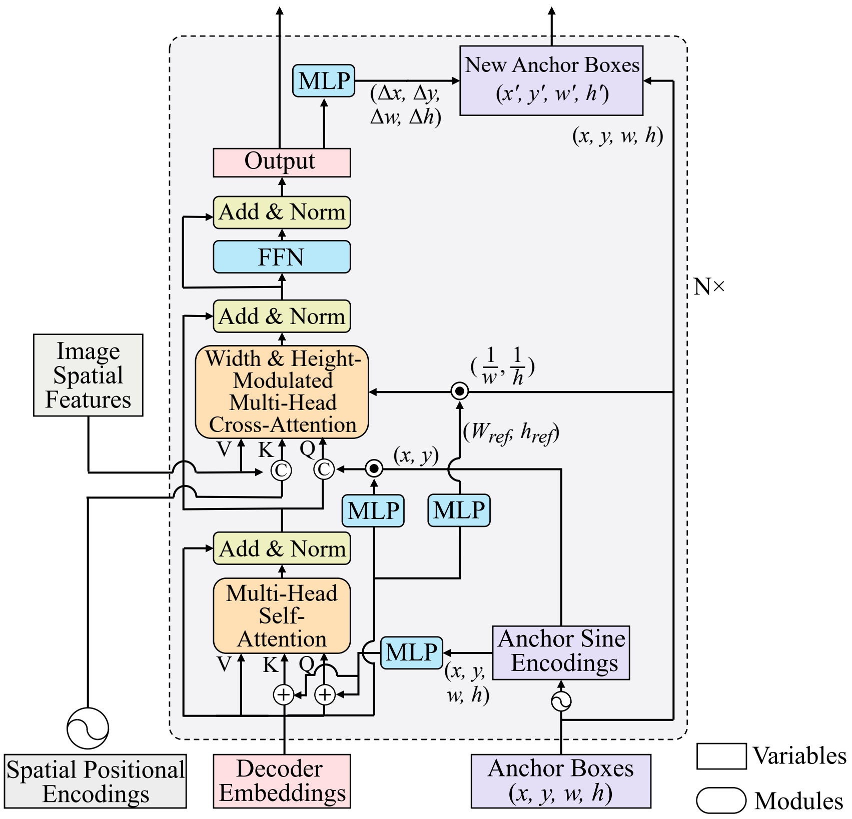
Learning Anchor Boxes

As discussed before, DAB-DETR directly learns anchor boxes as queries and derives the actual positional queries from them. Let $A_q = (x_q, y_q, w_q, h_q) \in \mathbb{R}^4$ be the q^{th} anchor, $\mathcal{C}_q \in \mathbb{R}^d$ and $\mathcal{P}_q \in \mathbb{R}^d$ its corresponding content and positional query. For a given anchor A_q, its positional query is calculated as followed [40].

$$\begin{aligned} \mathcal{P}_q &= MLP\left(PE\left(A_q\right)\right) \\ PE\left(A_q\right) &= Cat\left(PE\left(x_q\right), PE\left(y_q\right), PE\left(w_q\right), PE\left(h_q\right)\right) \end{aligned} \tag{2.19}$$

where PE is the positional encoding which generates a sinusoidal embedding with $\frac{d}{2}$ from a float number $\left(PE : \mathbb{R} \to \mathbb{R}^{d/2}\right)$. The MLP then projects the 2d-dimensional vector into d-dimensional representation $\left(MLP : \mathbb{R}^{2d} \to \mathbb{R}^{d}\right)$ [40].

The self-attention inputs are identical to the original DETR. Queries, keys and values all have the same content queries, while the positional queries are added to the queries and keys. In the cross-attention module, the queries and keys are formed by concatenating position and content information together. Moreover, a MLP is learned to rescale the positional embeddings $\left(MLP^{(scale)} : \mathbb{R}^d \to \mathbb{R}^d\right)$. The generation of the cross-attention inputs is described in Equation 2.20 [40].



New Anchor Boxes (x', y', w', h')
MLP
$(\Delta x, \Delta y, \Delta w, \Delta h)$
(x, y, w, h)
Output
Add & Norm
FFN
Add & Norm
N×
Image Spatial Features
Width & Height-Modulated Multi-Head Cross-Attention
$(\frac{1}{w}, \frac{1}{h})$
(W_{ref}, h_{ref})
V K Q
(x, y)
MLP
MLP
Add & Norm
Multi-Head Self-Attention
V K Q
MLP
(x, y, w, h)
Anchor Sine Encodings
Spatial Positional Encodings
Decoder Embeddings
Anchor Boxes (x, y, w, h)
Variables
Modules

Figure 2.16: The structure of the DAB-DETR decoder. It uses anchor boxes as positional queries and updates them layer-by-layer. The MHA module in the cross-attention is exchanged with a width & height modulated version. Analog to Deformable DETR, it predicts relative offsets for the corresponding anchor box (adapted from [40]).

$$\begin{aligned} \mathrm{Q}_q &= Cat\left(\mathrm{C}_q, PE\left(x_q, y_q\right) \cdot MLP^{(scale)}(\mathrm{C}_q)\right), \\ \mathrm{K}_{x,y} &= Cat(F_{x,y}, PE\left(x, y\right)), \\ \mathrm{V}_{x,y} &= F_{x,y} \end{aligned} \tag{2.20}$$

where $F_{x,y} \in \mathbb{R}^d$ is the image feature at position (x, y).

Anchor Update

Using anchor boxes as queries enables a layer-by-layer update by adding the predicted relative position offsets $(\Delta x, \Delta y, \Delta w, \Delta h)$ to them. For other models, like the original DETR, it is hard to perform iterative query refinement, because it is unclear how to convert the updated anchor boxes back to the query embeddings [40].

DAB-DETR uses only single-scale feature maps as input. As a result, the detection performance is inferior to that of Deformable DETR. However, the idea of dynamic anchor boxes is transferable to the Deformable DETR, and the resulting model is the DAB-Deformable DETR. Since this version is superior to the DAB-DETR, a detailed consideration of the DAB-DETR specific components is omitted and the DAB-Deformable DETR is considered instead [40].

DAB-Deformable DETR

The DAB-Deformable-DETR is derived by incorporating the dynamic anchor boxes design into the Deformable DETR framework. The primary distinction between this approach and Deformable DETR is that, rather than learning a d_m-dimensional positional query, anchor boxes are learned. The actual positional queries and anchor updates are then handled in a manner consistent with the approach previously described for DAB-DETR. To provide a comprehensive illustration of the differences between the various DETR architectures, the decoder structures for all previously mentioned DETR variations are presented in Figure 2.17 [40].

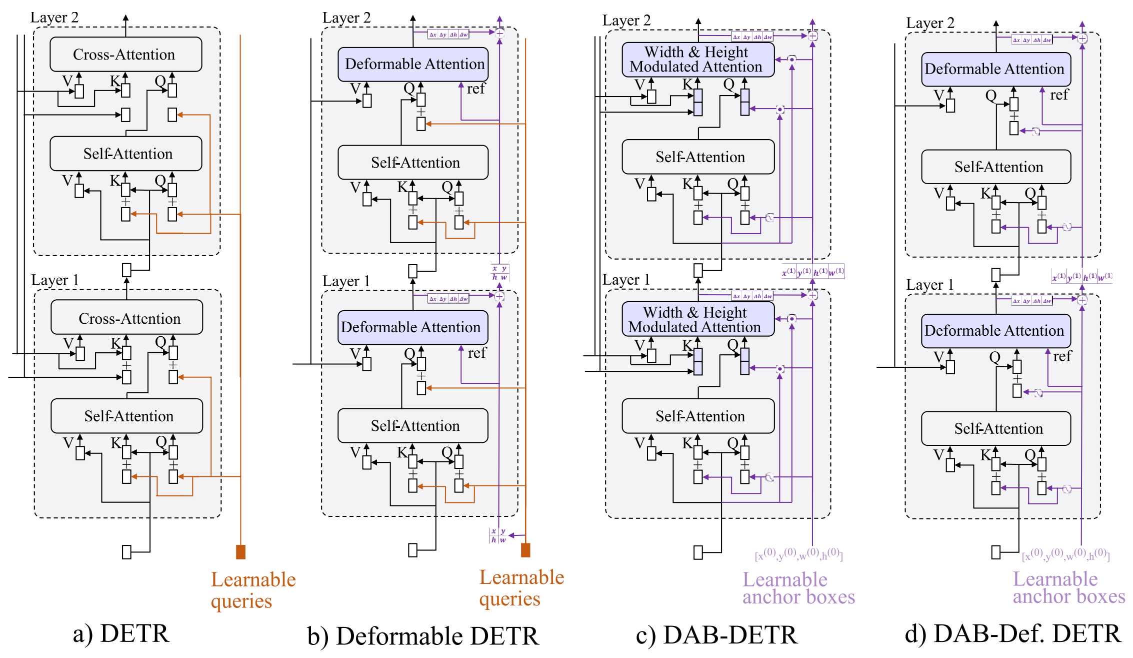


Figure 2.17: Comparison of different DETR-like models' decoders. a) shows the original DETR decoder. The positional queries are learned, while the content queries are kept empty. b) depicts the Deformable DETR. It uses learnable content and positional queries in combination with the deformable attention module. It can be seen that the reference points are updated layer-by-layer. In c) the DAB-DETR decoder can be seen. Positional queries are learned directly as 4-dimensional anchor boxes. d) illustrates the DAB-Deformable DETR decoder. It adopts the anchor boxes and the refinement from DAB-DETR, while keeping the content queries learnable and uses deformable attention (adapted from [40]).

2.4.4 DN-DETR

Denoising (DN)-DETR [41] introduces a universal denoising training method to speed up the convergence of DETR models. It reasons that the instability of the bipartite graph matching used in the Hungarian loss is the main reason for the slow convergence. As a solution, [41] propose to feed ground-truth boxes with noise into the decoder and train the model to reconstruct the original boxes. This reduces the bipartite graph matching difficulty, leading to faster convergence and better performance [41].

Hungarian Matching Instability

Hungarian matching introduces instability as a small change in the cost matrix may lead to a vastly different matching. This negatively effects the convergence speed as it leads to inconsistent optimization goals for the decoder queries. The training process of DETR-like models can be interpreted as a two stage process, where the first stage is learning anchors (in the form of queries) and the second stage is learning relative offsets. The inconsistent anchor updates introduced by the Hungarian matching complicate the learning of relative offsets [41].

DN-DETR, therefore introduces an additional denoising task, which bypasses the bipartite matching, to make the offset learning easier. It achieves this by generating 'good anchors' in the form of noisy ground-truth boxes. These 4-dimensional anchors can be interpreted as queries. The denoising task is then that each of this, so called denoising queries tries to reconstruct the original ground-truth box. This task has a clear optimization which avoids the conflict introduced by the bipartite matching [41].

Overview

DN-DETR is based on DAB-DETR. The architecture is identical. The only difference is the introduction of the denoising task during training [41].

During training, the DN-DETR queries consist of two different parts. The matching part are the already known learnable anchors from DAB-DETR. They are assigned to the ground-truths by bipartite matching. The denoising part are noised ground-truth box-label pairs (so called ground-truth objects). These queries try to reconstruct their original ground-truth object. This is done to increase the efficiency. In addition, an attention mask is used to prevent information leakage from the denoising part to the matching part. The DN-DETR decoder during training is illustrated in Figure 2.18. It is important to keep in mind that the denoising is only used during training. In inference only the matching part is present [41].

Denoising

The denoising queries are generated by adding random noise to all ground-truth objects' bounding boxes and class labels. Noise is applied to the boxes in two different ways:

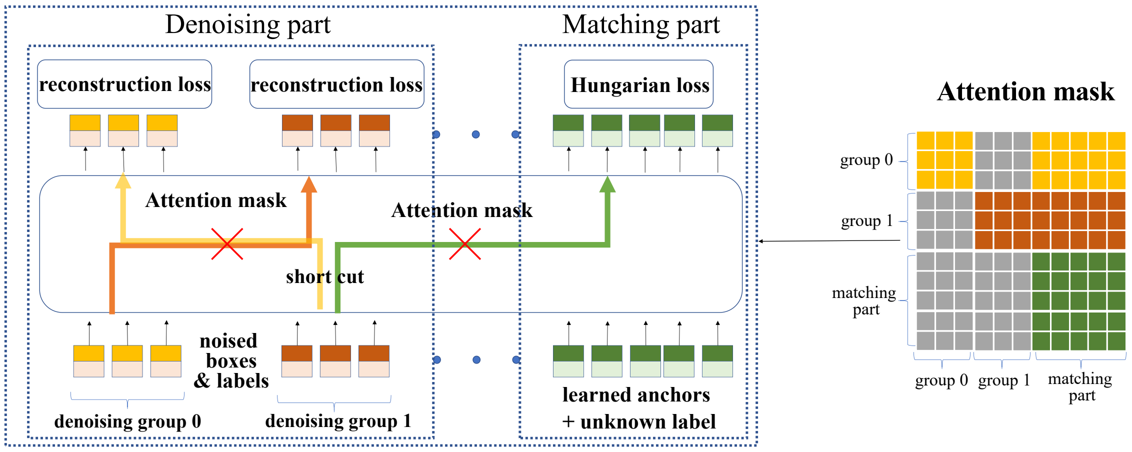


Figure 2.18: Overview of DN-DETR's decoder. The decoder is separated in the denoising part and the matching part. It can be seen that the object queries can not attend to the denoising queries due to the attention mask (adapted from [41]).

1. **center shifting:** Adding noise $(\Delta x, \Delta y)$ to the center coordinates of the box. To guarantee that the new center point is still in the original bounding box, the following needs to hold $(|\Delta x| < \frac{\lambda w}{2}, |\Delta y| < \frac{\lambda h}{2})$.
2. **box scaling:** The width and height (w, h) are randomly sampled from the intervals $[(1-\lambda)w, (1+\lambda)w]$ and $[(1-\lambda)h, (1+\lambda)h]$.

where $\lambda \in (0, 1)$ is a hyper-parameter. For label noising, the ground-truth labels are randomly changed to different ones. This forces the model to predict labels according to the noisy boxes. The reconstruction losses for the denoising part are the same as for the matching part (gIoU and l_1 loss for the box regression, focal loss [44] for label classification). To increase the efficiency of the denoising training, multiple noisy versions of each ground-truth object are created [41].

Attention Mask

An attention mask is necessary to ensure that no information leakage occurs, which would harm the performance. There are two possibilities for information leakage which need to be prevented. One is that the matching part may see the noisy ground-truth objects. This eases the prediction during training and would lead to worse performance during inference, where the denoising queries are not present. The other one is that a noisy version of a ground-truth object can see other versions of the same object [41].

To describe the structure of the attention mask, the denoising queries are separated into groups. Each group contains exactly one noisy version of each ground-truth object. The attention mask is denoted as $A \in [0, 1]^{N \times N}$, where N is the total number of queries. With D denoising groups, G ground-truth objects and M matching queries $N = D \cdot G + M$. The attention mask is structured such that the first $D \cdot G$ columns and rows represent the denoising part and the latter M the matching part. When A_{ij} is 1, the i^{th} and j^{th} query do not interact during attention.

If they are able to see each other A_{ij} is 0. The attention mask is then generated according to Equation 2.21 and an example mask can be seen in Figure 2.18 [41].

$$f(x) = \begin{cases} 1 & ,\text{if } j < D \cdot G \ \& \ \lfloor \frac{i}{M} \rfloor \neq \lfloor \frac{j}{M} \rfloor \\ 1 & ,\text{if } j < D \cdot G \ \& \ i \geq D \cdot G \\ 0 & , else \end{cases} \tag{2.21}$$

Label Embedding

The decoder embedding (content query) is 0 in DAB-DETR. DN-DETR uses a learned label embedding to support both box and label denoising. The embedding has the dimension $(num\_classes + 1) \times (d_{model} - 1)$. In the denoising part, the label embedding for each query is picked according to its noisy class label. To keep semantically consistency, the matching part also uses a label embedding. The embedding is the $(num\_classes + 1)^{th}$ embedding which represents an unknown class embedding. Moreover, an indicator is concatenated to the label embeddings. The indicator is set to 1 if it is a denoising query and 0 otherwise [41].

Complexity

The effect of the additional denoising queries on the training time is negligible. When training on MS COCO 2017 [45] with 5 denoising groups and a ResNet-50 backbone, the GFLOPS during training increase from 94.4 (DAB-DETR) to 94.6. This is due to the fact, that the majority of the computational complexity is due to the encoder. [46] observed that the encoder has a 4-8 times higher latency and computational cost than the decoder for most DETR-like models [41].

2.4.5 DINO

DINO [42] builds upon Deformable, DAB- and DN-DETR. It proposed a Mixed Query Selection (MQS) strategy to extract 4-dimensional positional queries from the encoder while keeping the content queries learned. Furthermore, it uses deformable attention analog to Deformable DETR. DINO also adopts the denoising training from DN-DETR. Instead of using the standard denoising, it introduces a contrastive approach by including hard negative samples. Finally, it expands Deformable DETRs iterative bounding box refinement with its own Look Forward Twice (LFT) method. The complete DINO architecture is visualized in Figure 2.19 [42].

Contrastive Denoising

DN-DETRs denoising training helps anchors to predict nearby ground-truth objects, but it lacks the ability to reject low quality anchors. Contrastive Denoising (CDN) addresses this issue. During CDN, a negative and a positive denoising query is generated for each ground-truth object. Therefore, for n ground-truth objects, each denoising group contains $2n$ denoising queries. Positive queries have a noise scale smaller than λ_1, whereas negative queries have one between λ_1 and

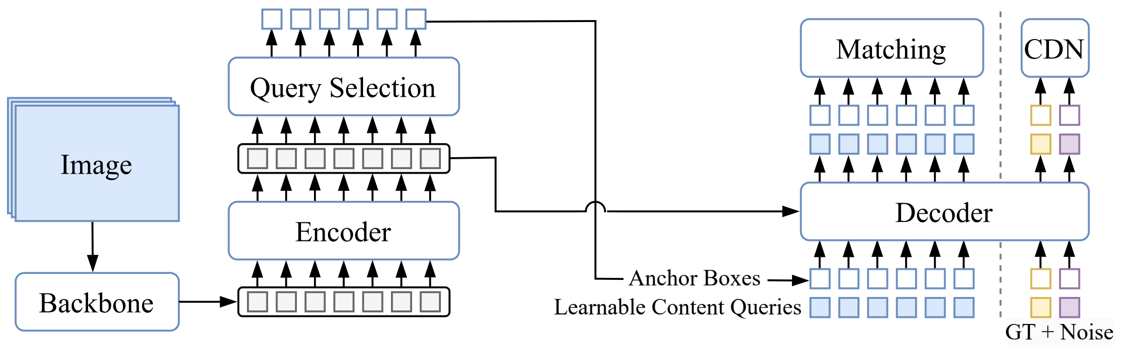


Figure 2.19: Overview of DINO's architecture. The positional queries are extracted from the encoder outputs, whereas the content queries are learned. Positive and negative denoising queries are fed through the decoder during training (adapted from [42]).

λ_2 with $\lambda_1 < \lambda_2$. Positive queries are expected to reconstruct their corresponding ground-truth object, while negative queries should predict 'no object'. λ_2 is chosen relatively small to get hard negative samples. Similar to DN-DETR, DINO uses multiple denoising groups. CDN assists the model in selecting the optimal anchor in instances where there are multiple anchors in close proximity to the ground-truth object, and in rejecting those that are more distant [42].

Mixed Query Selection

In previous works positional and content queries are either both learned or both extracted. DETR, DAB-DETR and DN-DETR learn the positional queries and set the content queries to 0. (DAB-)Deformable DETR learns both queries while its two-stage version extracts both from the encoder output. DINO proposes MQS, where the positional queries (in the form of 4-dimensional anchor boxes) are extracted from the encoder output and the content queries are learned. It reasons that the selected preliminary content queries from the encoder may be confusing for the decoder as they may contain multiple objects or only parts of an object. Therefore, DINO keeps the content queries learned. The positional queries are extracted to help the model to pool more comprehensive content features from the encoder during the cross-attention. A comparison between the different query generation methods for DETR-like models is given in Figure 2.20 [42].

