CHALMERS
UNIVERSITY OF TECHNOLOGY

Next Generation Multitarget Trackers: Random Finite Set Methods vs Transformer-based Deep Learning

Next Generation Multitarget Trackers: Random Finite Set Methods vs Transformer-based Deep Learning

Juliano Pinto, Georg Hess, William Ljungbergh, Yuxuan Xia, Lennart Svensson, Henk Wymeersch
Department of Electrical Engineering, Chalmers University of Technology, Sweden
Email: juliano@chalmers.se

***Abstract*—Multitarget Tracking (MTT) is the problem of tracking the states of an unknown number of objects using noisy measurements, with important applications to autonomous driving, surveillance, robotics, and others. In the model-based Bayesian setting, there are conjugate priors that enable us to express the multi-object posterior in closed form, which could theoretically provide Bayes-optimal estimates. However, the posterior involves a super-exponential growth of the number of hypotheses over time, forcing state-of-the-art methods to resort to approximations for remaining tractable, which can impact their performance in complex scenarios. Model-free methods based on deep-learning provide an attractive alternative, as they can, in principle, learn the optimal filter from data, but to the best of our knowledge were never compared to current state-of-the-art Bayesian filters, specially not in contexts where accurate models are available. In this paper, we propose a high-performing deep-learning method for MTT based on the Transformer architecture and compare it to two state-of-the-art Bayesian filters, in a setting where we assume the correct model is provided. Although this gives an edge to the model-based filters, it also allows us to generate unlimited training data. We show that the proposed model outperforms state-of-the-art Bayesian filters in complex scenarios, while matching their performance in simpler cases, which validates the applicability of deep-learning also in the model-based regime. The code for all our implementations is made available at https://github.com/JulianoLagana/MT3.**

***Index Terms*—Multitarget tracking, Multi-object tracking, Transformers, Deep learning, Multi-object conjugate prior.**

I. Introduction

Multitarget tracking (MTT) is the problem of tracking a varying number of targets/objects through time based on noisy measurements. Methods capable of such functionality are important for several applications, such as pedestrian tracking [1], autonomous driving [2], air traffic control [3], oceanography [4], and many others. The main challenge of MTT comes from the fact that objects may appear and disappear from the scene at every time-step, and that there can be missed and false detections. Therefore, a key challenge of MTT is to address the unknown correspondence between targets and measurements, a task referred to as data association.

In situations where we have access to accurate multitarget models, and observe only low-dimensional, individual object detections, the state of the art is achieved by model-based Bayesian methods. Typical examples are the Poisson multi-Bernoulli mixture (PMBM) filter [5] and the delta-generalized labeled multi-Bernoulli (δ-GLMB) [6] filter, which use the random finite set framework to formulate the tracking problem. These methods make use of multi-object conjugate priors to obtain a closed-form expression for the multi-object posterior and can, in theory, provide Bayes-optimal estimates. In practice, however, the complexity of the data association and track management makes it intractable to compute the optimal solution [5], [6]. Thus, these methods must resort to approximations of the true posterior density, which become increasingly inaccurate as the data association problem becomes more challenging.

In recent years, MTT algorithms based on deep learning have emerged as attractive alternatives to traditional Bayesian methods, usually optimizing a model with a large number of parameters by minimizing the empirical risk on an annotated dataset for the problem at hand [7]. Such methods have achieved unmatched performance in settings where no models are available and high-dimensional measurements are provided [8], [9], by extracting informative features from the data to aid in estimating the quantities of interest. However, to the best of our knowledge these methods have never been directly compared to the performance of current state-of-the-art model-based Bayesian methods such as the PMBM and δ-GLMB filters, specially not in contexts where accurate models of the environment are available.

In this paper, we study how deep learning MTT compares to the state-of-the-art Bayesian filters PMBM and δ-GLMB, in the model-based regime. In particular, we propose the MultiTarget Tracking Transformer (MT3), a high-performing, specific type of neural network for MTT based on the Transformer [10] architecture, and evaluate it in scenarios where we assume that the models are correct. In principle, this context gives the model-based methods an edge because they have access to the true models. However, at the same time we can use the available models to generate as much training data as necessary for MT3, allowing high-capacity architectures that can potentially obtain better performance. Our results show that MT3 performs competitively to PMBM and δ-GLMB with reasonable computational complexity in a relatively simple scenario, while outperforming them when the task becomes more complex. This demonstrates the applicability of deep-learning-based MTT methods also in the model-based regime.

The remainder of the paper is organized as follows. In Section II, we present the multitarget models used and the problem formulation. In Section III, a short background on the Transformer architecture is provided. Section IV details the proposed MT3 algorithm, and Section V the simulation results and ablation studies, followed by conclusions in Section VI.

Notations: Throughout the paper, we use the following notations: scalars are denoted by lowercase or uppercase letters with no special typesetting x, vectors by lowercase boldface letters $\mathbf{x}$, matrices by uppercase boldface letters $\mathbf{X}$, and sets by uppercase blackboard-bold letters $\mathbb{X}$. Sequences of scalars are abbreviated as $x_{1:n}$, and sequences of vectors as $\mathbf{x}_{1:n}$. In addition, we define $\mathbb{N}_a = \{i \in \mathbb{N} \mid i \leq a\}, a \in \mathbb{N}$.

