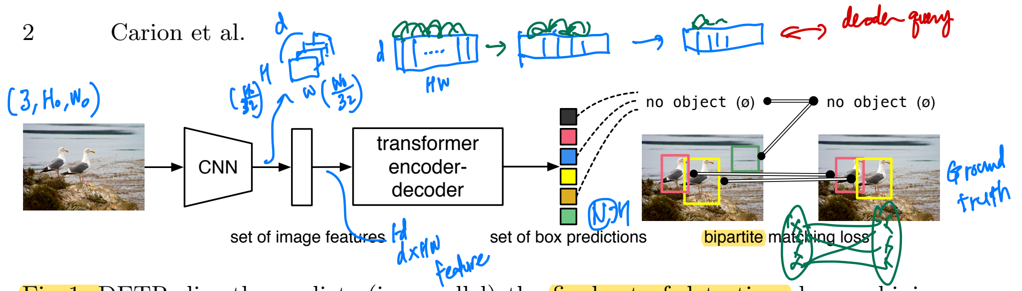
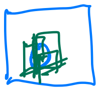


Fig. 1: DETR directly predicts (in parallel) the final set of detections by combining a common CNN with a transformer architecture. During training, bipartite matching uniquely assigns predictions with ground truth boxes. Prediction with no match should yield a "`no object`" (∅) class prediction.



We streamline the training pipeline by viewing object detection as a direct set prediction problem. We adopt an encoder-decoder architecture based on transformers [47], a popular architecture for sequence prediction. The self-attention mechanisms of transformers, which explicitly model all pairwise interactions between elements in a sequence, make these architectures particularly suitable for specific constraints of set prediction such as removing duplicate predictions.

Our DEtection TRansformer (DETR, see Figure 1) predicts all objects at once, and is trained end-to-end with a set loss function which performs bipartite matching between predicted and ground-truth objects. DETR simplifies the detection pipeline by dropping multiple hand-designed components that encode prior knowledge, like spatial anchors or non-maximal suppression. Unlike most existing detection methods, DETR doesn't require any customized layers, and thus can be reproduced easily in any framework that contains standard CNN and transformer classes.[1].

Compared to most previous work on direct set prediction, the main features of DETR are the conjunction of the bipartite matching loss and transformers with (non-autoregressive) parallel decoding [29,12,10,8]. In contrast, previous work focused on autoregressive decoding with RNNs [43,41,30,36,42]. Our matching loss function uniquely assigns a prediction to a ground truth object, and is invariant to a permutation of predicted objects, so we can emit them in parallel.

We evaluate DETR on one of the most popular object detection datasets, COCO [24], against a very competitive Faster R-CNN baseline [37]. Faster R-CNN has undergone many design iterations and its performance was greatly improved since the original publication. Our experiments show that our new model achieves comparable performances. More precisely, DETR demonstrates significantly better performance on large objects, a result likely enabled by the non-local computations of the transformer. It obtains, however, lower performances on small objects. We expect that future work will improve this aspect in the same way the development of FPN [22] did for Faster R-CNN.

Training settings for DETR differ from standard object detectors in multiple ways. The new model requires extra-long training schedule and benefits

[1] In our work we use standard implementations of Transformers [47] and ResNet [15] backbones from standard deep learning libraries.

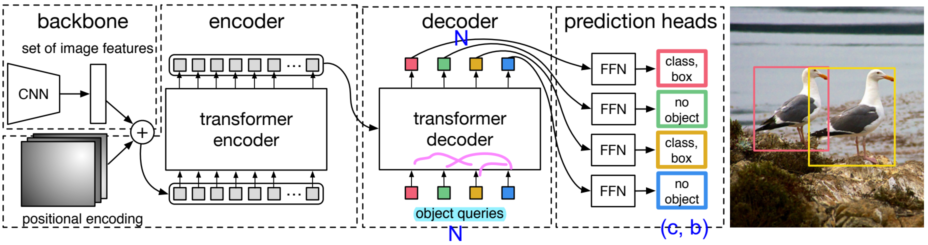


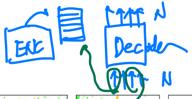
Fig. 2: DETR uses a conventional CNN backbone to learn a 2D representation of an input image. The model flattens it and supplements it with a positional encoding before passing it into a transformer encoder. A transformer decoder then takes as input a small fixed number of learned positional embeddings, which we call *object queries*, and additionally attends to the encoder output. We pass each output embedding of the decoder to a shared feed forward network (FFN) that predicts either a detection (class and bounding box) or a "`no object`" class.