Look Forward Twice

DINO employs a modified a version of Deformable DETRs iterative bounding box refinement to predict its bounding boxes. Deformable DETR blocks gradients during backpropagation, such that the parameters of the ith layer are updated based only on the ith auxiliary box loss. DINO however assumes that the information from later layers may be beneficial to correct the box predictions in the previous

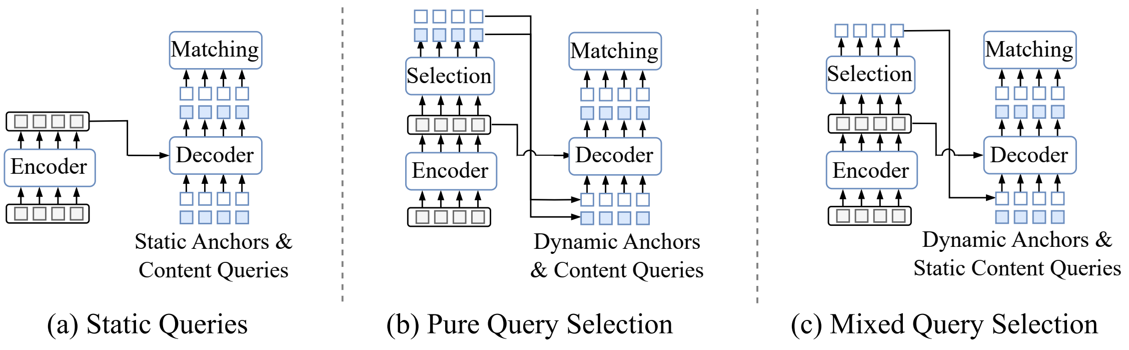

Matching
Decoder
Encoder
Static Anchors & Content Queries
(a) Static Queries
Selection
Dynamic Anchors & Content Queries
(b) Pure Query Selection
Dynamic Anchors & Static Content Queries
(c) Mixed Query Selection

Figure 2.20: Overview of different query generation methods. In (a) both query parts are learned, while in (b) both are extracted from the encoder outputs. (c) depicts DINO's Mixed Query Selection. There the positional queries are extracted and the content queries are learned (adapted from [42]).

layers. Hence, it updates the ith layers parameters depending on ith and the (i+1)th auxiliary box loss. Since, the parameters are updated twice, the method is called LFT. A visualization of the iterative bounding box refinement and LFT is displayed in Figure 2.21 [42].

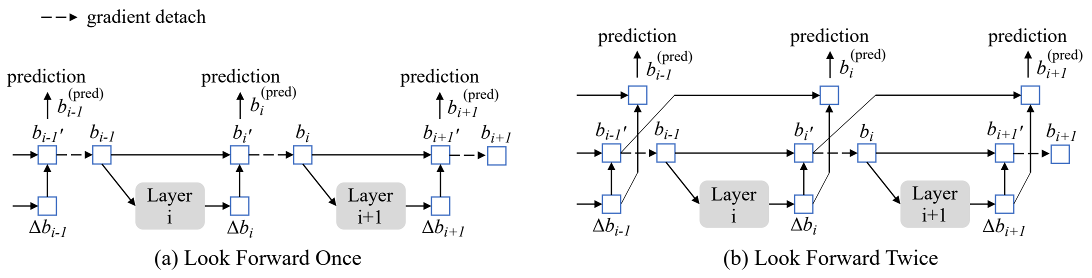


Figure 2.21: Comparison of iterative bounding box refinement and LFT. a) depicts the iterative bounding box refinement. It can be seen that the ith layer's parameters only depend on ith auxiliary box loss. b) shows LFT. There the ith layer's parameters also depend on the (i+1)th loss (adapted from [42]).

2.5 End-to-End Multi Object Tracking

Tracking-by-Detection is currently the most widespread MOT paradigm. Most of the existing methods separate the MOT temporal association into appearance and motion: appearance variance is usually measured by pair-wise Re-ID similarity [23] while motion is modeled via IoU or Kalman-Filtering heuristic [20].

Transformer-based E2E models, also known as Tracking-by-Query [25] paradigm, pursue a different approach. Instead of splitting detection and association in two stages, the association is handled implicitly through the queries in the DETR architecture. Furthermore, this paradigm features a joint motion and appearance modeling. Since the publication of the first E2E models, many publications have followed. The most influential models of this paradigm are TrackFormer [12] and MOTR [13]. While TrackFormer only trains on two consecutive frames, MOTR trains on a sequence of up to 5 frames. A selection of different models is illustrated in Figure 2.22. It can be seen that most of the models are based on MOTR.

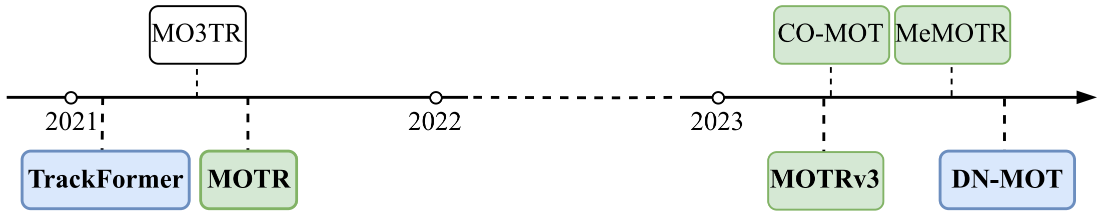


Figure 2.22: Timeline of selected transformer-based end-to-end models. The models highlighted in blue are based on TrackFormer, whereas the ones in green are based on MOTR. The gray models are independent.

This section looks at the various architectures. First, the basic functionality is explained using the example of TrackFormer. Then the changes that DN-MOT [14] proposes are discussed. The focus is then placed on MOTR. Finally, the modifications proposed in MOTRv3 [47] are examined.

2.5.1 TrackFormer

TrackFormer [12] repurposes the DETR architecture for MOT and interprets the tasks as a frame-to-frame set prediction problem. It achieves data association between frames by propagating a set of track predictions through a video sequence. The decoder initializes new tracks from object queries which are detecting objects and continues existing tracks with new identity preserving track queries. The structure of TrackFormer is nearly identical to DETR and no new components are introduced. The tracking process is visualized in Figure 2.23 [12].

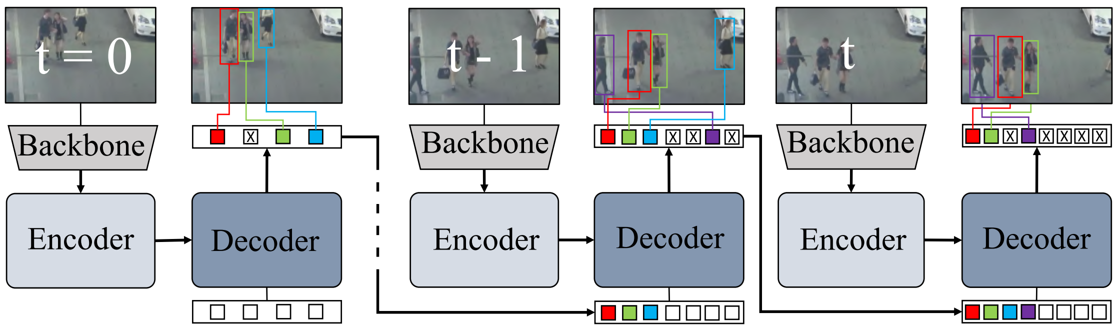


Figure 2.23: Overview of the TrackFormer architecture. TrackFormer is based on DETR. It propagates successful object queries through a sequence of frames to track objects. These queries are called track queries. A track query tries to track the same object over the whole sequence (adapted from [12]).

The following subsection introduces the Tracking-by-Query paradigm and the concept of track queries. Subsequently, a comprehensive examination of the training process, including tracking-specific image augmentations, and implementation details is conducted.

Tracking-by-Query

In DETR models, objects are implicitly represented by the decoder queries, which are embeddings utilized by the decoder to predict the bounding box and class label. The Tracking-by-Query paradigm leverages this fact by adding auto-regressive track queries in addition to the already existing object queries. The object queries allow the model to initialize tracks by detecting objects, while the track queries are responsible for tracking objects across frames. By simultaneously providing object and track queries to the decoder, the model is capable of performing detection and tracking in a unified manner [12].

Track Queries In order to achieve frame-to-frame association, TrackFormer introduces the concept of track queries to the decoder. Track queries follow objects through a video sequence carrying over their identity information while adapting to their changing position and appearance in an auto-regressive manner [12].

For this purpose, each new object which is successfully detected by an object query (classification score above τ_{object}) initializes a track query. The track queries are initialized with the positional and content part from their corresponding object queries after the final decoder layer. The newly initialized track queries, already existing track queries and the object queries are then fed into the decoder in the next frame. The cross-attention between the frame features and the queries continuously updates the representation of the tracked object in each track query. The self-attention mechanism of the decoder allows the detection of new objects while simultaneously suppressing the detection of already tracked objects. The decoder refines the joint set of queries and provides them to the bounding box and classification head to generate the predictions [12].

Whereas the number of object queries $N_{objects}$ is static, the number of track queries N_{track} changes between frames due to the detection of new objects and the removal of tracks. Tracks can be removed if their class score drops below a threshold τ_{track} or by NMS. TrackFormer uses a high IoU-threshold τ_{NMS}, which only removes highly overlapping duplicate boxes. NMS is used since the decoders self-attention mechanism is not able to suppress these conflicts by itself [12].

Re-Identification As a result of occlusion or short-term appearance changes, it can happen that the class score of a track query falls below τ_{track}. To have to possibility to continue these tracks nonetheless, TrackFormer introduces a short-term re-identification process. Instead of discarding queries directly when the class score drops below τ_{track}, the queries are kept for an additional T_{reid} frames. During this time window, track queries are considered to be inactive. This means that they are not contributing to the track until they reactivate themselves by achieving a class score above T_{reid}. Since a track query contains spatial information that does not adapt during the inactive time, this Re-ID mechanism is only suitable for short-term recovery [12].

Training

Since track queries are initialized by successful detections in the previous frame, training must be performed on a sequence of at least two adjacent frames. As a result, TrackFormer trains on frame t and a previous frame $t-1$. The loss is only calculated on frame t as it contains both track and object queries which allows to optimize both detection and tracking at once. Similar to DETR, the loss consists of a class loss and bounding box component for each query. The training process consists of the two following steps [12]:

1. **Detection step:** Object Detection on frame $t-1$ with N_{object} object queries. Detections are then matched with ground-truth objects. The matched object queries are initializing track queries for frame t.
2. **Tracking step:** Joint Detection and Tracking on frame t with $N = N_{object} + N_{track}$ queries. Object queries are detecting new-born objects, while track queries track their corresponding object. The loss is then calculated for all N queries.

Bipartite Matching DETR performs a bipartite matching between its predictions and the ground-truth objects which minimizes a given cost function. TrackFormer handles its object queries similar, but uses a fixed assignment strategy for matching the track queries with the ground truth. The matching $(y_i, \hat{y}_{\sigma(i)})$ between the ground-truth object y_i and the prediction $\hat{y}_i$ is based on similarity for object queries and on track identity for track queries [12].

To formulate the matching process, the set of ground-truth track identities in frame t is denoted as K_t. In the first training step (Detection step), there are no track queries. Therefore the object queries are matched with the ground-truth analogue to DETR. Each matched query is assigned its corresponding track identity from the set K_{t-1}. In the next step (Tracking step), when there are track queries present, the matching is performed according to:

1. $\mathbf{K_{t-1} \cap K_t}$: For tracks which are present in both frames, the track query initialized by the detections in frame $t-1$ is expected to detect the same object in frame t. These track queries are matched by their track identity k.
2. $\mathbf{K_{t-1} \setminus K_t}$: For tracks which are ending/occluded in frame t, the corresponding track queries are matched with the background class in frame t.
3. $\mathbf{K_t \setminus K_{t-1}}$: Newly appearing object identities are matched with object queries by minimizing a cost function. This process is identical with the bipartite matching in DETR (2.4.1).

Object queries which are not matched during the Hungarian matching are assigned to the background class [12].

Loss The MOT set prediction loss is computed over all $N = N_{object} + N_{track}$ output predictions from frame t:

$$\mathcal{L}_{MOT}(y, \hat{y}) = \sum_{i=1}^{N} \left[\mathcal{L}_{class}(y_i, \hat{y}_{\hat{\sigma}(i)}) + \mathcal{L}_{box}(y_i, \hat{y}_{\hat{\sigma}(i)}) \right] \tag{2.22}$$

The loss function itself is identical to the one used in DETR. It uses cross-entropy/ focal loss [44] as label loss and a combination of gIoU and l_1 loss as box loss. It is important to keep in mind that the loss is only calculated for frame t. Frame $t-1$ is only needed to initialize track queries [12].

Track Augmentation In addition to conventional image augmentation, like cropping or horizontal flipping, TrackFormer introduces track specific augmentation to mimic different possible tracking scenarios. The following three augmentations are proposed [12]:

1. **Previous Frame Sampling:** By sampling frame $t-1$ for the first training step (detection step) from an interval around frame t, low frame rates and more difficult tracking scenarios can be simulated.

2. **False Negatives:** Track queries initialized in step in the first training step are removed with a probability p_{FN} before continuing with the next step (tracking step). This frees their corresponding ground-truth objects for matching with object queries. A high p_{FN} is needed to joint-train tracking and detection. Without false negatives, nearly exclusively track queries would be trained as the loss is only calculated for frame t.

3. **False Positives:** By adding false positive track queries, the removal of duplicate/incorrect track queries is improved. The false positive queries are sampled from object queries which were not matched in frame $t-1$. A false positive query can be generated for each track query with the possibility p_{FP}.

Implementation Details There are two versions of TrackFormer, one is based on the original DETR and the other one on Deformable DETR. Due to the superior performance of the Deformable DETR version, all implementation details are based on this version [12].

As Deformable DETR uses reference points, the track queries adapt the center points of the previous frame bounding boxes as reference points. Moreover, TrackFormer stacks the encoder outputs from the current frame and the previous frame and computes the cross-attention between them and the queries. This aims to improve the temporal relation modeling. To allow the queries to distinguish between the different frame features, a temporal encoding analogous to [48] is used [12].

2.5.2 DN-MOT

DeNoising-MOT [14] builds upon the TrackFormer architecture by introducing denoising training and modifying the interaction between different kinds of queries in the decoder self-attention. It focuses on improving TrackFormer's performance in crowded scenes, where a lot of occlusion occurs.

In the denoising training, DN-MOT uses specific noise to simulate scenarios with occlusion to make the model more robust and perform better in crowded scenes. To modify the self-attention, it employs a cascaded self-attention mask to prevent the suppression of spatial close tracks in crowded scenarios. Moreover, DN-MOT uses MQS as opposed to TrackFormer which learns both positional and content query. Due to this, DN-MOT is based on 4D anchor boxes instead of 2D reference points [14].

Denoising

The role of track queries and object queries remains unchanged in DN-MOT, but denoising queries are added in the same way as in DN-DETR. The novelty of DN-MOT lies in how the various queries are handled and the way in which noise is applied.

The denoising query generation is strongly inspired by DINO, as for each ground-truth object a corresponding positive and negative noise query is generated. Even though, most MOT dataset only contain objects of the same class (e.g. pedestrians), label noise is used nonetheless. The goal is not to be able to differentiate between multiple classes, but to learn different representations of the same class. For track queries, the learned label embedding is replaced with the track queries content embedding. Moreover, instead of applying noise to the ground-truth boxes, for track queries noise is applied to their anchor boxes. The process is shown in Figure 2.24 [14].

In order to generate positive noise, DN-MOT differentiates between scenarios in which other objects are nearby and those in which they are not. If there is no other bounding box which overlaps with IoU $> \tau_{cond}$, then noise is applied by sampling a noise vector $N \in [-\lambda_{pos}, \lambda_{pos}]^4$ and applying it to the bounding box $B = [x, y, w, h]$ as follows:

$$B_{new} = B + B \odot N \tag{2.23}$$

If an objects overlaps with IoU$> \tau_{cond}$, then a conditional noise strategy is employed. In this case, the noisy box is a linear combination of the two overlapping boxes. The strategy is shown in Equation 2.24 [14].

$$B_{new} = \lambda_c B + (1 - \lambda_c) B_n \tag{2.24}$$

,where $\lambda_c \in [0, 1]$ is a hyper-parameter and B_n refers to the neighboring box. The idea behind this strategy is to model occlusion scenarios and make the denoising process more difficult. The negative noise is generated by sampling N from

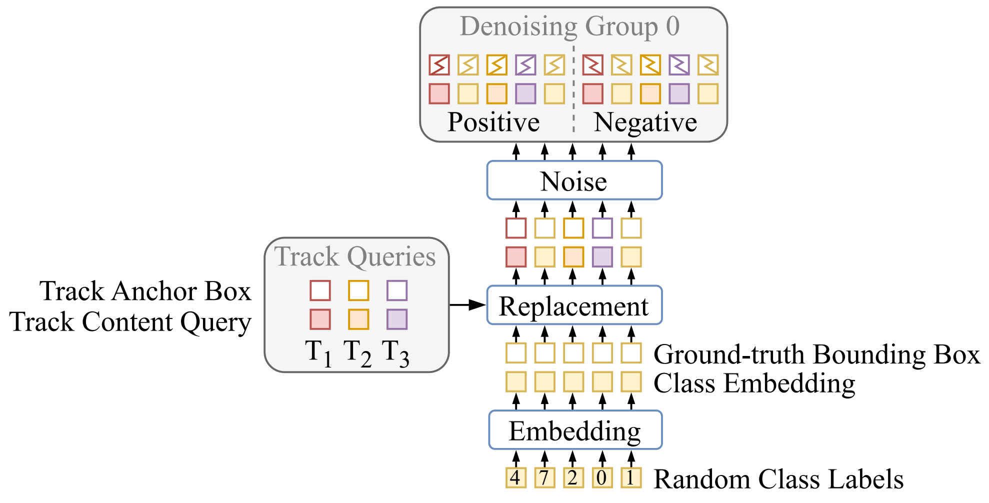


Figure 2.24: Overview of DN-MOT's denoising query generation. Class embeddings are generated by picking random class labels and choosing the corresponding embedding. When a denoising queries' corresponding ground-truth object is already tracked, it is replaced with the corresponding track query. Afterwards, noise is added to the boxes. For the positive denoising queries a lower noise scale is used than for the negative ones (adapted from [14]).

$[-\lambda_{neg}, \lambda_{neg}]^4$ instead, while $\lambda_{neg} > \lambda_{pos}$. Analog to DN-DETR, multiple denoising groups are used [14].

Cascaded Mask Self-Attention

During self-attention, queries interact which each other to exchange information. However, different kinds of queries have different demands during this process [14].

- **Object Queries:** Need to interact with other object queries and track queries to avoid duplicate detections.
- **Track Queries:** Track queries are suppressing other queries in their proximity. This is needed for detection queries, but may lead to the suppression of other track queries. However, inter-track communication can also positively effect the tracking process.
- **Denoising Queries:** Need to be blocked from the object/ track queries and other denoising groups to prevent information leakage (see DN-DETR).

DN-DETR aims to resolve the conflict of the track queries by modifying the attention mask accordingly. In the first half of the decoder layers, all track queries can interact with each other. In this stage information between tracks is exchanged. In the latter half of the layers, track queries can not interact with other track queries. The rationale is to prevent track query suppression. The self-attention mask for the latter layers can be seen in Figure 2.25. The gray parts are blocked, while the colored ones are visible [14].

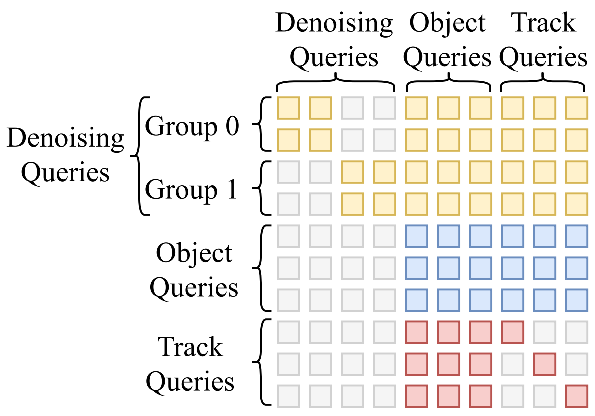


Figure 2.25: DN-MOTs cascaded self-attention mask for the latter layers. The gray parts are blocked, while the colored ones are visible. During the first half of the decoder layers, all track queries can interact with each other. In the latter half, track queries do not see other track queries. This is done to prevent the suppression of track queries in the second half of the decoder, while still allowing information exchange in the first half (adapted from [14]).

2.5.3 MOTR

MOTR [13] is an E2E MOT model, which was developed concurrent with TrackFormer. It handles the bipartite matching between predictions and ground-truth identical to TrackFormer. The main differences between TrackFormer and MOTR are that MOTR uses additional components and that the training is on a sequence of up to 5 images while TrackFormer only trains on two adjacent frames. MOTR adds a Query Interaction Module (QIM) to the DETR architecture. This refines the current track queries q^t_{track} based on the track queries from the previous frame q^{t-1}_{track} before they are passed to frame $t + 1$. MOTR also follows a different loss strategy. While TrackFormer only determines the loss on frame t, it calculates the loss over the whole sequence with Collective Averaging Loss (CAL). Since the loss is calculated over a sequence, detections and tracking are included inherently. This means that MOTR is not dependent on false negatives for joint learning. These two components improve the temporal relation modeling and allows MOTR to waive NMS and Re-ID. The architecture is visualized in Figure 2.26 [13].

Query-Interaction Module

This section describes the QIM. This module handles the initialization of new tracks, the erasure of old tracks and the refinement of current tracks. The refinement is handled by the Temporal Aggregation Network (TAN). An overview of the QIM is given in Figure 2.27 [13].

Initialization & Erasure In a video sequence objects may appear or disappear. To handle this, the QIM filters the object and track queries. During training, tracks are initialized by object queries which are successfully matched via bipartite matching.

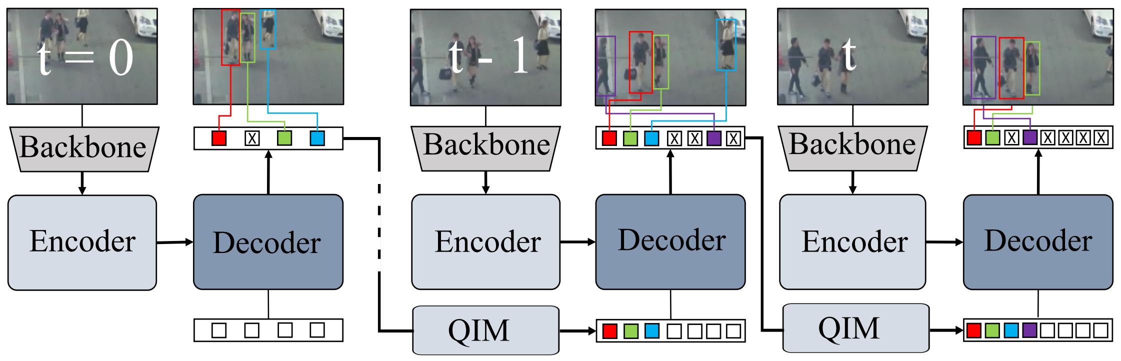


Figure 2.26: Overview of MOTR architecture. In the first frame, tracks are initialized by object queries. This is done in the QIM. Afterwards the track queries are propagated to the next frame and concatenated with the object queries. The queries are then fed into the decoder to either track existing objects or detect new-born ones (adapted from [12]).

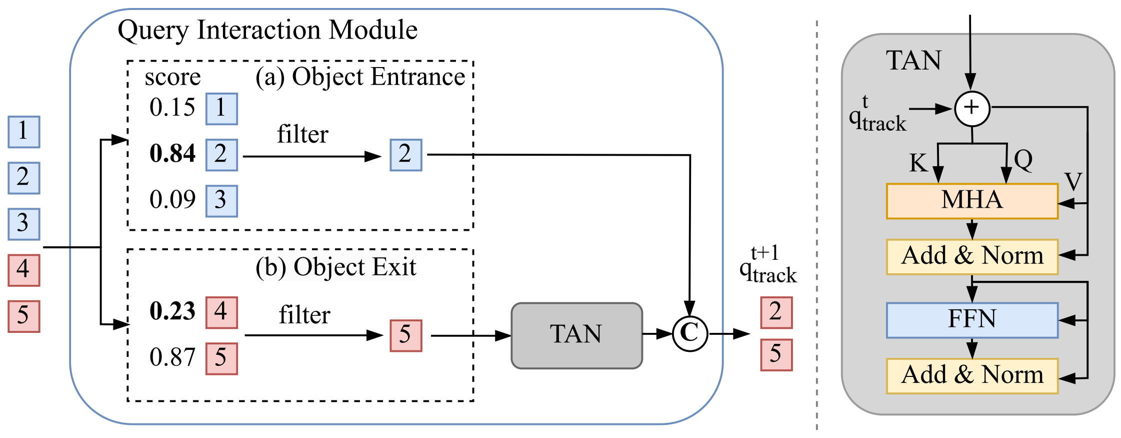


Figure 2.27: Structure of the Query-Interaction Module. The QIM handles track initialization and erasure by filtering the predictions. Object queries are marked in blue and track queries in red. Persisting track queries are refined by the TAN which can be seen on the right (adapted from [13]).

Track queries are deleted when the tracked ground-truth object disappears or the IoU between prediction and ground-truth is lower than 0.5. During inference, the classification score is used. When object queries score above τ_{en}, they initialize a new track and if an existing track scores below τ_{ex} for M consecutive frames, it is deleted [13].

Temporal Aggregation Network This components enhances the temporal relation modeling by processing the current track queries together with the ones from the previous frame. It consists of the cross-attention block of a transformer decoder-layer. The idea behind it is that it provides contextual priors to the tracked objects [13].

Collective Average Loss

MOTR reasons that training within two frames is not able to generate training samples which contain long-range object motion. Therefore, it tales a video sequence as input. This enables it to learn better temporal relation modeling [13].

To train on a sequence of frames, it calculates the loss over the whole sequence at once instead of calculating it on a frame-to-frame basis. This approach is called CAL. CAL is the loss over the sequence of M frames averaged by the number of objects. It is displayed in Equation 2.25 [13].

$$\mathcal{L}_{CAL} = \frac{\sum_{t=1}^{M} \sum_{i=1}^{N_t} \mathcal{L}_{query,(t,i)}}{\sum_{t=1}^{M} V_t} \tag{2.25}$$

,where N_t denotes the number of queries and V_t the number of ground-truth objects in frame t. $\mathcal{L}_{query,(t,i)}$ is the loss for the ith query in the tth frame. It is calculated identical to DETR [13].

2.5.4 MOTRv3

MOTRv3 [47] builds upon MOTR and claims that the poor detection performance occurs due to a conflict between detection and association. It argues that this conflict is caused by an unequal label assignment between track and detection queries. To counteract this, MOTRv3 proposes Release-Fetch-Supervision (RFS). This involves adapting the bipartite matching strategy between queries and ground-truth objects to place a greater emphasis on detection. In addition, two further modifications are presented. Pseudo Label Distillation (PSD) uses an additional object detection model to generate pseudo labels for training. Track Group Denoising (TGD), on the other hand, is a tracking-specific denoising strategy. Both methods are designed to provide more supervision during the training process. The three previously mentioned strategies only affect the decoder during training and are illustrated in Figure 2.28. This section focuses on these methods, starting with RFS [47].

Release-Fetch Supervision

In the conventional MOTR label assignment strategy, track queries are fixed assigned to their corresponding ground-truth objects, while the object queries are assigned to the 'free' ground-truth objects by bipartite matching. RFS weakens this fixed assignment by only performing it in the last decoder layer. The previous layers are now using pure Hungarian matching between all queries (including track queries) and all ground-truth objects [47].

Due to this, all ground-truth object are used to train both object and track queries. At the beginning of the training, the ground-truth objects are mostly assigned to the object queries as the track queries are not able to correctly follow the objects. Then, when the track queries are able to localize the corresponding objects, the labels are automatically fetched back to the track queries [47].

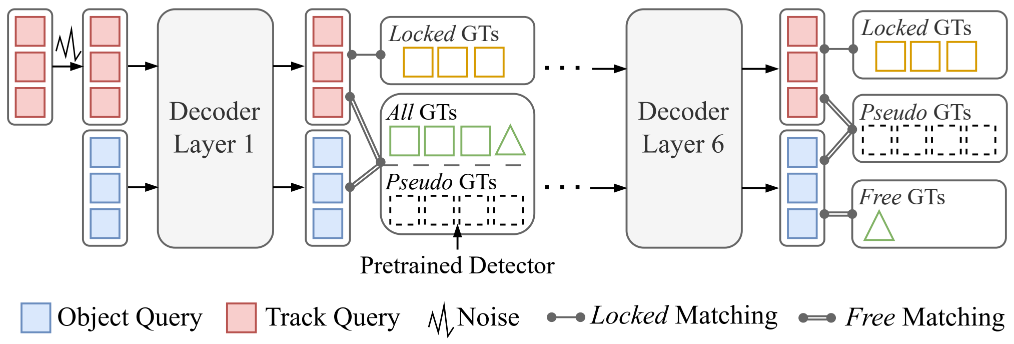


Figure 2.28: Overview of the MOTRv3 architecture. It uses three different matching strategies. The RFS matching between all queries and the ground-truth objects, the fixed assignment between denoising queries and ground-truths and the matching with the pseudo labels (adapted from [47]).

Pseudo Label Distillation

Pseudo Label Distillation (PLD) uses an additional trained object detector (e.g. YOLO-X [49]) to generate pseudo ground-truth objects as an additional supervision during training. Due to the diverse nature of those so called pseudo labels, the MOTR detection part can be trained more efficiently [47].

In PLD, a confidence threshold is used to select suitable candidates from the trained object detector predictions. The selected predictions are then used as pseudo labels and are assigned via bipartite matching to both object and track queries. Queries which are not matched are assigned to the background class. Based on this assignment the detection loss is calculated. As the predictions from the detection model are noisy, the detection loss for each query is weighted with the classification score of the detection model. The calculation of the loss is described in Equation 2.26 [47].

$$\mathcal{L}_{PLD} = \sum_{i=1}^{N} w_i \cdot \mathcal{L}_{query}(y_i, \tilde{y}_{\sigma_P(i)}), \quad w_i = \begin{cases} c_p & , \text{if } \tilde{y}_p \neq \emptyset \\ 0.5 & , \text{else} \end{cases} \tag{2.26}$$

,where $\sigma_P(i)$ denotes the mapping between the i^{th} query and the p^{th} pseudo label $\tilde{y}_p$ and c_p the classification score of the pth pseudo label [47].

Track Group Denoising

While the previous two methods improve the detection performance, TGD aims on boosting the association. Inspired by DN-DETR and Group-DETR [50], multiple noisy versions of each track query are generated. Each noisy version is assigned the corresponding ground-truth of the original track query. As opposed to previous denoising versions, TGD only applies noise to the height and width of the anchor box. Analog to DN-DETR, an attention mask is used to prevent information leakage [47].

Loss

Since all three methods presented assign a separate matching between queries and ground-truth objects or pseudo labels, the final loss is made up of three components. One loss term is calculated based on the RFS matching, one on the matching between the queries and pseudo labels and one by the fixed assignment by TGD. The final loss is over the sequence of M frames is formulated as:

$$\mathcal{L}_{seq} = \frac{\sum_{t=1}^{M} \mathcal{L}_{RFS}(t) + \mathcal{L}_{PLD}(t) + \mathcal{L}_{TGD}(t)}{\sum_{t=1}^{M} V_t} \tag{2.27}$$

,where $\mathcal{L}_{RFS/PLD/TGD}(t)$ denotes the corresponding loss and V_t the number of ground-truth objects in the t^{th} frame [47].

3 Methods

We propose an efficient End-to-End Multiple Object Tracking model, based on TrackFormer's [12] joint learning strategy, which uses historical trajectories to enhance its temporal-relation modeling. Different from most existing methods, that only update spatial information on a frame-to-frame basis, we are proposing to use spatial information from n previous frames to enhance the positional part of the track query. We achieve this by adding a new Trajectory Prediction Module (TPM), which predicts a refined initial spatial position based on the historical position of the tracked object.

Moreover, we are exchanging the Deformable DETR [39] base of TrackFormer [12] with a MQS-based DETR. We propose that the propagation of 4D anchor boxes further enhances the temporal modeling capabilities of the architecture. In addition, an encoder loss is introduced to learn the positional part of the object queries independently from the track query assignment. This lessens the reliance on false negatives to perform joint training.

Inspired by DN-MOT [14] and MOTRv3 [47], we are utilizing a specific denoising strategy to further improve both detection and association performance. Due to these methods, we reason that we no longer rely on TrackFormer's [12] NMS of track queries and the previous frame features to achieve high association performance. This leads to a significant training speed-up over TrackFormer-based models which are already faster than MOTR-based models as these rely on the sequential processing of up to 5 frames during training. Our architecture is visualized in Figure 3.1.

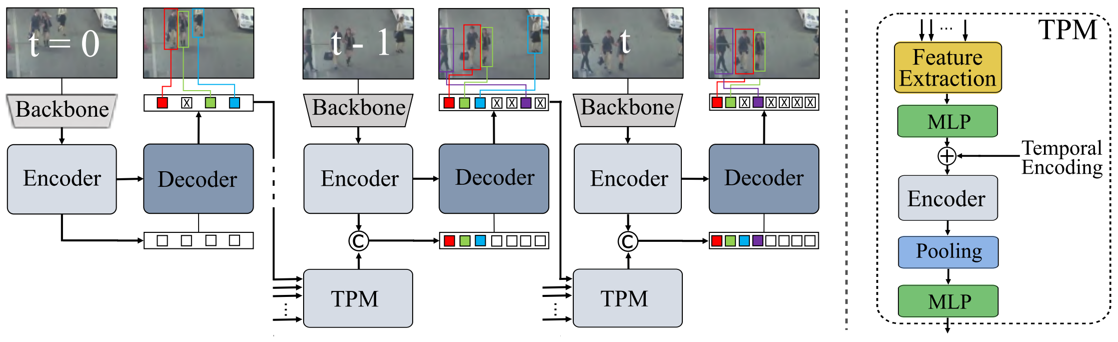


Figure 3.1: Overview of our model architecture. Instead of learning positional queries, we are extracting them from the encoder output (MQS). The TPM is added which predicts the next position of a track query based on the n previous ones. The predicted bounding boxes are combined with the track queries' content part to form the shadow queries. The shadow queries and the original track queries are concatenated and propagated to the next frame (adapted from [12]).

The following sections will present a discussion of the three previously mentioned methods. The initial section will address MQS and its impact on the tracking and detection joint training. Subsequently, our denoising strategy is presented. Finally, the chapter will conclude with an examination of TPM.

3.1 Mixed-Query Selection

Our model exchanges the Deformable DETR [39] base with DINO's [42] MQS. It extracts the positional part of the object queries adaptively from the encoder outputs instead of learning them directly. Analog to DAB-DETR [40], the positional queries are in the form of 4D anchor boxes. We reason that this change should lead to better detection performance due to the adaptively selected positional queries. Moreover, we expect an increase in the association performance as 4D anchor boxes can propagate more spatial information than the standard 2D reference points from Deformable DETR. The generation of the object queries with MQS is visualized in Figure 3.2.

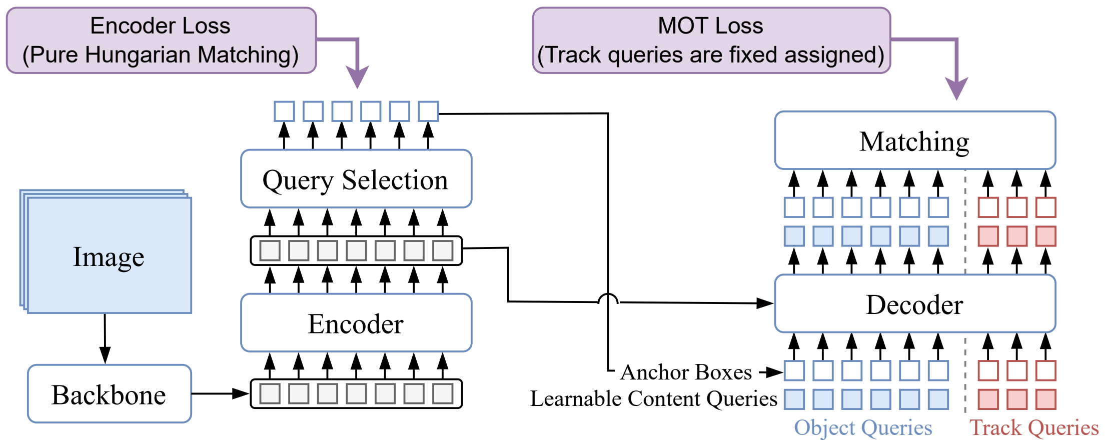


Figure 3.2: Visualization of MQS. Each element from the encoder output is a potential query candidate. By feeding them through two FFNs, a prediction box and score are obtained for each candidate. The predicted boxes of the top-K scoring candidates are chosen as the positional parts of the object queries. The content part is still learned. MQS only affects the object queries explicitly. The track queries are still initialized by successful detections in the previous frame (adapted from [42]).

To teach the model to select good positional queries, a separate encoder loss $\mathcal{L}_{enc}$ is introduced [42]. This loss is calculated between the prediction scores and boxes of the query candidates and all ground-truth objects. Analog to DETR, the predictions and the ground-truth objects are matched by Hungarian matching. There is no differentiation between tracked and untracked objects. $\mathcal{L}_{Enc}$ consists of a box and a class component analog to the normal loss. The box loss is normalized by the number of ground-truth objects inside the batch and the class loss by the number of queries.

$$\mathcal{L}_{enc}(y, \hat{y}) = \sum_{i=1}^{N} \left[\mathcal{L}_{class}(y_i, \hat{y}_{\hat{\sigma}(i)}) + \mathcal{L}_{box}(y_i, \hat{y}_{\hat{\sigma}(i)})\right] \tag{3.1}$$

As a consequence of the pure Hungarian matching in the encoder loss, the positional component of the object queries is trained without reliance on high false negative probabilities p_{FN}. Accordingly, p_{FN} was set to 0.1, with the objective of facilitating more efficient training of the association. A comparison between our DETR base and other models is provided in Table 3.1. Despite the fact that both TrackFormer and its subsequent variation, DN-MOT [14], utilize both the preceding and current frame features within the encoder, our approach diverges from this strategy. We propose that the minimal increase in performance is offset by the increase in computational time and VRAM requirements. As a result, our model does not require temporal encoding to differentiate between frame features. Additionally, our model employs a standard feature dimension of $d_{feat} = 256$, enabling the utilization of pre-trained DETR models as an initial point for training. In contrast, other architectures necessitate the retraining of DETR from scratch, as their architectural modifications are incompatible with existing pre-trained model weights [12, 14].

Table 3.1: Comparison of our DETR base to existing ones. Since we use a feature dimension of 256 and no prior frame features, our architecture can use already trained DETR weights as initialization. Thus, there is no need to pretrain a custom DETR model. '-' denotes unknown values.

Model	DETR-Base	Enc.-Loss	p_{FN}	$d_{feat.}$	Prev. frame feat.
TrackFormer [12]	Def.		0.5	288	✓
DN-MOT [14]	MQS	✓	-	288	✓
MOTR [13]	Def.		0.1	256	
MOTRv3 [47]	Def.		-	256	
CO-MOT [51]	Def.		0.1	256	
MeMOTR [15]	DAB-Def.		0.0	256	
Ours	MQS	✓	0.1	256	

3.2 Denoising

Inspired by DN-MOT [14] and MOTRv2/v3 [47, 52], we use denoising to improve both convergence speed and performance. Due to the low p_{FN} and the fact that we only calculate the loss for frame t, hardly any object queries are matched with ground-truth objects during training. We try to design our denoising strategy to shift focus back onto the object detection. As a result, we improve MOTRv2's [52] denoising strategy by adding label noise and multiple denoising groups. We argue that label noise also makes sense in scenarios with only one class, as it learns different representations of the denoising content query. The noise application of our method is shown in Equation 3.2. $B = (x, y, w, h)$ denotes the ground-truth bounding box and $D = (\frac{w}{2}, \frac{h}{2}, w, h)$ is a scaling vector to guarantee that the new center point is still inside of the old bounding box. It is important that the cropping is performed on the bounding box in (x, y, w, h)-representation, as we want to allow boxes to 'overflow', while still forcing the center point to be inside the image.

$$\begin{aligned} B_{new} &= B + D \odot N \quad , N \in [-\lambda, \lambda]^4 \\ B_{noise} &= \min(\max(B_{new}, 0), 1) \end{aligned} \tag{3.2}$$

Our method does not differentiate between tracked and untracked ground-truth objects. For every single ground-truth object, a denoising query is generated. The bounding box noise is uniformly sampled with a noise scale λ of 0.1. The content part of the denoising queries is a learned embedding. We are randomly choosing a class label from $\{0, ..., 19\}$ and then use the corresponding embedding. Our method employs five denoising groups. Other methods, such as DN-MOT [14] and MOTRv3 [47] apply noise to the track anchor boxes and use the track queries' content part instead of learned denoising embeddings when generating denoising queries for tracked ground-truth objects. Moreover, DN-MOT [14] uses both conditional positive noise and contrastive denoising [42]. Our method does not employ this concepts. We are in principle using DN-DETR's [41] denoising strategy with a suitable bounding box noise scale for the tracking task. In contrast to MOTRv2's [52] method, we utilize label noise and multiple denoising groups. A comparison between the different denoising strategies is given in Table 3.2.