**Transformer decoder.** The decoder follows the standard architecture of the transformer, transforming $N$ embeddings of size $d$ using multi-headed self- and encoder-decoder attention mechanisms. The difference with the original transformer is that our model decodes the $N$ objects in parallel at each decoder layer, while Vaswani et al. [47] use an autoregressive model that predicts the output sequence one element at a time. We refer the reader unfamiliar with the concepts to the supplementary material. Since the decoder is also permutation-invariant, the $N$ input embeddings must be different to produce different results. These input embeddings are learnt positional encodings that we refer to as *object queries*, random, initially and similarly to the encoder, we add them to the input of each attention layer. The $N$ object queries are transformed into an output embedding by the decoder. They are then *independently* decoded into box coordinates and class labels by a feed forward network (described in the next subsection), resulting $N$ final predictions. Using self- and encoder-decoder attention over these embeddings, the model globally reasons about all objects together using pair-wise relations between them, while being able to use the whole image as context.

**Prediction feed-forward networks (FFNs).** The final prediction is computed by a 3-layer perceptron with ReLU activation function and hidden dimension $d$, and a linear projection layer. The FFN predicts the normalized center coordinates, height and width of the box w.r.t. the input image, and the linear layer predicts the class label using a softmax function. Since we predict a fixed-size set of $N$ bounding boxes, where $N$ is usually much larger than the actual number of objects of interest in an image, an additional special class label $\varnothing$ is used to represent that no object is detected within a slot. This class plays a similar role to the "background" class in the standard object detection approaches.

**Auxiliary decoding losses.** We found helpful to use auxiliary losses [1] in decoder during training, especially to help the model output the correct number

20 obj queries they learned



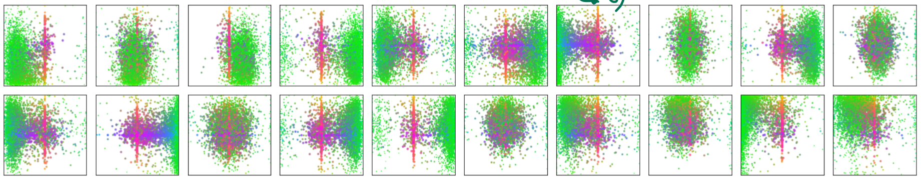


Fig. 7: Visualization of all box predictions on all images from COCO 2017 val set for 20 out of total $N = 100$ prediction slots in DETR decoder. Each box prediction is represented as a point with the coordinates of its center in the 1-by-1 square normalized by each image size. The points are color-coded so that green color corresponds to small boxes, red to large horizontal boxes and blue to large vertical boxes. We observe that each slot learns to specialize on certain areas and box sizes with several operating modes. We note that almost all slots have a mode of predicting large image-wide boxes that are common in COCO dataset.

also cooperating each other by self attn
(But, one box for each obj)

simple ablations of different losses (using the same weighting every time), but other means of combining them may achieve different results.

### 4.3 Analysis

**Decoder output slot analysis** In Fig. 7 we visualize the boxes predicted by different slots for all images in COCO 2017 val set. DETR learns different specialization for each query slot. We observe that each slot has several modes of operation focusing on different areas and box sizes. In particular, all slots have the mode for predicting image-wide boxes (visible as the red dots aligned in the middle of the plot). We hypothesize that this is related to the distribution of objects in COCO.