Table 3.2: Comparison of our denoising strategy to existing ones. Our architecture uses DN-DETR's denoising strategy with a tracking-specific noise scale '+' denotes strategies, where noise is only applied to the width and height of the bounding box and λ. '-' denotes unknown values. The terms 'tracked' and 'untracked' describe the manner in which the method addresses the respective ground-truth objects. In the case of 'gt,' the tracked ground-truth object is treated in a manner identical to that of the untracked ones. Conversely, the term 'query' indicates that the noise is applied directly to the track queries. 'cond' and 'contr' refer to conditional positive noise [14] and contrastive denoising [42] respectively.

	tracked							
Model	gt	query	untracked	contr.	cond.	label n.	groups	λ
DN-DETR			✓			✓	5	0.4
DN-MOT		✓	✓	✓	✓	✓	dynamic	0.2
MOTRv2	✓		✓				1	0.1
MOTRv3		✓					4	-
Ours	✓		✓			✓	5	0.1

3.3 Trajectory Prediction Module

Current End-to-End Multiple Object Tracking methods take the bounding boxes from the previous frame as anchor boxes for the track queries. We reason that this method is suboptimal, as it only allows pooling of information at the previous object location instead of the current one. Moreover, it shifts the focus on appearance information in the content query to match objects correctly in spatial ambiguous scenarios. This may lead to false associations in scenes with occlusion and objects with similar appearance. This scenario is shown in Figure 3.3.

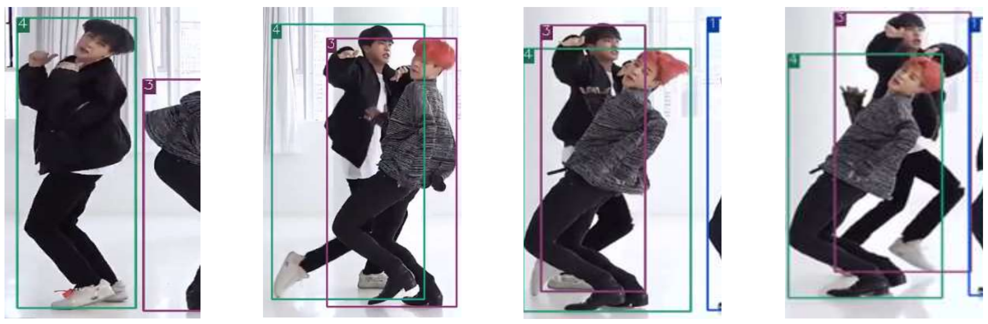


Figure 3.3: Example of false association in scenarios featuring occlusion. Despite the opposing motion of the two objects, the high level of occlusion results in an ID switch between frames 2 and 3 (adapted from [53]).

We propose to use a Trajectory Prediction Module to predict a track anchor box, which takes previous motion behavior into consideration. Instead of taking the bounding box from the previous detection, we predict the next bounding box based on the previous ones. For linear motion, as in MOT17 [1], a Kalman filter [54] is a suitable motion prediction. However, it fails in scenarios with complex nonlinear motion patterns as in DanceTrack [3].

Inspired by MotionTrack [53], we propose a transformer encoder-based prediction module which takes the n previous locations to predict the next one. Since this is a learning-based component, unlike the Kalman filter [54], we have to train the module. In order to continue to act in the spirit of E2E methods, we joint train the module together with the rest of our architecture. As we only have access to two frames during training and therefore only have one previous location for each track, we are generating pseudo historical trajectories by applying noise to previous ground-truth boxes.

Instead of only supplying the model with the predicted anchor boxes directly, we are generating an alternative version of each track query which uses the predicted anchor box instead of the previous one. Both versions are sharing the same content query. Inspired from CO-MOT [51], we are referring to these alternative queries as shadow queries. Each tracked objects then has one corresponding track and shadow query. During training, both version are matched with the same ground truth. The training process is visualized in Figure 3.4.

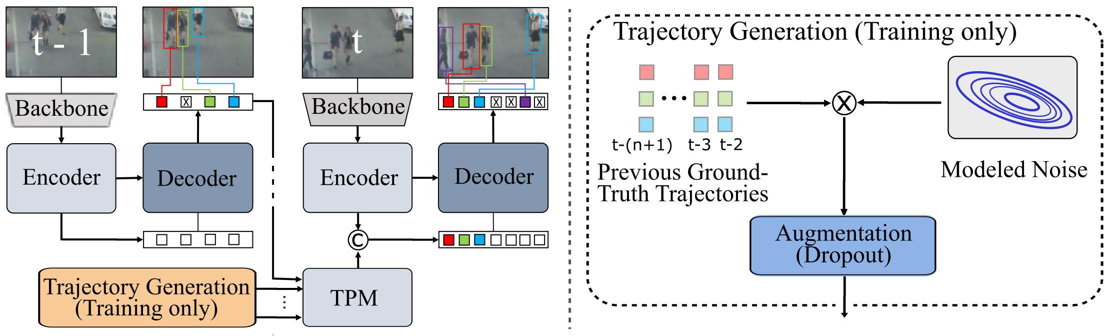

t - 1
t
Backbone
Encoder
Decoder
Backbone
Encoder
Decoder
Trajectory Generation (Training only)
TPM
Trajectory Generation (Training only)
t-(n+1) t-3 t-2
Previous Ground-Truth Trajectories
Modeled Noise
Augmentation (Dropout)

Figure 3.4: Overview of the Trajectory Prediction Module during training. The TPM processes the $n + 1$ locations of the object to predict the bounding box in the next frame. It generates a shadow query for each track query, which uses the predicted bounding box as an anchor box. During training, the historical trajectories are artificially generated by applying noise to the previous ground-truth boxes (adapted from [12]).

During inference, the model picks either the shadow query or the track query for each track based on the prediction score. The individual components of TPM are described in detail in the following subsections.

3.3.1 Architecture

The Trajectory Prediction Module consists of several components. We use a noise module that adaptively models the noise during training and then applies it to the ground-truth bounding boxes. The feature extraction module then extracts a series of features from the historical trajectory. These features are then passed to the prediction module, which predicts the next bounding box. Instead of predicting the box directly, the offset to the previous box is predicted. In this way, the previous box serves as a positional bias, which simplifies the problem. A detailed view of the module architecture is shown in Figure 3.5.

Noise Modulation

To realistically simulate the historical trajectories during training, we use multivariate Gaussian noise, which models the prediction behavior of our architecture. During training, we store the last 1000 prediction-ground-truth pairs and compute the relative error for each pair using Equation 3.3. The error function is chosen so that 0 elements in the ground-truth (possible for the center point) are not leading to undefined expressions. ϵ is added for numerical stability.

$$err_i = 2 \cdot \frac{b_i - \hat{b}_i}{|b_i| + |\hat{b}_i| + \epsilon} \tag{3.3}$$

Given that the noise is applied by multiplication to the ground-truth boxes, we add 1 to the relative error in order to obtain an error factor. In the next step, the noise vector err is concatenated with the predicted scores. We do this, because

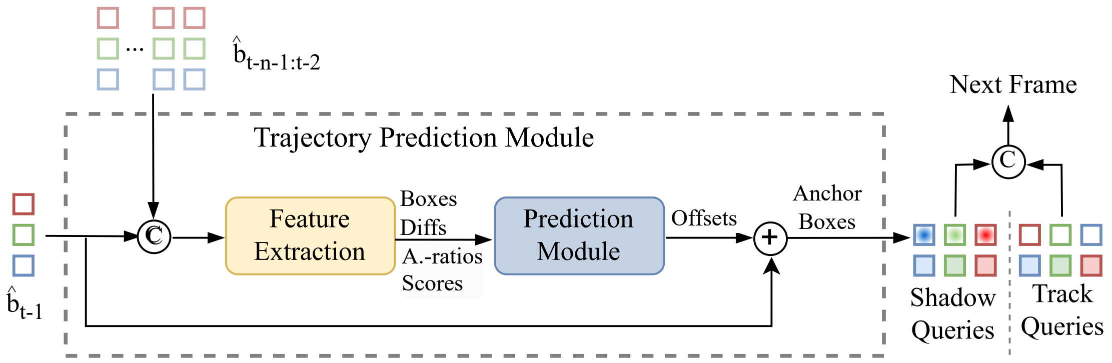


Figure 3.5: Architecture of the Trajectory Prediction Module. The Trajectory Prediction Module concatenates the $n+1$ previous (predicted) bounding boxes to form historical trajectories. Features are extracted for each bounding box at each time step. The features are then fed through the prediction module to generate a bounding box offsets relative to the bounding boxes $\hat{b}_{t-1}$. By adding the offsets to $\hat{b}_{t-1}$, the predicted bounding box for the next frame is generated. We take this prediction and use it as an anchor box for alternative versions of the track queries, so called shadow queries (inspired from [51]). The track and shadow queries are then propagated to the next frame.

we expect that there is a relation between the quality of the prediction box and the predicted score. Then we fit a multivariate Gaussian to the values. This is done in each iteration of our model. We generate the noise by sampling a 5-dimensional vector from the distribution $\mathcal{N}(\mu, \Sigma)$ for each bounding box. The first four dimensions are the relative errors, which we use as noise and the last dimension is a prediction score.

$$\begin{aligned} data &= \text{concat}(err + 1, scores) \\ \mu &= \text{mean}(data) \\ \Sigma &= \text{cov}(data) \\ \text{noise model} &= \mathcal{N}(\mu, \Sigma) \end{aligned} \tag{3.4}$$

Trajectory Generation

During training, we are sampling the previous frame from the interval $[t-5, t+5]$. We are using the same sampling distance $d \in [t-5, t+5]$ between the current frame and the previous frame to uniformly sample n more previous time steps. Accordingly, all time steps in the trajectory are equally spaced. The ground-truth boxes are then extracted from each of these n time steps. By grouping the ground-truth boxes according to their object ID, we obtain the ground-truth trajectories. Missing objects in certain frames are padded with zeros. By sampling noise from $\mathcal{N}(\mu, \Sigma)$ and multiplying it with the ground-truth trajectories, we obtain the pseudo historical trajectories. Furthermore, we also obtain prediction scores by sampling from $\mathcal{N}(\mu, \Sigma)$, which we concatenate with the trajectories. As we are already using noise which models the actual prediction characteristic, we reason that augmentations like false positives or spatial jitter are already handled implicitly. We are only using false negative augmentation, where we exchange random boxes in the trajectory with zeros. In the final step, the actual predictions from the previous frame are concatenated with the pseudo historical trajectories. The historical trajectory for a single tracked object can be denoted as $\mathcal{T} = \big(x_{t-d(n+1)}, .., x_{t-d}\big) \in \mathbb{R}^{(n+1)\times 5}$.

Feature Extraction

Currently, the historical trajectories consists of a bounding box and a score for each element. Inspired by MotionTrack [53], we are extracting the aspect ratio and the normalized offset to the previous bounding box ('velocity') according to Equation 3.5. The offset is calculated between two non-zero bounding boxes. Due to false negatives, there can be a gap of length $n \neq 1$ between two nonzero boxes. Accordingly, the term n^{-1} is employed for normalization purposes.

$$\Delta_{x,y,w,h}(t) = \frac{1}{n}\left[\hat{b}_{x,y,w,h}(t) - \hat{b}_{x,y,w,h}(t-n)\right] \tag{3.5}$$

The new features are then concatenated with the existing ones for a final feature dimension of 10. The representation of the object at a single time step $t-1$ is then denoted as $x_{t-1} = (x, y, w, h, \Delta x, \Delta y, \Delta w, \Delta h, \text{aspect ratio}, \text{score})$.

Prediction Module

The prediction module is a transformer encoder. The extracted features are passed through a MLP to gain a 256-dimensional representation for each time step and track. Then a temporal encoding is added to distinguish the different time steps. Afterwards, the trajectories are passed through the encoder. The encoder outputs are average-pooled along the time domain and fed through another MLP to gain the prediction offsets $\hat{\delta}_t = (\hat{\delta}_x, \hat{\delta}_y, \hat{\delta}_w, \hat{\delta}_h)$. A self-attention mask is used such that tracks are only able to see themselves. Similar to MotionTrack, there is no inter-frame communication. The structure of the module is illustrated in Figure 3.6.

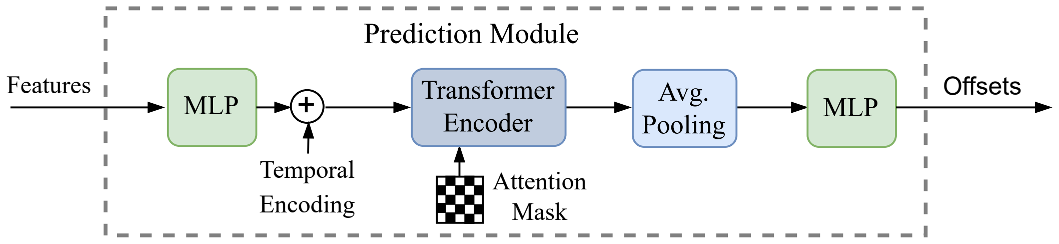


Figure 3.6: Structure of the prediction module. The module is based on a transformer encoder. A temporal encoding is then added to distinguish between different time steps. The encoder employs an attention mask to ensure that there is no inter-track communication. As a result of the mask, a track-specific encoding is not necessary. The information is pooled track-wise before being fed through the MLP to predict the final offsets.

3.3.2 Shadow Queries

Inspired by CO-MOT [51], we utilize shadow queries to integrate the predicted anchor boxes. For each track query, a shadow query is generated that has the same content part but a new predicted anchor box as a positional query. During training, the shadow queries are matched in an identical manner to their corresponding track queries. Consequently, we no longer have a one-to-one matching between track queries and ground-truth objects, but a one-to-set matching. This allows to learn different representations for the same object, thereby accelerating training as more track predictions are backpropagated. The matching of object queries to untracked ground-truth objects is still conducted on a one-to-one basis. To avoid duplicate predictions during inference, the prediction with the higher prediction score is selected from a track-shadow query pair.

3.3.3 Prediction Loss

The TPM is trained with two distinct losses: The decoder losses ($\mathcal{L}_{class}$ & $\mathcal{L}_{box}$) and a prediction loss $\mathcal{L}_{pred}$. The prediction loss is based on the difference between the prediction and the ground-truth bounding box. Analogous to MotionTrack, the smooth l$_1$ loss is employed. The prediction loss is calculated according to Equation 3.6.

$$\mathcal{L}_{pred} = \sum_{i \in \{x,y,w,h\}} l_{1,smooth}\left(b_i(t) - \left[\hat{b}_i(t-1) + \hat{\delta}_i\right]\right) \tag{3.6}$$

,where Equation 3.7 denotes the smooth l$_1$ loss.

$$l_{1,smooth}(x) = \begin{cases} \frac{0.5}{\beta} \cdot x^2 & \text{,if } |x| < \beta \\ |x| - 0.5 \cdot \beta & \text{,otherwise} \end{cases} \quad \text{,with } \beta = 1 \tag{3.7}$$

3.4 Losses

Given that each additional component introduces a distinct loss, the model in conjunction with the TrackFormer loss is characterized by four losses. These include the TrackFormer loss $\mathcal{L}_{MOT}$ after each decoder layer, the encoder loss $\mathcal{L}_{enc}$ designed to extract higher quality query candidates for MQS, the denoising loss $\mathcal{L}_{DN}$ and the prediction loss $\mathcal{L}_{pred}$. The total model loss is displayed in Equation 3.8.

$$\mathcal{L}_{total} = \mathcal{L}_{MOT} + \mathcal{L}_{enc} + \mathcal{L}_{DN} + \mathcal{L}_{pred} \tag{3.8}$$

The various losses differ primarily in the manner in which the matching between prediction and ground-truth is determined, as well as their individual loss components. A summary is given in Table 3.3.

Table 3.3: Summary of all used losses in our model. In addition to the standard TrackFormer loss $\mathcal{L}_{MOT}$, our model employs three further task-specific loss terms.

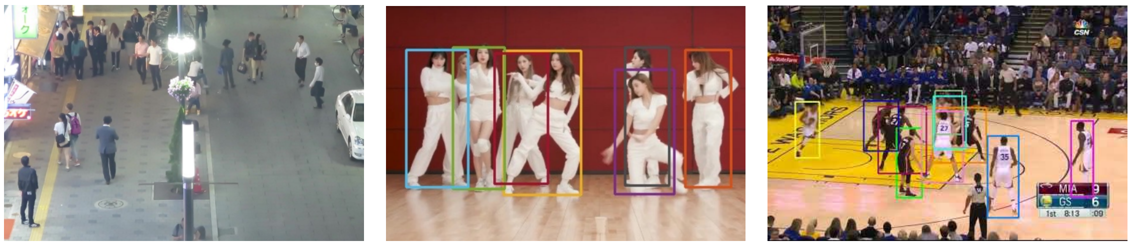
Loss	Matching	Components
$\mathcal{L}_{MOT}$	untracked: hungarian; tracked: fixed	$\mathcal{L}_{focal}, \mathcal{L}_{l1}, \mathcal{L}_{giou}$
$\mathcal{L}_{enc}$	hungarian	$\mathcal{L}_{focal}, \mathcal{L}_{l1}, \mathcal{L}_{giou}$
$\mathcal{L}_{DN}$	fixed (based on ground-truth box)	$\mathcal{L}_{focal}, \mathcal{L}_{l1}, \mathcal{L}_{giou}$
$\mathcal{L}_{pred}$	fixed (based on track ID)	$\mathcal{L}_{\text{smooth\_}l1}$

4 Experiments

This chapter presents an overview of the experimental setups. Initially, the datasets and metrics utilized in the experiments are examined. Subsequently, the implementation methodology is outlined. The implementation section provides comprehensive descriptions of the experimental setups.

4.1 Datasets

The model is evaluated on MOT17 [1], DanceTrack [3] and SportsMOT [4]. Each of these datasets exhibits a distinct characteristic with regard to scenario, appearance and motion pattern. Example images from all three datasets are displayed in Figure 4.1.



(a) MOT17. This dataset features crowded pedestrian scenes in urban scenarios [1].

(b) DanceTrack. DanceTrack contains mostly group dancing videos [3].

(c) SportsMOT. SportsMOT consists of football, basketball and volleyball sequences [4].

Figure 4.1: Visualization of the utilized datasets. We test our model on MOT17, DanceTrack and SportsMOT.

MOT17 It consists of 14 sequences with crowded scenarios, featuring different viewpoints, camera motion and diverse weather conditions. It has multiple labeled classes, but only pedestrian tracking will be evaluated. The pedestrians are diverse in appearance and exhibit mostly linear motion. Due to the high density of pedestrians, MOT17 features a considerable number of cases of occlusion [1].

DanceTrack DanceTrack is a multi-human tracking dataset which contains mostly group dancing videos. The dancers have a uniform appearance, due to similar or identical clothes, and complex motion patterns. It has frequent occlusion and body deformations. This makes the association challenging [3].

SportsMOT SportsMOT is a multi-object tracking dataset of diverse sports scenes. It consists of basketball, volleyball and football videos. All players in the court are supposed to be tracked. SportsMOT is characterized by variable speed motion and similar appearance. The statistics for all previously discussed datasets are displayed in Table 4.1 [4].

Table 4.1: Statistics for used datasets. Split ratio denotes how many sequences are in each split. Average lengths and tracks refers to the average length and amount of tracks in a sequence.

	Datasets		
Metrics	MOT17	DanceTrack	SportsMOT
splits	train/test	train/val/test	train/val/test
split ratios	7/7	40/25/35	45/45/150
sequences	14	100	240
frames	11,235	105,855	150,379
length (s)	463	5,292	6,015
tracks	1,342	996	3,401
avg. length (s)	33.1	52.9	25.1
avg. tracks	95.9	10.0	14.2
frames per second	14-30	20	25
resolution	480-1080p	720-1080p	720p

4.2 Metrics

We are using the HOTA metrics [29], CLEAR metrics [27], Identity metrics [28] and IDSW [26] for evaluation. The evaluation of the association is conducted using AssA, IDF$_1$, and IDSW. The focus is on AssA. The assessment of detection performance employs DetA and MOTA. DetA is the preferred metric as it exclusively evaluates detection. HOTA is the overall metric, encompassing both subtasks. The use of MOTA and IDF$_1$ is primarily for the purpose of enhancing comparability, as some existing trackers utilize only the CLEAR metrics for evaluation.

4.3 Implementation Details

The model implementation is primarily based on the official DN-DETR repository [55] and the official TrackFormer repository [56]. It utilizes the DINO implementation from the official DN-DETR repository and also its pretrained weights. Instead of employing DN-DETR's multi-scale deformable attention module, we are using the MMCV [57] implementation. For evaluation, TrackEval [58] is utilized.

4.3.1 High-Resolution Tests

In the final high-resolution tests, a batch size of 8 is employed in conjunction with gradient accumulation. The AdamW [59] optimizer is used with an initial learning rate of $5 \cdot 10^{-5}$ and a weight decay of $1 \cdot 10^{-4}$, whereas the backbone and the MHA input projections utilize a learning rate of $5 \cdot 10^{-6}$. The learning rate is reduced by a factor of 10 for the final 10 epochs. The dropout ratio for the attention mechanism is set to zero. The model is trained for 50, 20, and 28 epochs on the MOT17 [1], DanceTrack [3], and SportsMOT [4] datasets, respectively. All high-resolution tests are conducted using PyTorch in conjunction with a single NVIDIA L40s GPU.

The track augmentations employed in this study are those proposed by TrackFormer, which sample the previous frame from the interval $[t-5, t+5]$. The false negative and false positive probability are set to 0.1. The image augmentations of MOTRv2/CO-MOT [51, 52] are employed. During training, the images are scaled so that the longest side is 1536 pixels long, and the aspect ratio remains constant. Conversely, during inference, the images are scaled to 1333 pixels. The image sizes are chosen to ensure comparability with other E2E methods. A complete list of the hyperparameter choices are given in Appendix B.

4.3.2 Ablation Study

In order to test a large selection of different configurations, an ablation study is conducted on a small dataset, such as MOT17 [1]. As MOT17 has only one train and test split, and the test split ground-truth is not publicly available, the train split is divided into two parts. The initial 50% of each sequence constitutes the training set, while the remaining 50% is designated as the validation set. The process is visualized in Figure 4.2

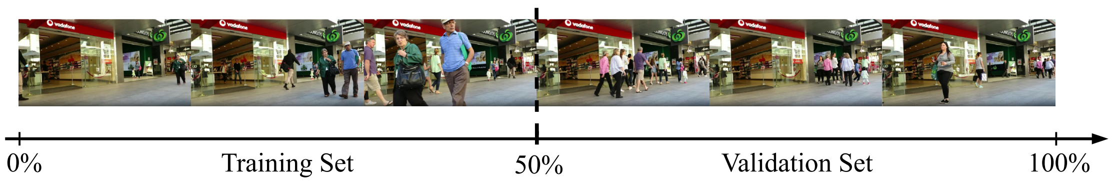


Figure 4.2: MOT17 ablation split. MOT17 does not contain a validation set. During the ablation tests, we take the first 50% of each sequence in the train set for training and the latter half for validation.

Since TrackFormer [12] has to stack its track queries for a batch size > 1, the number of track queries is cut to the lowest number in the batch. This significantly reduces the association performance. Normally the problem is circumvented by processing only one frame pair at a time on several Graphics Processing Units (GPUs) in parallel. Due to limited hardware, only one GPU is used during the ablation study and the process is simulated using gradient accumulation. All ablation experiments on MOT17 are performed on NVIDIA RTX 2080 Ti, while the ones on DanceTrack are done on NVIDIA L40s. A batch size of 8 is employed in all ablation experiments. The input images are resized so that the longest side is 1000 pixels long, maintaining the aspect ratio. The images are normalized using the ImageNet [33] statistics. During training, multi-scale cropping and horizontal flipping are used as image augmentations analogous to TrackFormer. The track augmentations are also adapted from TrackFormer. The false positive probability is kept at 0.1 and the previous frame is randomly sampled from the interval $[t-5, t+5]$. The models are trained for 50 epochs on the MOT17 ablation train split and are tested on the MOT17 ablation validation split. For the tests on DanceTrack, we are training for 20 epochs on the train set and evaluate the performance on the validation set. For the last 10 epochs, the learning rate is dropped by a factor of 10. An overview of the ablation configurations is given in Table 4.2.

Table 4.2: Ablation study configurations. All tests are performed on the best configuration of the previous experiment

	Optimization Target	Datasets
(1)	DETR base	MOT17
(2)	Conflict resolution strategy & p_{FN}	MOT17, DanceTrack
(3)	Denoising strategy	MOT17
(4)	Noise scale λ_{box}	
(5)	TPM inclusion strategy	MOT17
(6)	TPM training strategy	
(7)	Query masking	
(8)	Quality Focal Loss	
(9)	Image augmentations	MOT17, DanceTrack
(10)	Inference hyperparameter	MOT17, DanceTrack

DETR Bases

In the first set of ablation experiments, different DETR bases are compared. Deformable DETR, its two-stage version, DAB-Deformable DETR and DINO's MQS are tested. To ensure comparability, all models utilize 300 object queries. The comparison is limited to the query selection and processing components. Consequently, only DINO's MQS is included in the comparison, and other components, such as CDN or LFT, are not considered. The learning rate is set to $5 \cdot 10^{-5}$ for the transformer and $5 \cdot 10^{-6}$ for the backbone, as this has demonstrated the best results in previous experiments. All DETR models are initialized with the corresponding official weights from object detection training on MS COCO 2017 [45].

Conflict Resolution

In these ablation tests, the effect of various detection-tracking conflict resolution methods in combination with different false negative probabilities p_{FN} is investigated. We are testing the following methods:

- Gradient Coordination (GC): This method was proposed in OneTrack [60], which is a 3D E2E MOT model. It introduces an additional classification head which matches the object queries to all ground-truth objects. Based on this assignment and the original assignment, gradients of predictions which are only matched in one of these heads, are cut. The idea is that by removing contradicting gradients, the conflict between detection and tracking is resolved.

- Coopetition Label Assignment (COLA): COLA was introduced by CO-MOT [51] and changes the matching process in the first 5 decoder layers. Instead of only matching untracked ground-truth objects to the object queries, all ground-truth objects are matched with them. The idea behind this method is that the matched object queries can enhance the representation of the corresponding track queries.

- RFS: RFS was introduced by MOTRv3 (subsection 2.5.4) and also modifies the matching process in the first 5 decoder layers. In these layers all queries and ground-truth objects are solely matched based on Hungarian matching. This is supposed to lead to a focus on detection in the beginning and shifts back to tracking, when the track queries are achieving better predictions.

All tests are performed on the best DETR base from the previous ablation test. Promising configurations are in addition tested on DanceTrack as MOT17 is not suited to highlight the difference in association performance due to its simple motion patterns.

Denoising Strategies

In this ablation test stage, the previously best configuration is taken and different denoising strategies are tested on top of it. We are comparing DN-MOT's [14], MOTRv2's [52] and MOTRv3's [47] denoising strategies with our own method. To ensure a fair comparison, all methods are using 5 denoising groups and a box noise scale λ of 0.1. After the best denoising strategy was chosen, different box noise scales are compared. All test are performed on MOT17 validation split.

Trajectory Prediction Module

During these experiments, we are taking the best performing previous model and are testing our TPM on it. We are comparing different integration and training strategies as well as different other influencing factors. Most tests are performed on the MOT17 validation split and some selected ones on Dancetrack.

Image Augmentations

Most E2E MOT models are based on either TrackFormer or MOTR and have adopted many components of their respective base models, including the image transformations. As a result, there are currently two different sets of image augmentations. These augmentations differ mostly in the choice of parameters, the methods themselves are except for MOTR's colorspace augmentations identical. The training augmentations are shown in Listing 4.1.

```
T.MotCompose([
    T.MotRandomHorizontalFlip(),
    T.MotRandomSelect(
        T.MotRandomResize(scales, max_size),
        T.MotCompose([
            T.MotRandomResize(resizes),
            T.FixedMotRandomCrop(crops),
            T.MotRandomResize(scales, max_size=1536),
        ])
    ),
    T.MOTHSV(),
    normalize])
```

Listing 4.1: Structure of used image augmentations. 'MOTHSV' modifies the colorspace in Hue-Value-Saturation (HSV) format. It add/subtracts random values from each channel.

All augmentations are applied to frame t and $t-1$ identically. The different parameters for the augmentations are displayed in Table 4.3.

Table 4.3: Comparison of image augmentations. TrackFormer-based and MOTR-based models are using different augmentation parameters. HSV augmentations are only used in MOTR-based archtectures.

	TrackFormer	MOTR
horizontal flip	✓	✓
max_size	1333	1536
scales	[480, ..., 800]	[608, ..., 992]
resizes	[400, 500, 600]	[800, 1000, 1200]
crops	(384, 600)	(800, 1200)
HSV		✓

These augmentations are only applied to the train set. For the validation/test set all models try to resize the shortest image side to 800px, while limiting the longest side to 1333px and keeping the aspect ratio constant. To check the influence of the augmentations, we test our final model on both possible configurations before proceeding to the high-resolution tests. This experiment is performed on the MOT17 and DanceTrack validation set. As we are using a maximum size of 1000px during the ablation experiments, all values are scaled down accordingly.

Inference Hyperparameter

Regardless of the training hyperparameters such as learning rate or batch size, our model has inference-specific hyperparameters. These are the thresholds for detection τ_{det}, track initialization τ_{track} and track reidentification τ_{reid}, as well as the number of frames a track can be inactive before it is deleted T_{reid}. Given the numerous alterations made to the architectural design and parameter settings, it is no longer reasonable to assume that the tracking hyperparameters employed by TrackFormer remain the optimal selection for our model. Since no retraining is necessary to optimize these parameters, we can use an iterative optimizer. We use Optuna [61] for the optimization. As sampler we choose 'TPESampler' and optimize for 50 iterations on MOT17 and 20 on DanceTrack. As optimization criterion we use the HOTA score. The optimal obtained hyperparameters are rounded to 0.05 increments. The ranges of the hyperparameters are listed below.

- $\tau_{det} \in [0, 0.9]$
- $\tau_{track} \in [0, 0.9]$
- $\tau_{reid} \in [0, 0.9]$
- $T_{reid} \in \{0, 1, .., 20\}$

Different components on DanceTrack

In this experiments, we are testing the effect of each newly introduced component/modification on DanceTrack. We are comparing the performance change due to MQS, lower p_{FN}, our denoising strategy, the TPM and the image augmentations.

4.3.3 Training Time and Memory Requirement

For the measurements of the computational time and the VRAM requirements, we are using the configurations from the high-resolution tests (subsection 4.3.1). We are training the model for one epoch on the MOT17 train set to get the training time per iteration and the maximum training VRAM requirement. For the values during inference, we are tracking all sequences in the MOT17 train set once. Due to the about 5000 frames in the train set, we get a reliable estimate for the time per iteration. To calculate the VRAM requirement, we employ '*torch.cuda.max_memory_allocated()*' and log the maximum value encountered during the epoch.

In order to facilitate a comparison of the computational time and VRAM requirement of our model with those of other E2E models, we proceed in a manner analogous to that previously described. However, due to compatibility issues, the tests are conducted on an NVIDIA RTX Titan. To ensure greater comparability, a batch size of 1 is employed for all models, and the longest side of the image is scaled to 1536 pixels. In the case of MOTR, all tests are conducted with a sequence length of five.

5 Results

5.1 High-Resolution Tests

This section presents the results of our model, which was trained with a maximum image size of 1536 pixels and tested with a maximum image size of 1333 pixels. The model is being tested on the MOT17, DanceTrack, and SportsMOT datasets. Table 5.1 presents a comparative analysis of the performance of our model with other state-of-the-art models on the MOT17 test split.

Table 5.1: Final results on MOT17 test split. '*' denotes the usage of ConvNeXt-B [62] backbone instead of ResNet-50 [16], '+' joint training on the CrowdHuman [63] dataset and '†' the usage of a pretrained object detector for pseudo label generation during training. '↑'/'↓' indicates that a higher/lower score is better. Tracking-by-Detection models perform significantly better than End-to-End models on MOT17 due to the focus on detection and the simple motion patterns. Our architecture performs worse than other End-to-End models. We reason that this is due to our renounce of additional dataset and the short training schedule. The best results are highlighted in bold.

	Metrics					
Model	HOTA ↑	DetA ↑	AssA ↑	MOTA ↑	IDF_1 ↑	IDSW ↓
	Tracking-by-Detection					
ByteTrack [64]	63.1	62.0	-	**80.3**	**77.3**	2,277
OC-SORT [65]	**63.2**	**63.2**	-	78.0	77.5	**1,950**
ETTrack [66]	61.9	60.5	-	79.0	75.9	2,118
	End-to-End					
TrackFormer^{+} [12]	57.3	-	-	74.1	68.0	2,829
DN-MOT^{+} [14]	58.0	-	-	75.6	68.1	2,529
MOTR^{+} [13]	57.8	60.3	55.7	73.4	68.6	2,439
$\text{MOTRv3}^{*,\dagger}$ [47]	60.2	62.1	58.7	75.9	72.4	2,403
MeMOTR^{+} [15]	58.8	59.6	58.4	72.8	71.5	-
CO-MOT [51]	60.1	-	60.6	72.6	72.7	-
Ours	56.5	58.7	55.0	71.9	68.4	2,664

It can be seen that Tracking-by-Detection (TbD) models perform significantly better than End-to-End models on MOT17. This is because MOT17 has a strong focus on detection, whereas the association following the simple motion patterns of the pedestrians is comparatively simple. As a result, TbD models that use a strong detector in the first step and then exploit the a priori knowledge of the motion pattern perform significantly better than End-to-End models. Another disadvantage for E2E architectures on MOT17 is the size of the training set. Since MOT17 train only includes 5,316 frames, the performance of the data-intense transformer models suffers. Our architecture performs worse on MOT17 than other DETR-based models. We claim that this is primarily due to the length of the training and the fact that we do not use any additional datasets. Table 5.2 compares the datasets used and the total training iterations of different End-to-End models. It can be seen that our model makes 4-10 times fewer iterations during training than comparable models, apart from MOTRv3, which leverages a pretrained object detector to speed up convergence.

Table 5.2: Training configurations on MOT17. '†' denotes the usage of a pretrained object detector for pseudo label generation during training. Our model is trained for 4-10 times less iterations than other End-to-End models except for MOTRv3, which leverages a pretrained object detector to speed up convergence.

Model	Epochs	Datasets	Training Iterations
TrackFormer	85 + 40	CH/ MOT17 + CH	$2.54 \cdot 10^6$
DN-MOT	80 + 40	CH/ MOT17	$1.76 \cdot 10^6$
MOTR	200	CH + MOT17	$1.94 \cdot 10^6$
MOTRv3†	50	MOT17	$\mathbf{0.27 \cdot 10^6}$
MeMOTR	130	CH + MOT17	$1.26 \cdot 10^6$
CO-MOT	200	MOT17	$1.06 \cdot 10^6$
Ours	50	MOT17	$\mathbf{0.27 \cdot 10^6}$

In order to evaluate the efficacy of our model in terms of its association performance, we evaluate it on the DanceTrack dataset. Given the comparatively simple detection and complex motion patterns, the objective of this benchmark is to assess the association performance. The results on the DanceTrack test split are listed in Table 5.3. E2E models demonstrate markedly improved performance on DanceTrack, reflecting a pronounced emphasis on association. The most successful models are all based on the MOTR architecture, which is both time and hardware intensive because of training on a sequence of five frames. It was not possible for more lightweight models, such as DN-MOT, which train on only two frames, to match this performance. Our model demonstrates that architectures that train on a sequence of only two frames can achieve competitive results on the DanceTrack dataset by improving the performance of the previously best two training frame model DN-MOT by 8.1 HOTA.

Table 5.3: Final results on DanceTrack test split. '*' denotes the usage of ConvNeXt-B [62] backbone instead of ResNet-50 [16] and $^+$ additional training on the CrowdHuman [63] dataset. End-to-End models are achieving significantly better results due to their better association performance in scenarios featuring complex motion pattern. The best performing models are based on the MOTR architecture. Our model achieves the best results among E2E models, which are trained on two frames, beating DN-MOT by 8.1 HOTA. The best results are highlighted in bold.

	Metrics					
Model	HOTA ↑	DetA ↑	AssA ↑	MOTA ↑	IDF$_1$ ↑	IDSW ↓
			Tracking-by-Detection			
ByteTrack	47.7	71.0	32.1	89.6	53.9	-
OC-SORT	55.1	80.3	38.3	92.0	54.6	-
ETTrack	56.4	81.7	39.1	92.2	57.5	-
			End-to-End			
DN-MOT	53.5	-	-	89.1	49.7	-
MOTR	54.2	73.5	40.2	79.7	51.5	-
MOTRv3	68.3	-	-	91.7	70.1	-
MOTRv3*	**70.4**	**83.8**	**59.3**	**92.9**	**72.3**	-
MeMOTR	68.5	80.5	58.4	89.9	71.2	-
CO-MOT	65.3	80.1	53.5	89.3	66.5	-
Ours	61.6	80.3	47.5	90.3	61.8	1,244

At last, we assess the performance of our model on the SportsMOT test set, the results of which are presented in Table 5.4. It is evident that Tracking-by-Detection models demonstrate superior performance in both detection and association.

Table 5.4: Final results on SportsMOT test split. Specialized Tracking-by-Detection models achieve the best results. Our models is not able to match MeMOTR's performance due to worse association performance. The best results are highlighted in bold.

	Metrics					
Model	HOTA ↑	DetA ↑	AssA ↑	MOTA ↑	IDF$_1$ ↑	IDSW ↓
			Tracking-by-Detection			
ByteTrack	62.8	77.1	51.2	94.1	69.8	4,499
OC-SORT	71.9	86.4	59.8	94.5	72.2	**3,474**
ETTrack	**74.3**	**88.8**	**62.1**	**96.8**	**74.5**	3,862
			End-to-End			
MeMOTR	70.0	83.1	59.1	91.5	71.4	-
Ours	66.5	81.5	54.4	93.7	68.9	3,526

5.2 Ablation Study

This section discusses the effects of various components in our architecture. First, different DETR bases are analyzed. Afterwards various conflict avoidance strategies are discussed, followed by a comparison of different denoising techniques. Then our proposed trajectory prediction module is analyzed. Finally, the effect of the components of our architecture on DanceTrack is discussed.

5.2.1 DETR Base

The performance of different DETR bases on the MOT17 validation split is displayed in Table 5.5. It can be seen that MQS achieves the best results, closely followed by DAB-Deformable DETR. This shows that learning 4D anchor points as positional queries and the iterative query refinement are suitable additions to the our architecture. This methods significantly increase both DetA and AssA. The Two-Stage approach performs worst, due to its poor detection performance. Deformable DETR has an acceptable DetA, but falls behind in association. This could be due to the fact, that Deformable DETR propagates 2D reference points while all other models use 4D anchor boxes. It also needs to be taken into consideration that each model is initialized with the corresponding DETR parameter from object detection on MS COCO 2017 [45]. Therefore, DAB-Deformable DETR and MQS (DINO) already have an advantage as those models have a significantly better performance in the object detection task.

Table 5.5: Comparison of different DETR bases. MQS achieves the best association performance and DAB-Deformable DETR the best DetA. Taking both aspects into consideration, MQS performs best. The tests are performed on MOT17 val. The best results are highlighted in bold.