**Generalization to unseen numbers of instances.** Some classes in COCO are not well represented with many instances of the same class in the same image. For example, there is no image with more than 13 giraffes in the training set. We create a synthetic image[3] to verify the generalization ability of DETR (see Figure 5). Our model is able to find all 24 giraffes on the image which is clearly out of distribution. This experiment confirms that there is no strong class-specialization in each object query.

### 4.4 DETR for panoptic segmentation

semantic segmentation + instance segmentation

Panoptic segmentation [19] has recently attracted a lot of attention from the computer vision community. Similarly to the extension of Faster R-CNN [37] to Mask R-CNN [14], DETR can be naturally extended by adding a mask head on top of the decoder outputs. In this section we demonstrate that such a head can be used to produce panoptic segmentation [19] by treating stuff and thing classes

[3] Base picture credit: https://www.piqsels.com/en/public-domain-photo-jzlwu

Semantic classes can be either things (objects with a well-defined shape, e.g. car, person) or stuff (amorphous background regions, e.g. grass, sky).

Meinhardt, T., Kirillov, A., Leal-Taixe, L., Feichtenhofer, C.: Trackformer: Multiobject tracking with transformers. CVPR 2022

In contrast, TrackFormer casts the entire tracking objective into a single set prediction problem, applying attention not only for the association step. It jointly reasons about track initialization, identity, and spatio-temporal trajectories. We only rely on feature-level attention and avoid additional graph optimization and appearance/motion models.

## 3. TrackFormer

We present TrackFormer, an end-to-end trainable multi-object tracking (MOT) approach based on an encoder-decoder Transformer [50] architecture. This section describes how we cast MOT as a set prediction problem and introduce the new *tracking-by-attention* paradigm. Furthermore, we explain the concept of *track queries* and their application for frame-to-frame data association.

### 3.1. MOT as a set prediction problem

Given a video sequence with $K$ individual object identities, MOT describes the task of generating ordered tracks $T_k = (b^k_{t_1}, b^k_{t_2}, \dots)$ with bounding boxes $b_t$ and track identities $k$. The subset $(t_1, t_2, \dots)$ of total frames $T$ indicates the time span between an object entering and leaving the scene. These include all frames for which an object is occluded by either the background or other objects. ✓

In order to cast MOT as a set prediction problem, we leverage an encoder-decoder Transformer architecture. Our model performs online tracking and yields per-frame object bounding boxes and class predictions associated with identities in four consecutive steps:

(i) Frame-level feature extraction with a common CNN backbone, *e.g.*, ResNet-50 [17].

(ii) Encoding of frame features with self-attention in a Transformer encoder [50].

( obj query: learn
track query: autoregressive

(iii) Decoding of queries with self- and encoder-decoder attention in a Transformer decoder [50].

(iv) Mapping of queries to box and class predictions using multilayer perceptrons (MLP).

Objects are implicitly represented in the decoder *queries*, which are embeddings used by the decoder to output bounding box coordinates and class predictions. The decoder alternates between two types of attention: (i) self-attention over all queries, which allows for joint reasoning about the objects in a scene and (ii) encoder-decoder attention, which gives queries global access to the visual information of the encoded features. The output embeddings accumulate bounding box and class information over multiple decoding layers. The permutation invariance of Transformers requires additive feature and object encodings for the frame features and decoder queries, respectively.

### 3.2. Tracking-by-attention with queries

The total set of output embeddings is initialized with two types of query encodings: (i) static object queries, which allow the model to initialize tracks at any frame of the video, and (ii) autoregressive track queries, which are responsible for tracking objects across frames.

The simultaneous decoding of object and track queries allows our model to perform detection and tracking in a unified way, thereby introducing a new *tracking-by-attention* paradigm. Different tracking-by-X approaches are defined by their key component responsible for track generation. For tracking-by-detection, the tracking is performed by computing/modelling distances between frame-wise object detections. The tracking-by-regression paradigm also performs object detection, but tracks are generated by regressing each object box to its new position in the current frame. Technically, our TrackFormer also performs regression in the mapping of object embeddings with MLPs. However, the actual track association happens earlier via attention in the Transformer decoder. A detailed architecture overview which illustrates the integration of track and object queries into the Transformer decoder is shown in the appendix.