	Metrics					
DETR	HOTA $\uparrow$	DetA $\uparrow$	AssA $\uparrow$	MOTA $\uparrow$	IDF$_1$ $\uparrow$	IDSW $\downarrow$
Deformable	51.8	54.8	49.8	61.4	59.4	903
Two-Stage	50.8	51.7	50.9	55.7	59.2	534
DAB-Deformable	56.3	**57.2**	56.3	**65.5**	64.5	534
MQS	**57.0**	56.8	**58.5**	63.5	**66.7**	**503**

5.2.2 Conflict Avoidance

This subsection compares different strategies to reduce the conflict between tracking and detection. We are analyzing the effect of GC, COLA and RFS in combination with different false negative probabilities on our model. The results are visualized in Table 5.6. It can be seen that MQS with $p_{FN} = 0.5$ performs best, while MQS with $p_{FN} = 0.1$ achieves the 2^{nd} best results. There seems to be no positive effect of either of the other methods on our architecture.

Table 5.6: Effect of different conflict avoidance strategies. Different strategies are tested in combination with different false negative probabilities p_{FN} on MOT17 val. No tested methods improves the performance over the MQS baseline. The best results are highlighted in bold.

		Metrics					
	p_{FN}	HOTA ↑	DetA ↑	AssA ↑	MOTA ↑	IDF$_1$ ↑	IDSW ↓
Baseline	0.0	55.4	52.1	**59.7**	60.2	66.6	**248**
	0.1	56.8	**57.6**	56.9	**65.2**	**66.9**	487
	0.2	55.9	56.6	56.2	63.5	65.5	523
	0.3	55.6	55.1	56.1	64.9	66.2	437
	0.5	**57.0**	56.8	58.5	63.5	66.7	467
+ GC	0.2	56.1	57.4	55.8	64.8	65.4	532
	0.5	55.1	57.0	54.2	63.7	64.1	629
+ COLA	0.2	55.4	55.6	56.5	63.5	65.2	522
	0.5	55.1	55.4	55.8	61.9	64.0	654
+ RFS	0.2	52.3	54.4	52.0	59.7	61.7	635
	0.5	50.1	52.7	49.0	55.5	57.3	885

Surprisingly, MQS with $p_{FN} = 0.0$ achieves an acceptable DetA score even though TrackFormer [12] reasons that a high p_{FN} is needed for successful joint-training. We are suggesting that this is due to the additional encoder loss which trains the extraction layers in MQS. This loss $\mathcal{L}_{enc}$ is obtained by matching the encoder proposals with the ground-truth objects independent of if they are tracked or not. Due to this the positional queries are also trained when all ground-truth objects are assigned to track queries in the decoder. Furthermore, we propose that positional queries are more important as the original DETR and DAB-DETR achieve good detection performance with only positional queries. To validate our hypothesis, we tested the effect of $p_{FN} = 0.0$ on DAB-Deformable DETR. The results are listed in Table 5.7.

Table 5.7: Effect of p_{FN} on other DETR architectures. The tests are performed on MOT17 val. MQS achieves a significantly better detection performance than DAB-Deformable DETR when no false negatives are used. This observation suggests that the MQS-specific encoder loss mitigates the necessity for high false negative probabilities.

		Metrics					
	p_{FN}	HOTA ↑	DetA ↑	AssA ↑	MOTA ↑	IDF$_1$ ↑	IDSW ↓
MQS	0.0	**55.4**	**52.1**	**59.7**	**60.2**	**66.6**	248
DAB-Def.	0.0	50.2	45.2	56.3	50.8	58.5	**212**

It can be seen that MQS performs significantly better without false positives than DAB-Deformable. That DAB-Deformable DETR still achieves a moderate DetA is due to the fact, that MOT17 features many newborn objects. Therefore, the detection task is still trained in this scenario.

Furthermore, we propose that the MOT17 set is not suited for showing advantages in the association performance as it only features simple motion patterns. Due to this we are testing MQS with $p_{FN} = 0.1$ and $p_{FN} = 0.0$ as well as GC with $p_{FN} = 0.2$ on DanceTrack validation set. The results are displayed in Table 5.8. It can be seen that a low p_{FN} leads to significant better AssA, while having nearly no impact on DetA. GC also does not have a positive influence on DanceTrack. Due to this, we are continuing with MQS with $p_{FN} = 0.1$ for the following ablation tests.

Table 5.8: Effect of different p_{FN} on DanceTrack val. A lower p_{FN} has a considerable impact on the association performance, as the model primarily matches track queries during training. In the MOT17 dataset, the effect is not discernible due to the relatively simple motion pattern.

		Metrics					
	p_{FN}	HOTA ↑	DetA ↑	AssA ↑	MOTA ↑	IDF_1 ↑	IDSW ↓
Baseline	0.1	**54.8**	72.6	**41.8**	**83.1**	**55.0**	1,962
	0.5	53.3	**72.7**	39.4	**83.1**	52.9	**1,891**
+ GC	0.2	53.8	72.3	40.4	81.8	53.9	2,171

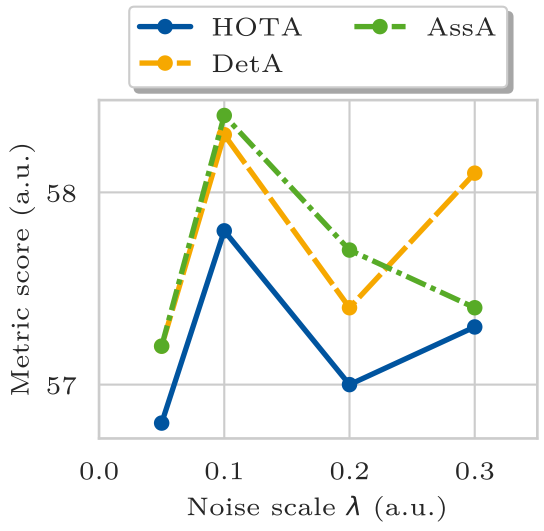
5.2.3 Denoising Strategies

In this ablation experiment, multiple denoising strategies are compared with our own strategy. The results are compared in Table 5.9. 'gt' and 'query' refer to how tracked ground-truth objects are denoised. For 'gt', the query is handled analog to untracked ground-truths, where the noise is directly applied to the ground-truth bounding box and a separate denoising embedding is used as the content query. 'query' refers to the case, where the track query is replacing the denoising query of the corresponding ground-truth object and the noise is applied to the track box. Contrastive ('contr.') refers to DINO's CDN and conditional ('cond.') to DN-MOT's conditional positive noise. It can be seen that our model performs best when using the MOTRv2 denoising strategy in combination with label noise. Other strategies, like MOTRv3's or DN-MOT's are only able to achieve marginally gains or even harm our performance. We reason that the choice of MQS and the low p_{FN} have a dramatical effect on the optimal choice of the denoising strategy. Consequently, strategies that have demonstrated considerable performance gains on alternative architectures may not necessarily be the most suitable for our model.

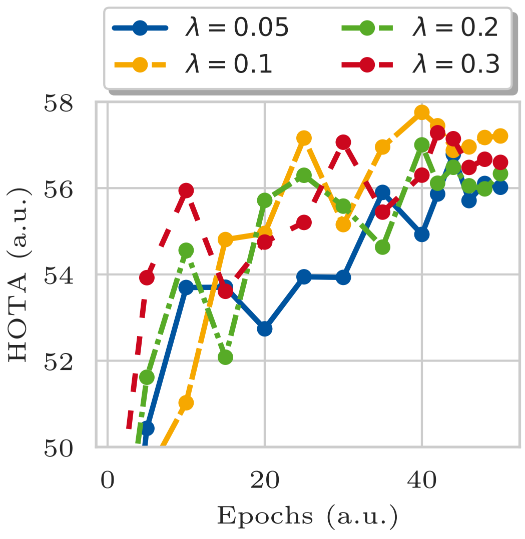
Table 5.9: Comparison of different denoising types. The best results are highlighted in bold. All tests are performed on MOT17 val. '+' denotes strategies, where noise is only applied to the width and height of the bounding box and '*' strategies, where the denoising class embeddings are randomly chosen. Dn-DETR's denoising strategy achieves the best results with our model.

	tracked					Metrics		
	gt	query	untracked	contr.	cond.	HOTA ↑	DetA ↑	AssA ↑
DN-MOT*		✓	✓	✓	✓	56.6	58.0	56.2
		✓	✓	✓		57.6	57.6	**58.5**
		✓	✓			56.6	57.7	56.5
MOTRv2	✓		✓			57.3	57.6	57.9
	✓		✓	✓		56.8	57.1	57.2
MOTRv3$^+$		✓				57.0	56.9	57.9
		✓				56.6	57.0	57.0
		✓			✓	56.8	56.2	58.1
Ours*	✓		✓			**57.8**	**58.3**	58.4

Given that previous denoising methods use very different noise scales, we test the effect of different noise scales on our method. The results are shown in Figure 5.1. It can be seen in Figure 5.1a that $\lambda = 0.1$ achieves the best results. Lower values lead to a rapid drop in detection and association performance.



(a) HOTA, DetA and Assa for different noise scales.



(b) HOTA for different noise scales over the training duration.

Figure 5.1: Effect of bounding box noise scales. The noise scale has a significant effect on the performance. Our own method achieves best results with $\lambda = 0.1$.

We reason that the performance decrease is because too little noise leads to a reconstruction task that is too simple. Consequently, the model is unable to learn any meaningful relationships. On the other hand, if the noise scale is too high, the association accuracy suffers. This may be due to the fact that too high noise is implausible in tracking scenarios. The model may erroneously learn to associate over larger distances, which is rarely advantageous in videos with moderate frame rates.

5.2.4 Trajectory Prediction

The following series of ablation experiments is designed to investigate the impact of the TPM and different configurations of it. In the initial experiment, we consider different ways of integrating the module into our model. We either refine the existing track query bounding box or generate an additional track query, called shadow query, which has the same content part but the refined bounding box from the TPM. The results are presented in Table 5.10. It can be observed that, with the current configurations, no method improves the performance of the previous baseline. However, the shadow query integration method achieves significantly better results than refining the track box directly. Consequently, the shadow query integration method is employed for the subsequent tests.

Table 5.10: Comparison of different trajectory integration methods. The tests are performed on MOT17 val. The introduction of shadow queries to integrate the TPM seems better suited than naively updating the anchor box.

	Metrics					
	HOTA ↑	DetA ↑	AssA ↑	MOTA ↑	IDF_1 ↑	IDSW ↓
Baseline	**57.8**	**58.3**	**58.4**	**67.0**	67.6	444
Refine track box	56.3	56.1	57.4	63.1	65.9	468
Shadow queries	57.7	57.9	**58.4**	66.6	**68.0**	**400**

The next experiment deals with the effect of the different losses on the TPM. The TPM is trained using a combination of two loss functions: the bounding box regression loss and the decoder losses, which are propagated back via shadow queries. To mitigate the influence of the decoder losses, the gradients of the shadow queries are cut after the TPM. The effects of the two loss components are shown in Table 5.11. It can be observed, that both losses have a positive effect on the performance of our model. Although DetA remains relatively constant, the association performance declines when one of the two losses is omitted. As a result, both losses are employed in the subsequent experiments.

Table 5.11: Comparison of trajectory prediction training methods. The tests are performed on MOT17 val. Both loss components have a positive effect on performance.

	Losses		Metrics		
	Prediction	Decoder	HOTA ↑	DetA ↑	AssA ↑
Shadow queries	✓		55.8	57.2	55.2
		✓	56.3	57.7	56.0
	✓	✓	**57.7**	**57.9**	**58.4**

One potential explanation for the absence of performance improvement of the TPM is the fact that both versions of the same track query can interact in self-attention. Since self-attention is used to suppress duplicate predictions, it is possible that a conflict occurs within the module due to the forced two versions. This is investigated in Table 5.12. The results demonstrate that the interaction between the track queries and the shadow queries has a positive effect on the tracking performance.

Table 5.12: Effect of masking shadow queries in the trajectory prediction module. The configurations are tested on MOT17 val. Communication between shadow and track queries has a positive impact on performance.

	Metrics					
	HOTA ↑	DetA ↑	AssA ↑	MOTA ↑	IDF_1 ↑	IDSW ↓
No masking	**57.7**	**57.9**	**58.4**	**66.6**	**68.0**	**400**
Masking	57.6	**57.9**	58.1	66.5	67.7	431

During inference, the variant with the higher prediction score is selected from a given track-shadow-query pair. Figure 5.2 demonstrates that both types of queries are selected.

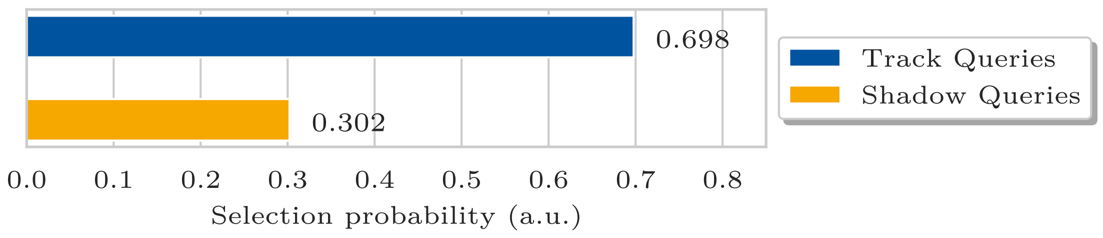


Figure 5.2: Selection probability for each query type. During inference the highest scoring version of each track-shadow query pair is picked. Track queries tend to achieve higher scores than shadow queries.

However, the selection based on the prediction score is dependent on the assumption that predictions with a higher score also have a higher gIoU. This assumption is not necessarily justified by the use of the focal loss, given that the ground-truth labels are binary. To make quantitative statements regarding the box quality based on the score, we investigate the effect of quality focal loss [67] instead of focal loss on our model. The results are presented in Table 5.13. It can be seen that quality focal loss has no significant effect on the performance, therefore we are staying with the normal focal loss for the following experiments.

Table 5.13: Effect of quality focal loss (QFL). The methods are tested on MOT17 val. Quality focal loss does not result in any discernible improvement over the conventional focal loss.

	Metrics					
	HOTA ↑	DetA ↑	AssA ↑	MOTA ↑	IDF_1 ↑	IDSW ↓
Shadow queries	**57.7**	57.9	**58.4**	66.6	68.0	**400**
+ QFL	57.6	**58.0**	58.3	**67.0**	**68.1**	447

Another influencing factor could be the generation of noise. We compare our modeled noise with Gaussian noise with mean 1 and different standard deviations. It can be seen in Table 5.14 that the modeled noise performs better and does not require dataset/model-dependent hyper-parameters as it is fit to the model predictions directly.

Table 5.14: Effect of different noise models. The methods are tested on MOT17 val. Modeled noise achieves the best results without using additional hyperparameters. We reason that a fixed noise distribution is not suited for the task, as the real prediction distribution changes during training.

		Metrics					
	Std	HOTA ↑	DetA ↑	AssA ↑	MOTA ↑	IDF_1 ↑	IDSW ↓
Modeled noise		**57.7**	**57.9**	**58.4**	**66.6**	**68.0**	**400**
Gaussian	0.01	56.6	56.8	57.3	63.6	66.7	471
	0.03	57.3	57.6	57.7	65.1	67.2	502

To verify the basic concept of our method, we test the effect of an already established motion model, the Kalman filter. To do this, we replace the TPM with a Kalman filter during inference. The results are presented in Table 5.15.

Table 5.15: Effect of different motion models. The Kalman filter demonstrates a good performance on MOT17, as its linear velocity assumption aligns with the the simple motion pattern. Conversely, on Dancetrack, it results in a decline in performance, whereas our proposed TPM exhibits a notable enhancement in AssA.

	Datasets		Metrics		
	MOT17	DanceTrack	HOTA ↑	DetA ↑	AssA
Baseline	✓		57.8	**58.3**	58.4
TPM + Shadow queries	✓		57.7	57.9	58.4
Kalman + Shadow queries	✓		57.5	57.8	58.2
Kalman + Box refine	✓		**58.5**	57.7	**60.3**
Baseline		✓	54.7	**73.6**	40.9
TPM + Shadow queries		✓	**54.8**	72.3	**42.0**
Kalman + Shadow queries		✓	54.3	72.2	41.2
Kalman + Box refine		✓	52.8	72.0	39.2

The results allow us to infer a number of conclusions. First and foremost, a well-suited motion model can be utilized in conjunction with E2E models to enhance their performance. This can be seen as the Kalman filter, which is due to its linearity assumption a good motion estimator for MOT17, increases the performance significantly. Additionally, it is evident that the prediction score is not an effective surrogate for anchor box quality, as the Kalman filter does not yield any discernible enhancement when employed in conjunction with shadow queries. Therefore, the model is unable to reliably select the optimal query. On DanceTrack, the Kalman filter leads to a decline in performance, which is anticipated given that the Kalman filter is not applicable to the complex and nonlinear motions that are characteristic of DanceTrack. Our TPM is currently not an effective motion model for MOT17, but it does yield modest performance gains on DanceTrack.

5.2.5 Image Augmentations

The primary distinction between MOTR and TrackFormer lies in the utilization of HSV color space augmentations and the dimensions of the random crops employed in the image augmentations. The outcomes achieved through the deployment of MOTR and TrackFormer image augmentation are illustrated in Table 5.16. It is evident that the use of MOTR augmentations results in superior outcomes in both the MOT17 and Dancetrack datasets. The lower increase on MOT17 may be attributed to the fact that MOT17 is a very small dataset, and the use of larger crops leads to a reduction in variance during training.

Table 5.16: Effect of different image augmentations. The image augmentations employed by MOTR result in notable enhancements in performance on both the MOT17 and Dancetrack datasets.

	Metrics					
Augmentations	HOTA ↑	DetA ↑	AssA ↑	MOTA ↑	IDF_1 ↑	IDSW ↓
			MOT17 val			
TrackFormer	57.7	**57.9**	58.4	**66.6**	68.0	400
MOTR	**58.0**	57.5	**59.3**	65.5	**68.6**	**358**
			DanceTrack val			
TrackFormer	54.8	72.3	42.0	82.0	**55.3**	1,890
MOTR	**55.2**	**72.9**	**42.2**	**82.7**	55.2	**1,795**

5.2.6 Hyperparameter Optimization

The hyperparameters determined by Optuna [61] are shown in Table 5.17. It is noticeable that in MOT17 both τ_{det} and T_{reid} are chosen significantly higher, whereas τ_{track} and τ_{reid} remain unchanged. For DanceTrack, all hyperparameters except for τ_{track} are increased.

Table 5.17: Results from hyperparameter optimization. We conducted 50 and 20 optimization iterations with Optuna on MOT17 and DanceTrack, respectively. Of particular note are the increases to τ_{det} and T_{reid}. This suggests that more confident detections result in fewer, but better track initializations. Due to the TPM, reidentifications are still possible after longer time frames.

			Hyperparameters			
Hyperparameter	MOT17	DanceTrack	τ_{det}	τ_{track}	τ_{reid}	T_{reid}
Standard	✓		0.4	0.4	0.4	5
Optimized	✓		0.65	0.4	0.4	20
Standard		✓	0.4	0.4	0.4	5
Optimized		✓	0.85	0.4	0.6	17

The results in Table 5.18 show that the association performance in particular benefits significantly from optimized hyperparameters. Since the TPM also updates the anchor box of inactive tracks, reidentifications are still possible after a longer period of time. The fact that T_{reid} was set to 20 on MOT17 indicates that this is particularly holds for linear, uniform movements. As the detection task in DanceTrack is comparatively simple, higher thresholds perform better. The fact that τ_{det} is higher than τ_{track} in both datasets ensures that only tracks are initialized if the model is certain, whereas the threshold for continuing an existing track is significantly lower. This results in fewer competing tracks, reducing the risk of IDSW.

Table 5.18: Performance with optimized hyperparameters. The dets are conducted on MOT17 and DanceTrack val. Our model demonstrates a notable improvement in performance when optimized hyperparameters are employed. As a consequence of the architectural modifications, the hyperparameters of TrackFormer are no longer optimal for our model.

	Metrics					
Hyperparameter	HOTA ↑	DetA ↑	AssA ↑	MOTA ↑	IDF$_1$ ↑	IDSW ↓
			MOT17 val			
Standard	58.0	**57.5**	59.3	65.5	68.6	358
Optimized	**59.3**	57.2	**62.4**	**66.8**	**71.6**	**223**
			DanceTrack val			
Standard	55.2	**72.9**	42.2	**82.7**	55.2	1,795
Optimized	**57.0**	72.6	**45.2**	82.4	**58.7**	**1,152**

5.2.7 Different components on DanceTrack

As the majority of previous ablation experiments were conducted using MOT17, the individual components will be evaluated once more using on the DanceTrack validation set. Table 5.19 illustrates the performance of each of the components we introduced.

Table 5.19: Effect of the different components on DanceTrack val. The most significant factors influencing performance are MQS, the lower p_{FN}, and the use of MOTR's image augmentation. The denoising approach evaluated on MOT17 is not effective on Dancetrack.

		Metrics					
	Component	HOTA ↑	DetA ↑	AssA ↑	MOTA ↑	IDF$_1$ ↑	IDSW ↓
(1)	Def. DETR	46.1	68.7	31.3	79.1	44.9	2,150
(2)	MQS	53.3	72.7	39.4	83.1	52.9	1,891
(3)	(2) + p_{FN}=0.1	54.8	72.6	41.8	83.1	55.0	1,962
(4)	(3) + DN	54.7	**73.6**	40.9	**84.1**	54.1	1,780
(5)	(4) + TPM	54.8	72.3	42.0	82.0	55.3	1,890
(6)	(5) + Augm.	55.2	72.9	42.2	82.7	55.2	1,795
(7)	(6) + Hyperp.	**57.0**	72.6	**45.2**	82.4	**58.7**	**1,152**

It can be observed that a reduction in p_{FN} has a considerable impact on the performance of the system, However, the denoising strategy employed in this study does not appear to be effective for DanceTrack, despite achieving the best results on MOT17. This suggests that the performance of the model on MOT17 may not be a reliable indicator of its performance on DanceTrack. The integration of the TPM has led to a notable enhancement in the models association performance, albeit at the cost of detection performance. It is hypothesized that the pronounced impact of the optimized hyperparameters is, in part, attributable to the TPM, as this facilitates reidentification after considerably longer intervals. In the absence of the TPM, the spatial location of inactive track queries exhibits minimal change. In contrast, the TPM enables the anchor box to be updated in accordance with the preceding motion.

5.3 Training Time and Memory Requirement

Since our model is trained exclusively on two frames and does not use separate object detectors or any other frame features, it can be trained quickly. The high resolution training for 50 epochs on MOT17 took a total of 36h on a single NVIDIA L40s. The training for 20 epochs on DanceTrack and 28 on SportsMOT took 110h and 75h respectively. The computing time of the different components per training iteration is shown in Table 5.20. It is hypothesized that the temporal component of the criterion is disproportionately represented in the tests, as the loss calculation is not significantly enhanced by a more rapid GPU, whereas the transformer structure is optimized for GPU training. On slower GPUs, the percentage shifts more towards the encoder/decoder.

Table 5.20: Time spend per component for a single iteration. The times were measured over one epoch of MOT17 train using a NVIDIA L40s. '(×2)' denotes, that the component is called two times during a single training iteration. The criterion contributes a significant portion of the total computation time, since the loss calculation does not notably benefit from a faster GPU. The use of a slower GPU results in a comparatively lower relative contribution of the criterion.

	Training		Inference	
Component	Time (ms)	Time (%)	Time (ms)	Time (%)
Backbone	9.4 (×2)	1.9 (×2)	9.8	10.7
Encoder	23.6 (×2)	4.9 (×2)	24.1	26.3
MQS	3.0 (×2)	0.6 (×2)	2.0	2.2
Decoder	29.8 (×2)	6.2 (×2)	17.2	18.7
TPM	28.9	6.0	28.6	31.2
Criterion	81.6	16.9	-	-
Backpropagation	169.5	35.1	-	-
Other	71.8	28.5	10.1	11.0
Total	483.4	100.0	91.8	100.0

In total, the model requires approximately 0.5 seconds per training iteration and achieves up to 11 Hz during inference. The impact of MQS and denoising (29.3 vs 29.8 ms in the decoder) on the forward pass of the model is negligible; however, the introduction of additional parameters and losses results in an increase in time for both the criterion and backpropagation. Figure 5.3 presents a comparison of the time per training iteration of our model with MOTR and TrackFormer. The runtime of the two architectures also serves as a lower bound for all models based on them. It can be seen that our model has a significantly lower computation time. The advantage over TrackFormer lies in the reduced feature dimension and the fact that we do not use previous frame features. The prolonged processing time of MOTR can be attributed to its approach of processing five frames in sequence.

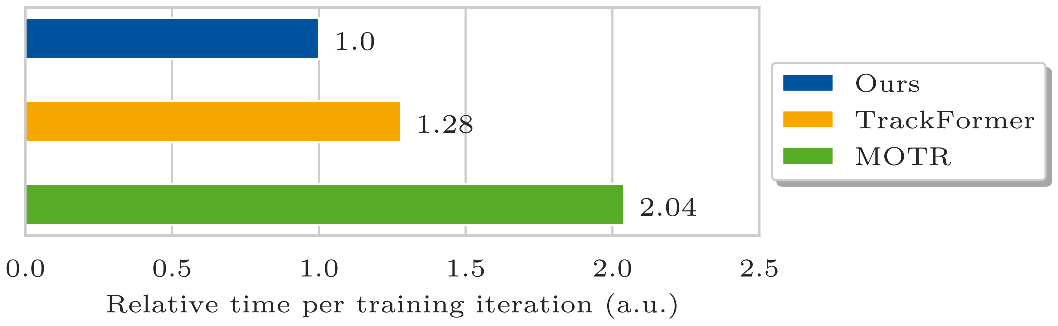


Figure 5.3: Relative time for a single iteration during training for different models. Our architecture only trains on to adjacent frames and does not employ additional features from previous frames. This leads to a significant speedup compared to TrackFormer and MOTR.

In addition to runtime, the VRAM requirement is a significant consideration. A reduced VRAM requirement permits the model to be trained on less costly or less recent GPUs. Figure 5.4 provides a summary of the VRAM requirements for each of the modifications that have been introduced. It can be observed, that the largest increase in VRAM is due to MQS. This is due to the fact, that MQS introduces 7 million additional parameters compared to Deformable DETR, resulting in a total of 47 million parameters. This proportionally increases the required VRAM by 20%. In contrast, denoising has nearly no effect on the allocated VRAM, and the TPM does also only leads to an increase of approximately 5%.

Figure 5.5 presents a comparative analysis of the VRAM requirements of various models. It is evident that our model requires a considerably lesser amount of VRAM in comparison to other E2E MOT models. As a consequence of the absence of any previous frame features and the smaller feature dimension employed in comparison to TrackFormer, our model requires approximately 40% less VRAM than TrackFormer. The advantage of our approach over that of MOTR is that we calculate the loss on a single frame, whereas MOTR calculates the loss on all five frames. This results in a reduction of the allocated VRAM by approximately 70% in comparison to MOTR.

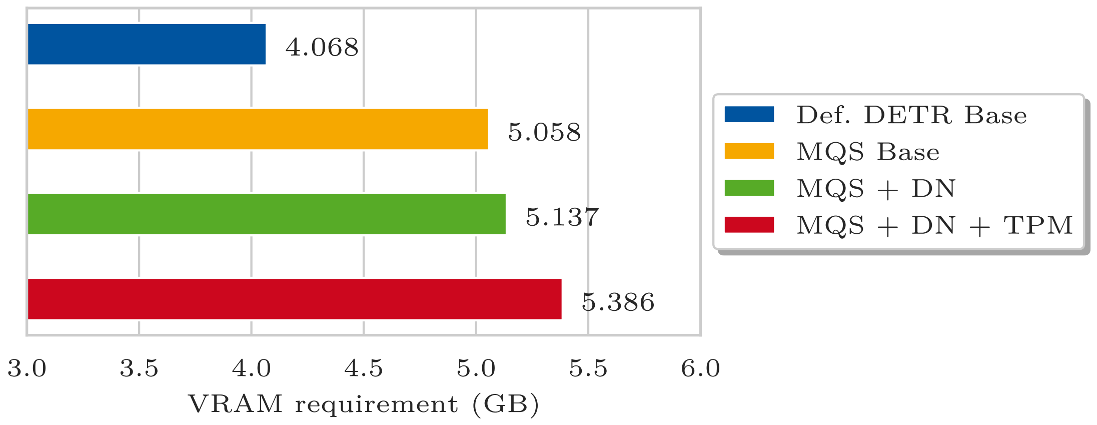


Figure 5.4: VRAM requirement during training. The maximum allocated VRAM was measured during one epoch on MOT17 train with a maximum image size of 1536px. MQS introduces 7 million additional parameters over the Deformable DETR base resulting in 47 million parameters in total. This leads to an approximate 20% increase in the VRAM requirement. Both denoising and TPM have a minimal impact on the requirements.

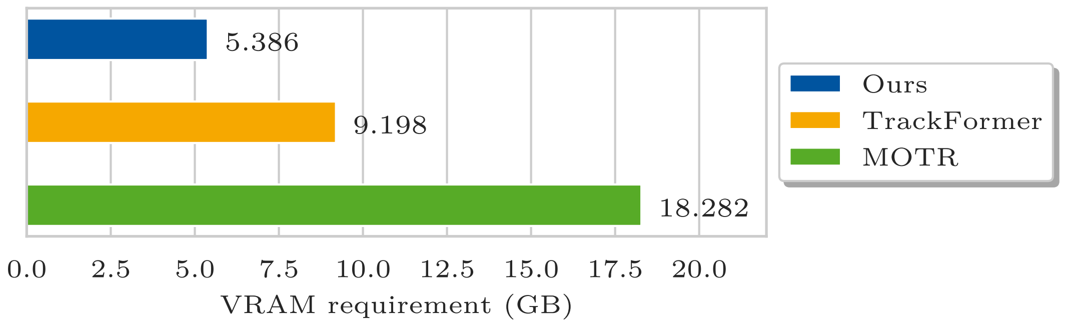


Figure 5.5: VRAM requirement of different models during training. Our architectural approach does not incorporate any additional features from previous frames and solely calculates the loss on a single frame. Consequently, the number of requisite gradients during training is substantially reduced. This results in a markedly lower VRAM requirement in comparison to TrackFormer and MOTR.

6 Discussion

We have proposed a lightweight End-to-End Multiple Object Tracking model that employs the use of historical trajectories. By limiting the training to two adjacent frames and calculating the loss for a single frame, our model requires less than 6 GB of VRAM and exhibits minimum 20% acceleration compared to TrackFormer-based and 50% to MOTR-based models. Our model employs MQS to enhance its performance and mitigate the conflict between detection and association. A specific denoising strategy is employed to further improve the detection performance. By leveraging the Trajectory Prediction Module, our model is capable of enhancing the spatial information associated with track queries, based on the historical trajectories of the tracked objects. Moreover, we optimized both the augmentation strategy and the track hyperparameters for our model, thereby boosting its overall performance. Our results demonstrate that lightweight End-to-End approaches are capable of competing with more heavy-weight approaches, such as MOTR. Additionally, we were able to demonstrate that the incorporation of additional motion predictors can significantly improve the performance of End-to-End Multiple Object Tracking models.

While our model demonstrates notable advancements over existing TrackFormer-based models in scenarios with complex motion patterns, it still exhibits inferior performance compared to more recent MOTR-based models. Furthermore, our experiments revealed that the most effective methods on MOT17 are not always the most effective on DanceTrack, and vice versa. This resulted in the deployment of a suboptimal denoising strategy. Additionally, our current motion prediction model is not fully leveraging the potential of enhancing track queries' spatial information. Despite reducing the conflict between detection and association, our model's detection performance still lags behind two-stage approaches. We believe that resolving this conflict is a pivotal challenge for the Tracking-by-Query paradigm.

This thesis suggests two primary directions for future research. The first focuses on enhancing track bounding box prediction using historical trajectories, while the second aims at developing a more efficient, lightweight model. For trajectory prediction, selecting an appropriate motion model is essential. It would be beneficial to draw further inspiration from existing Tracking-by-Detection and trajectory prediction models. To improve efficiency, exploring more advanced and efficient DETR models, such as RT-DETR [68] or D-Fine [69], would be a promising approach.

Bibliography

[1] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler, "MOT16: A Benchmark for Multi-Object Tracking," *arXiv:1603.00831*, Mar. 2016.

[2] P. Dendorfer, H. Rezatofighi, A. Milan, J. Shi, D. Cremers, I. Reid, S. Roth, K. Schindler, and L. Leal-Taixé, "MOT20: A benchmark for multi object tracking in crowded scenes," Mar. 2020. arXiv:2003.09003.

[3] P. Sun, J. Cao, Y. Jiang, Z. Yuan, S. Bai, K. Kitani, and P. Luo, "Dancetrack: Multi-object tracking in uniform appearance and diverse motion," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 20993–21002, 2022.

[4] Y. Cui, C. Zeng, X. Zhao, Y. Yang, G. Wu, and L. Wang, "Sportsmot: A large multi-object tracking dataset in multiple sports scenes," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9921–9931, 2023.

[5] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell, "Bdd100k: A diverse driving dataset for heterogeneous multitask learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2636–2645, 2020.

[6] L. Zhang, J. Gao, Z. Xiao, and H. Fan, "Animaltrack: A benchmark for multi-animal tracking in the wild," *International Journal of Computer Vision*, vol. 131, no. 2, pp. 496–513, 2023.

[7] R. Girdhar, G. Gkioxari, L. Torresani, M. Paluri, and D. Tran, "Detect-and-Track: Efficient Pose Estimation in Videos," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, (Salt Lake City, UT), pp. 350–359, IEEE, June 2018.

[8] P. Pareek and A. Thakkar, "A survey on video-based Human Action Recognition: recent updates, datasets, challenges, and applications," *Artificial Intelligence Review*, vol. 54, pp. 2259–2322, Mar. 2021.

[9] Y. Liu, B. Li, X. Zhou, D. Li, and Q. Duan, "FishTrack: Multi-object tracking method for fish using spatiotemporal information fusion," *Expert Systems with Applications*, vol. 238, p. 122194, Mar. 2024.

[10] Z. Sun, J. Chen, L. Chao, W. Ruan, and M. Mukherjee, "A Survey of Multiple Pedestrian Tracking Based on Tracking-by-Detection Framework," *IEEE*

Transactions on Circuits and Systems for Video Technology, vol. 31, pp. 1819–1833, May 2021.

[11] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-End Object Detection with Transformers," in *European Conference on Computer Vision (ECCV)* (A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, eds.), vol. 12346, pp. 213–229, Cham: Springer International Publishing, 2020.

[12] T. Meinhardt, A. Kirillov, L. Leal-Taixe, and C. Feichtenhofer, "TrackFormer: Multi-Object Tracking with Transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, (New Orleans, LA, USA), pp. 8834–8844, IEEE, June 2022.

[13] F. Zeng, B. Dong, Y. Zhang, T. Wang, X. Zhang, and Y. Wei, "Motr: End-to-end multiple-object tracking with transformer," in *European Conference on Computer Vision (ECCV)*, pp. 659–675, Springer, 2022.

[14] T. Fu, X. Wang, H. Yu, K. Niu, B. Li, and X. Xue, "DeNoising-MOT: Towards Multiple Object Tracking with Severe Occlusions," in *Proceedings of the 31st ACM International Conference on Multimedia*, MM '23, (New York, NY, USA), pp. 2734–2743, Association for Computing Machinery, 2023. event-place: Ottawa ON, Canada.

[15] R. Gao and L. Wang, "MeMOTR: Long-term memory-augmented transformer for multi-object tracking," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9901–9910, 2023.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, (Las Vegas, NV, USA), pp. 770–778, IEEE, 2016.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is All you Need," in *Advances in Neural Information Processing Systems (NIPS)* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[18] G. Ciaparrone, F. Luque Sánchez, S. Tabik, L. Troiano, R. Tagliaferri, and F. Herrera, "Deep learning in video multi-object tracking: A survey," *Neurocomputing*, vol. 381, pp. 61–88, Mar. 2020.

[19] M. Bashar, S. Islam, K. K. Hussain, M. B. Hasan, A. B. M. A. Rahman, and M. H. Kabir, "Multiple Object Tracking in Recent Times: A Literature Review," 2022.

[20] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *IEEE International Conference on Image Processing (ICIP)*, (Phoenix, AZ, USA), pp. 3464–3468, IEEE, Sept. 2016.

[21] P. Bergmann, T. Meinhardt, and L. Leal-Taixe, “Tracking Without Bells and Whistles,” in *The IEEE/CVF International Conference on Computer Vision (ICCV)*, (Seoul, Korea (South)), pp. 941–951, IEEE, Oct. 2019.

[22] S. Han, P. Huang, H. Wang, E. Yu, D. Liu, and X. Pan, “MAT: Motion-aware multi-object tracking,” *Neurocomputing*, vol. 476, pp. 75–86, Mar. 2022.

[23] F. Yu, W. Li, Q. Li, Y. Liu, X. Shi, and J. Yan, “POI: Multiple Object Tracking with High Performance Detection and Appearance Feature,” in *Computer Vision – ECCV 2016 Workshops* (G. Hua and H. Jégou, eds.), vol. 9914, pp. 36–42, Cham: Springer International Publishing, 2016.

[24] Y. Zhang, C. Wang, X. Wang, W. Zeng, and W. Liu, “FairMOT: On the Fairness of Detection and Re-identification in Multiple Object Tracking,” *International Journal of Computer Vision*, vol. 129, pp. 3069–3087, Nov. 2021.

[25] R. Gao, Y. Zhang, and L. Wang, “Multiple Object Tracking as ID Prediction,” Mar. 2024. arXiv:2403.16848 [cs].

[26] R. Nevatia, “Tracking of Multiple, Partially Occluded Humans based on Static Body Part Detection,” in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, (New York, NY, USA), pp. 951–958, IEEE, 2006.

[27] K. Bernardin and R. Stiefelhagen, “Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics,” *EURASIP Journal on Image and Video Processing*, vol. 2008, pp. 1–10, 2008.

[28] E. Ristani, F. Solera, R. Zou, R. Cucchiara, and C. Tomasi, “Performance Measures and a Data Set for Multi-target, Multi-camera Tracking,” in *Computer Vision – ECCV 2016 Workshops* (G. Hua and H. Jégou, eds.), vol. 9914, pp. 17–35, Cham: Springer International Publishing, 2016.

[29] J. Luiten, A. Osep, P. Dendorfer, P. Torr, A. Geiger, L. Leal-Taixé, and B. Leibe, “HOTA: A Higher Order Metric for Evaluating Multi-Object Tracking,” *International Journal of Computer Vision*, pp. 1–31, 2020.

[30] H. W. Kuhn, “The Hungarian method for the assignment problem,” *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, Mar. 1955.

[31] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation Applied to Handwritten Zip Code Recognition,” *Neural Computation*, vol. 1, pp. 541–551, Dec. 1989.

[32] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems (NIPS)* (F. Pereira, C. J. Burges, L. Bottou, and K. Q. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[33] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[34] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, pp. 6999–7019, Dec. 2022.

[35] A. Mahmood, A. G. Ospina, M. Bennamoun, S. An, F. Sohel, F. Boussaid, R. Hovey, R. B. Fisher, and G. A. Kendrick, "Automatic Hierarchical Classification of Kelps Using Deep Residual Features," *Sensors*, vol. 20, p. 447, Jan. 2020.

[36] S. Hochreiter, "Long Short-term Memory," *Neural Computation*, vol. 9, p. 1735–1780, Nov. 1997.

[37] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," in *NIPS 2014 Deep Learning and Representation Learning Workshop*, 2014.

[38] Z. Niu, G. Zhong, and H. Yu, "A review on the attention mechanism of deep learning," *Neurocomputing*, vol. 452, pp. 48–62, 2021.

[39] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, "Deformable {DETR}: Deformable Transformers for End-to-End Object Detection," in *International Conference on Learning Representations (ICLR)*, 2021.

[40] S. Liu, F. Li, H. Zhang, X. Yang, X. Qi, H. Su, J. Zhu, and L. Zhang, "DAB-DETR: Dynamic Anchor Boxes are Better Queries for DETR," in *International Conference on Learning Representations (ICLR)*, 2022.