**Track initialization.** New objects appearing in the scene are detected by a fixed number of $N_{\text{object}}$ output embeddings each initialized with a static and learned object encoding referred to as *object queries* [7]. Intuitively, each object query learns to predict objects with certain spatial properties, such as bounding box size and position. The decoder self-attention relies on the object encoding to avoid duplicate detections and to reason about spatial and categorical relations of objects. The number of object queries is ought to exceed the maximum number of objects per frame. DETR

**Track queries.** In order to achieve frame-to-frame track generation, we introduce 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 in an autoregressive manner.

For this purpose, each new object detection initializes a track query with the corresponding output embedding of the previous frame. The Transformer encoder-decoder performs attention on frame features and decoder queries *continuously updating* the instance-specific representation of an object‘s identity and location in each track query embedding. Self-attention over the joint set of both query types allows for the detection of new objects while simultaneously avoiding re-detection of already tracked objects.

In Figure 2, we provide a visual illustration of the track query concept. The initial detections in frame $t = 0$ spawn new track queries following their corresponding objects to frame $t$ and beyond. To this end, $N_{\text{object}}$ ob-

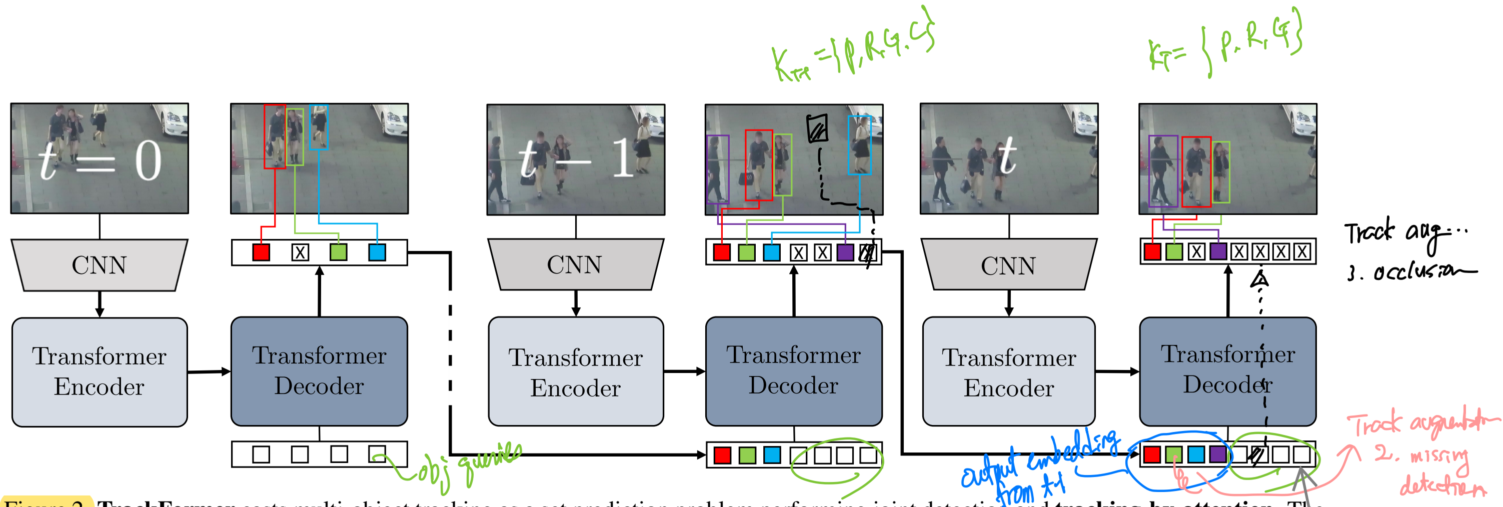


Figure 2. **TrackFormer** casts multi-object tracking as a set prediction problem performing joint detection and **tracking-by-attention**. The architecture consists of a CNN for image feature extraction, a Transformer [50] encoder for image feature encoding and a Transformer decoder which applies self- and encoder-decoder attention to produce output embeddings with bounding box and class information. At frame $t = 0$, the decoder transforms $N_{\text{object}}$ object queries (white) to output embeddings either initializing new autoregressive **track queries** or predicting the background class (crossed). On subsequent frames, the decoder processes the joint set of $N_{\text{object}} + N_{\text{track}}$ queries to follow or remove (blue) existing tracks as well as initialize new tracks (purple).

concatenation?

ject queries (white) are decoded to output embeddings for potential track initializations. Each valid object detection $\{\boldsymbol{b}_0^0, \boldsymbol{b}_0^1, \dots\}$ with a classification score above $\sigma_{\text{object}}$, *i.e.*, output embedding not predicting the background class (crossed), initializes a new track query embedding. Since not all objects in a sequence appear on the first frame, the track identities $K_{t=0} = \{0, 1, \dots\}$ only represent a subset of all $K$. For the decoding step at any frame $t > 0$, track queries initialize additional output embeddings associated with different identities (colored). The joint set of $N_{\text{object}} + N_{\text{track}}$ output embeddings is initialized by (learned) object and (temporally adapted) track queries, respectively.