[41] F. Li, H. Zhang, S. Liu, J. Guo, L. M. Ni, and L. Zhang, "Dn-detr: Accelerate detr training by introducing query denoising," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13619–13627, 2022.

[42] H. Zhang, F. Li, S. Liu, L. Zhang, H. Su, J. Zhu, L. Ni, and H.-Y. Shum, "DINO: DETR with Improved DeNoising Anchor Boxes for End-to-End Object Detection," in *International Conference on Learning Representations (ICLR)*, 2023.

[43] N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, and D. Tran, "Image Transformer," in *Proceedings of the 35th International Conference on Machine Learning (ICML)* (J. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, pp. 4055–4064, PMLR, July 2018.

[44] T.-Y. Ross and G. Dollár, “Focal loss for dense object detection,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2980–2988, 2017.

[45] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft coco: Common objects in context,” in *Computer Vision – ECCV 2014*, pp. 740–755, Springer, 2014.

[46] D. Zheng, W. Dong, H. Hu, X. Chen, and Y. Wang, “Less is more: Focus attention for efficient detr,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6674–6683, 2023.

[47] E. Yu, T. Wang, Z. Li, Y. Zhang, X. Zhang, and W. Tao, “Motrv3: Release-fetch supervision for end-to-end multi-object tracking,” *arXiv preprint arXiv:2305.14298*, 2023.

[48] Y. Wang, Z. Xu, X. Wang, C. Shen, B. Cheng, H. Shen, and H. Xia, “End-to-end video instance segmentation with transformers,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8741–8750, 2021.

[49] Z. Ge, “Yolox: Exceeding yolo series in 2021,” *arXiv preprint arXiv:2107.08430*, 2021.

[50] Q. Chen, X. Chen, J. Wang, S. Zhang, K. Yao, H. Feng, J. Han, E. Ding, G. Zeng, and J. Wang, “Group detr: Fast detr training with group-wise one-to-many assignment,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6633–6642, 2023.

[51] F. Yan, W. Luo, Y. Zhong, Y. Gan, and L. Ma, “Bridging the gap between end-to-end and non-end-to-end multi-object tracking,” *arXiv preprint arXiv:2305.12724*, 2023.

[52] Y. Zhang, T. Wang, and X. Zhang, “Motrv2: Bootstrapping end-to-end multi-object tracking by pretrained object detectors,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22056–22065, 2023.

[53] C. Xiao, Q. Cao, Y. Zhong, L. Lan, X. Zhang, Z. Luo, and D. Tao, “Motiontrack: Learning motion predictor for multiple object tracking,” *Neural Networks*, vol. 179, p. 106539, 2024.

[54] R. E. Kalman, “A New Approach to Linear Filtering and Prediction Problems,” *Journal of Basic Engineering*, vol. 82, pp. 35–45, Mar. 1960.

[55] “Implementation of dn-detr.” https://github.com/IDEA-Research/DN-DETR, Nov. 2024. Accessed: 2024-11-23.

[56] T. Meinhardt, “Implementation of "TrackFormer: Multi-Object Tracking with Transformers”.” https://github.com/timmeinhardt/trackformer, Nov. 2024. Accessed: 2024-11-23.

[57] M. Contributors, “MMCV: OpenMMLab Computer Vision Foundation.” https://github.com/open-mmlab/mmcv, 2018. Accessed: 2024-11-23.

[58] J. Luiten and A. Hoffhues, “TrackEval.” https://github.com/JonathonLuiten/TrackEval, 2020. Accessed: 2024-11-23.

[59] I. Loshchilov, “Decoupled weight decay regularization,” *arXiv preprint arXiv:1711.05101*, 2017.

[60] Q. Wang, J. He, Y. Chen, and Z. Zhang, “OneTrack: Demystifying the Conflict Between Detection and Tracking in End-to-End 3D Trackers,” in *Computer Vision – ECCV 2024* (A. Leonardis, E. Ricci, S. Roth, O. Russakovsky, T. Sattler, and G. Varol, eds.), (Cham), pp. 387–404, Springer Nature Switzerland, 2025.

[61] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A Next-Generation Hyperparameter Optimization Framework,” in *The 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2623–2631, 2019.

[62] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, “A ConvNet for the 2020s,” *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[63] S. Shao, Z. Zhao, B. Li, T. Xiao, G. Yu, X. Zhang, and J. Sun, “CrowdHuman: A Benchmark for Detecting Human in a Crowd,” *arXiv preprint arXiv:1805.00123*, 2018.

[64] Y. Zhang, P. Sun, Y. Jiang, D. Yu, F. Weng, Z. Yuan, P. Luo, W. Liu, and X. Wang, “ByteTrack: Multi-Object Tracking by Associating Every Detection Box,” *Proceedings of the European Conference on Computer Vision (ECCV)*, 2022.

[65] J. Cao, J. Pang, X. Weng, R. Khirodkar, and K. Kitani, “Observation-centric sort: Rethinking sort for robust multi-object tracking,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9686–9696, 2023.

[66] X. Han, N. Oishi, Y. Tian, E. Ucurum, R. Young, C. Chatwin, and P. Birch, “ETTrack: Enhanced Temporal Motion Predictor for Multi-Object Tracking,” *arXiv preprint arXiv:2405.15755*, 2024.

[67] X. Li, W. Wang, L. Wu, S. Chen, X. Hu, J. Li, J. Tang, and J. Yang, “Generalized focal loss: Learning qualified and distributed bounding boxes for dense

object detection," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21002–21012, 2020.

[68] Y. Zhao, W. Lv, S. Xu, J. Wei, G. Wang, Q. Dang, Y. Liu, and J. Chen, "DETRs Beat YOLOs on Real-time Object Detection," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, (Seattle, WA, USA), pp. 16965–16974, IEEE, June 2024.

[69] Y. Peng, H. Li, P. Wu, Y. Zhang, X. Sun, and F. Wu, "D-FINE: Redefine Regression Task in DETRs as Fine-grained Distribution Refinement," Oct. 2024. arXiv:2410.13842.

[70] "Reproducibility — PyTorch 2.5 documentation." https://pytorch.org/docs/stable/notes/randomness.html, Dec. 2024. Accessed: 2024-12-11.

[71] "API Reference guide for cuBLAS." https://docs.nvidia.com/cuda/cublas/index.html, Dec. 2024. Accessed: 2024-12-11.

A Reproducibility of Results

In order to ensure the reproducibility of the results, the code illustrated in Listing A.1 is employed before the training starts.

```python
# Reproducibility
# fix seeds
seed = args.seed
os.environ['PYTHONHASHSEED'] = str(seed)
np.random.seed(seed)
random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)

# set environment variable for CuBLAS
os.environ["CUBLAS_WORKSPACE_CONFIG"] = ":4096:8"

# use only deterministic algorithms or print warning of not possible
torch.backends.cudnn.deterministic = True
torch.use_deterministic_algorithms(True, warn_only=True)

# forces cuDNN to deterministically select an algorithm
torch.backends.cudnn.benchmark = False

# preserve reproducibility for multi-process data loading
def seed_worker(worker_id):
    worker_seed = torch.initial_seed() % 2**32
    np.random.seed(worker_seed)
    random.seed(worker_seed)

g = torch.Generator()
g.manual_seed(seed)
```

Listing A.1: Necessary settings for reproducibility.

The initial block serves to fix the seeds of the various packages. Given that the weights are initialized randomly and the augmentations are selected randomly, it is necessary to utilize a fixed seed [70]. Subsequently, the CUDA-specific settings are configured. It is necessary to set a CuBLAS environment variable, as the multi-stream execution introduces non-deterministic behavior [71]. In instances where PyTorch and CUDA have accelerated, non-deterministic implementations of certain functions, it is essential to ensure that only deterministic algorithms are employed. Additionally, CUDA selects optimal algorithms based on the hardware, which can result in non-deterministic behavior [70].

B Hyperparameters

Table B.1: Overview of hyperparameters used for each dataset.

	Datasets		
Parameter	MOT17		SportsMOT
Learning rate	$5 \cdot 10^{-5}$	$5 \cdot 10^{-5}$	$5 \cdot 10^{-5}$
Learning rate (backbone)	$5 \cdot 10^{-6}$	$5 \cdot 10^{-6}$	$5 \cdot 10^{-6}$
Batch size	1	1	1
Accumulation steps	8	8	8
Epochs	50	20	28
Learning rate drop epoch	40	10	18
Learning rate drop decay	0.1	0.1	0.1
FFN dimension	2048	2048	2048
Feature dimension	256	256	256
Dropout	0	0	0
Number of object queries	300	300	300
λ_{class}	2	2	2
λ_{box}	5	5	5
λ_{giou}	2	2	2
λ_{pred}	50	50	50
Denoising groups	5	5	5
Bounding box noise scale	0.1	0.1	0.1
Label noise scale	1	1	1
Category noise range	20	20	20
Noise queue length	1000	1000	1000
Trajectory false negative probability	0.1	0.1	0.1
Memory length	10	10	10
Previous frame range	[-5, 5]	[-5, 5]	[-5, 5]
Track false positive probability	0.1	0.1	0.1
Track false negative probability	0.1	0.1	0.1
τ_{det}	0.65	0.85	0.65
τ_{track}	0.4	0.4	0.4
τ_{reid}	0.4	0.6	0.4
T_{reid}	20	17	20
$\tau_{NMS,det}$	0.9	0.9	0.9

C Extended Results

Table C.1: Results per sequence on MOT17 test.

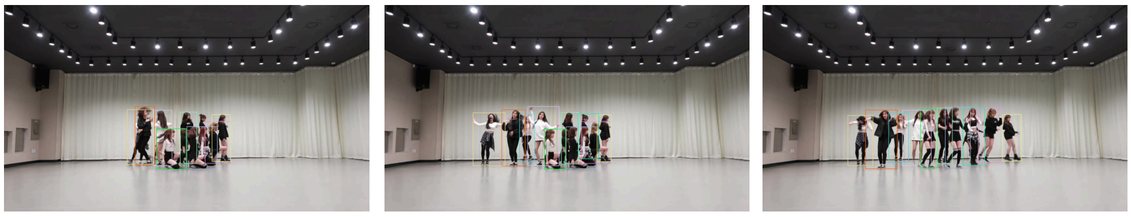
	Metrics																	
Sequence	HOTA	DetA	DetRe	DetPr	AssA	AssRe	AssPr	LocA	MOTA	MOTP	IDF1	IDP	IDR	TP	FP	FN	Rcll	Prcn	IDSW
MOT17-01	43.5	43.6	54.8	59.7	44	48.5	72.1	79.5	47.3	76.1	51.9	54.2	49.8	4511	1416	1939	69.9	76.1	46
MOT17-03	65.4	70.3	75.8	78.9	61.2	66.3	78.2	82.5	88.4	80	79.7	81.4	78.1	96575	3935	8100	92.3	96.1	114
MOT17-06	50.4	52.9	60	73.1	48.3	60.6	65.5	83.2	60.9	80.8	61.7	68.5	56.2	8502	1172	3282	72.2	87.9	159
MOT17-07	43.4	50.7	56.4	72.3	37.8	41.1	71.3	81	60.2	78.3	51	58.2	45.4	11742	1428	5151	69.5	89.2	148
MOT17-08	39.4	39.6	42.1	77.8	39.9	43.9	73.9	83.1	45.6	80.8	44.4	63.3	34.2	10617	811	10507	50.3	92.9	183
MOT17-12	53.6	49.7	55.9	73.9	58.1	62.4	81.3	83.9	57	81.6	65.2	75.7	57.2	5765	788	2902	66.5	88	39
MOT17-14	37.2	38.1	43.6	65.6	36.8	41.4	67	80	41.9	77	48.1	60.2	40.1	10118	2180	8365	54.7	82.3	199
Total	56.5	58.7	64.4	76	55	60.2	76.1	82.3	71.9	79.7	68.4	74.5	63.2	443490	35190	120738	78.6	92.7	2664

Table C.2: Results per sequence on DanceTrack test.

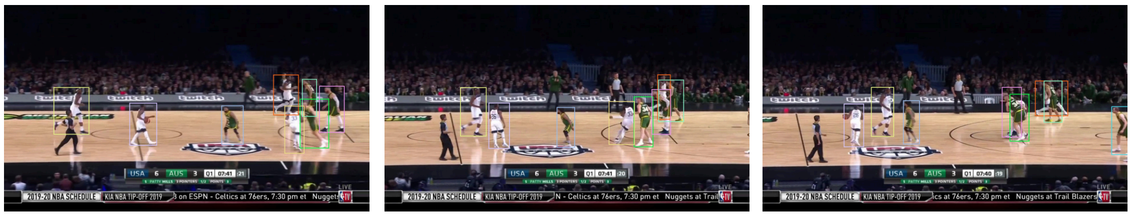
	Metrics																
Sequence	HOTA	DetA	AssA	DetRe	DetPr	AssRe	AssPr	LocA	MOTA	MOTP	MODA	IDSW	IDF1	IDR	IDP	IDTP	IDFN	IDFP
dancetrack0003	69.2	80.3	59.7	83.6	92.2	64.1	83.3	91.4	90.1	90.6	90.4	32.0	73.6	70.2	77.5	9499.0	4034.0	2764.0
dancetrack0009	38.7	75.2	20.0	79.9	86.8	22.5	68.0	88.0	88.9	86.7	89.7	79.0	35.7	34.3	37.3	3214.0	6166.0	5412.0
dancetrack0011	62.4	89.5	43.5	92.5	94.8	44.2	93.1	93.7	96.2	93.3	96.5	19.0	55.2	54.5	55.9	3646.0	3043.0	2875.0
dancetrack0013	50.0	86.5	28.9	89.5	95.4	31.6	86.4	94.5	92.7	94.4	93.2	20.0	47.1	45.6	48.6	1960.0	2334.0	2071.0
dancetrack0017	70.1	90.6	54.3	93.3	95.8	58.4	86.0	94.4	96.1	93.9	96.2	18.0	68.5	67.6	69.4	7466.0	3575.0	3293.0
dancetrack0021	62.3	83.8	46.4	87.3	90.7	49.9	82.5	90.5	95.5	89.8	95.8	21.0	63.2	62.0	64.5	5160.0	3157.0	2842.0
dancetrack0022	51.6	81.2	32.8	89.7	87.1	39.3	71.3	91.2	93.0	89.7	93.5	30.0	48.9	49.6	48.2	2949.0	2996.0	3173.0
dancetrack0028	39.1	75.1	20.4	79.2	87.2	24.2	62.3	87.9	88.4	86.5	89.2	85.0	38.6	36.9	40.6	4061.0	6958.0	5947.0
dancetrack0031	58.7	80.3	43.0	84.9	91.7	50.6	67.9	92.3	88.9	91.8	89.4	48.0	59.2	57.0	61.5	5598.0	4227.0	3500.0
dancetrack0036	47.2	82.3	27.1	87.0	91.4	31.5	76.4	92.0	90.6	90.8	91.1	43.0	45.6	44.5	46.8	4028.0	5015.0	4580.0
dancetrack0038	84.8	93.2	77.2	94.8	97.3	78.2	96.8	94.9	97.1	94.7	97.3	10.0	88.5	87.4	89.7	5084.0	734.0	584.0
dancetrack0040	91.6	93.9	89.3	96.3	96.4	91.4	96.4	94.6	99.8	94.2	99.8	3.0	95.5	95.5	95.6	11027.0	520.0	512.0
dancetrack0042	43.6	69.9	27.2	74.1	84.5	30.7	66.1	85.9	86.1	83.6	86.9	50.0	44.6	41.8	47.7	2945.0	4093.0	3226.0
dancetrack0046	42.3	63.8	28.1	68.4	80.2	32.7	64.9	83.3	77.6	81.0	78.4	123.0	50.7	47.0	55.1	7639.0	8614.0	6228.0
dancetrack0048	60.0	85.7	42.0	90.1	91.8	47.2	78.2	91.9	94.6	91.0	95.0	28.0	61.7	61.1	62.2	4670.0	2971.0	2833.0
dancetrack0050	48.0	74.3	31.1	78.1	87.8	35.5	71.9	88.2	86.9	86.5	87.4	47.0	49.3	46.6	52.4	4947.0	5680.0	4502.0
dancetrack0054	77.9	93.1	65.2	95.9	96.0	71.4	82.2	94.3	98.6	94.0	98.7	5.0	79.9	79.8	79.9	3365.0	850.0	844.0
dancetrack0056	71.2	90.3	56.2	93.7	93.5	63.3	73.5	92.4	98.8	91.7	98.9	5.0	69.7	69.7	69.6	4185.0	1816.0	1824.0
dancetrack0059	62.4	83.1	46.8	89.6	90.3	52.6	79.3	92.1	97.0	91.4	97.3	15.0	63.8	63.5	64.1	4016.0	2304.0	2254.0
dancetrack0060	82.6	90.4	75.5	93.3	94.0	77.3	94.7	93.0	97.5	92.5	97.6	5.0	86.1	85.8	86.4	3972.0	660.0	627.0
dancetrack0064	72.2	88.6	58.9	92.8	92.9	63.5	85.0	92.5	99.1	92.2	99.2	13.0	78.3	78.3	78.4	5870.0	1628.0	1617.0
dancetrack0067	84.6	91.8	77.9	94.9	94.8	81.0	92.7	93.4	99.2	92.7	99.3	4.0	87.5	87.5	87.5	5393.0	771.0	772.0
dancetrack0070	71.2	84.1	60.3	88.8	91.7	63.5	88.9	91.9	95.0	92.1	95.2	24.0	72.9	71.7	74.1	8231.0	3247.0	2874.0
dancetrack0071	92.2	93.9	90.6	95.9	97.3	92.2	97.5	95.5	98.4	95.1	98.5	3.0	96.8	96.1	97.5	2897.0	118.0	74.0
dancetrack0076	56.4	79.0	40.4	84.6	88.0	45.2	77.2	89.7	90.8	88.1	91.1	25.0	54.5	53.4	55.6	3762.0	3281.0	3003.0
dancetrack0078	82.3	92.2	73.5	94.3	96.5	76.9	91.4	94.4	97.6	93.8	97.7	7.0	84.9	84.0	85.9	6060.0	1158.0	997.0
dancetrack0084	63.9	85.3	47.9	89.1	91.7	52.4	80.0	91.3	95.1	90.4	95.4	49.0	67.5	66.5	68.4	10904.0	5491.0	5033.0
dancetrack0085	38.4	63.8	23.3	68.6	83.0	26.4	65.8	86.0	75.9	84.2	76.8	150.0	40.2	36.7	44.5	5945.0	10241.0	7418.0
dancetrack0088	65.0	76.7	55.2	80.7	91.9	58.1	90.4	92.1	83.4	91.5	83.9	32.0	72.8	68.4	77.9	3981.0	1843.0	1129.0
dancetrack0089	60.4	77.6	47.2	83.8	89.1	52.0	81.7	91.8	83.7	91.1	84.3	42.0	63.6	61.8	65.6	4131.0	2557.0	2164.0
dancetrack0091	63.2	83.7	47.7	87.1	94.3	49.8	84.0	93.7	90.6	92.7	91.2	27.0	61.3	59.0	63.9	2737.0	1901.0	1548.0
dancetrack0092	51.7	89.1	30.0	91.8	96.1	33.4	81.8	95.0	94.3	94.5	94.6	25.0	43.3	42.3	44.3	3436.0	4687.0	4322.0
dancetrack0093	56.5	76.9	41.6	82.3	89.7	46.5	79.1	91.5	84.1	90.5	84.7	42.0	57.7	55.3	60.3	4100.0	3317.0	2703.0
dancetrack0095	49.1	58.8	41.1	62.3	87.0	48.1	69.5	88.9	66.7	87.3	67.8	78.0	58.7	50.3	70.3	3527.0	3481.0	1489.0
dancetrack0100	49.1	72.4	33.4	75.8	91.3	36.4	76.8	91.2	81.2	89.4	81.9	37.0	50.3	46.0	55.5	2436.0	2857.0	1955.0
Total	61.6	80.3	47.5	84.8	90.8	51.6	80.5	91.2	90.3	90.2	90.7	1244.0	61.8	59.8	64.1	172841.0	116325.0	96989.0

D Visualization

a) MOT17: Sequence MOT-17-01



b) DanceTrack: Sequence dancetrack003



c) SportsMOT: Sequence v_-9kabh1K8UA_c008

Figure D.1: Visualization of the predictions on MOT17, DanceTrack and SportsMOT test sets.