The Transformer decoder transforms the entire set of output embeddings at once and provides the input for the subsequent MLPs to predict bounding boxes and classes for frame $t$. The number of track queries $N_{\text{track}}$ changes between frames as new objects are detected or tracks removed. Tracks and their corresponding query can be removed either if their classification score drops below $\sigma_{\text{track}}$ or by non-maximum suppression (NMS) with an IoU threshold of $\sigma_{\text{NMS}}$. A comparatively high $\sigma_{\text{NMS}}$ only removes strongly overlapping duplicate bounding boxes which we found to not be resolvable by the decoder self-attention.

**Track query re-identification.** The ability to decode an arbitrary number of track queries allows for an attention-based short-term re-identification process. We keep decoding previously removed track queries for a maximum number of $T_{\text{track-reid}}$ frames. During this *patience window*, track queries are considered to be inactive and do not contribute to the trajectory until a classification score higher than $\sigma_{\text{track-reid}}$ triggers a re-identification. The spatial information embedded into each track query prevents their application for long-term occlusions with large object movement, but, nevertheless, allows for a short-term recovery from track loss. This is possible without any dedicated re-identification training; and furthermore, cements TrackFormer's holistic approach by relying on the same attention mechanism as for track initialization, identity preservation and trajectory forming even through short-term *occlusions*.

## 3.3. TrackFormer training

For track queries to work in interaction with object queries and follow objects to the next frame, TrackFormer requires dedicated frame-to-frame tracking training. As indicated in Figure 2, we train on two adjacent frames and optimize the entire MOT objective at once. The loss for frame $t$ measures the set prediction of all output embeddings $N = N_{\text{object}} + N_{\text{track}}$ with respect to the ground truth objects in terms of class and bounding box prediction.

The set prediction loss is computed in two steps:

(i) Object detection on frame $t-1$ with $N_{\text{object}}$ object queries (see $t = 0$ in Figure 2).

(ii) Tracking of objects from (i) and detection of new objects on frame $t$ with all $N$ queries.

The number of track queries $N_{\text{track}}$ depends on the number of successfully detected objects in frame $t-1$. During training, the MLP predictions $\hat{y} = \{\hat{y}_j\}_{j=1}^{N}$ of the output embeddings from step (iv) are each assigned to one of the ground truth objects $y$ or the background class. Each $y_i$ represents a bounding box $b_i$, object class $c_i$ and identity $k_i$. step (ii)?

**Bipartite matching.** The mapping $j = \pi(i)$ from ground truth objects $y_i$ to the joint set of object and track query predictions $\hat{y}_j$ is determined either via track identity or costs based on bounding box similarity and object class. For the