II. Multitarget Models and Problem Formulation

In this work we follow the standard multitarget transition and observation models for point objects [11, Chap. 5], without target spawning. The state vector for object i at time-step t is $\mathbf{x}_i^t \in \mathbb{R}^{d_x}$, and $\mathbb{X}^t$ is the set of all object states at time-step t. New objects arrive according to a Poisson point process (PPP) with birth intensity $\lambda_b(\mathbf{x})$, while existing objects depart according to independent and identically distributed (iid) Markovian processes where the survival probability is $p_s(\mathbf{x})$. The motion models for the objects are also iid Markovian processes, where the single-object transition density is denoted $f(\mathbf{x}^{t+1} \mid \mathbf{x}^t)$.

At every time-step, each of the existing objects may give rise to at most one measurement (and each measurement is the result of at most one object), where the probability of detection in state $\mathbf{x}$ is $p_d(\mathbf{x})$. Clutter measurements arrive according to a PPP with intensity λ_c, independent of the existing objects or true measurements. Each non-clutter/true measurement is independent of all other objects and measurements conditioned on its corresponding target, and the single object measurement likelihood is denoted $\mathbf{g}(\mathbf{z}^t \mid \mathbf{x}^t)$, $\mathbf{z}^t \in \mathbb{R}^{d_z}$. Lastly, the set of all measurements (true and clutter) generated at time-step t is denoted $\mathbb{Z}^t$. For this paper we focus on the problem of multitarget estimation using a moving window: the estimation of $\mathbb{X}^T$ given knowledge of the measurements from τ time-steps in the past until the current time $[\mathbb{Z}_{T-\tau}, \cdots, \mathbb{Z}_T]$.

III. Background on Transformers

The Transformer architecture, first introduced in [10], is a type of neural network tailored for sequence-to-sequence function approximation, making it well-suited to the MTT problem. Its structure is comprised of an encoder and a decoder, as shown in Fig. 1. The Transformer processes the input sequence[1] $\mathbf{x}_{1:n}$ into an output sequence $\mathbf{y}_{1:k}$ in a learnable way, and its main building block is the *self-attention layer*. The optimization of the trainable parameters of the Transformer architecture is typically done by stochastic gradient descent on some loss function $\mathcal{L}_T(\mathbf{y}_{1:k}, \mathbf{g}_{1:k})$, that compares the output sequence with a ground-truth sequence $\mathbf{g}_{1:k}$. This section provides a background review on the self-attention layer and the encoder/decoder modules of the Transformer model.

A. Self-Attention Layer

The self-attention layer is the main building block of the Transformer architecture and processes an input sequence $\mathbf{x}_{1:n}$

[1]The symbols $\mathbf{x}$, $\mathbf{X}$, $\mathbf{y}$, $\mathbf{Y}$ used for this section are not connected to the rest of the paper; we make use of the standard notation from the Transformer literature (e.g. $\mathbf{x}$ for the input to layers/modules, and $\mathbf{y}$ for their outputs).

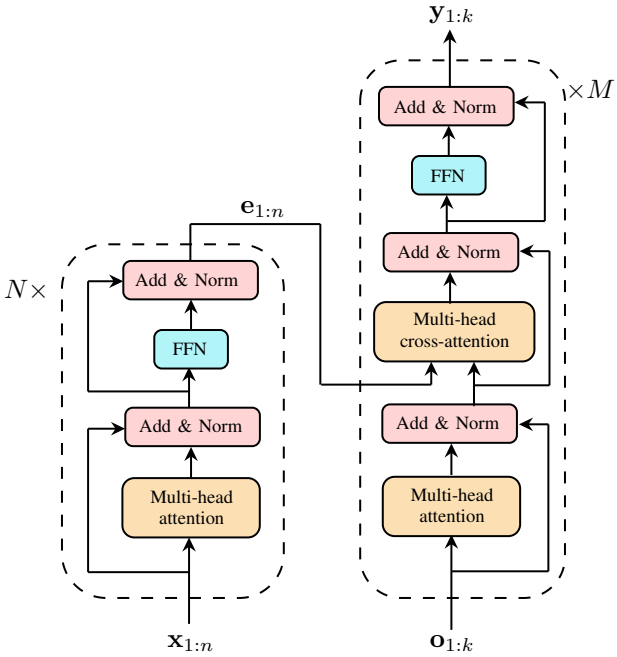

Fig. 1. Simplified diagram illustrating the Transformer architecture. Encoder on the left, containing N encoder blocks, processes the input sequence $\mathbf{x}_{1:n}$ into embeddings $\mathbf{e}_{1:n}$. DETR decoder on the right, containing M decoder blocks, uses the embeddings $\mathbf{e}_{1:n}$ produced by the encoder together with the object queries $\mathbf{o}_{1:k}$ to predict the output sequence $\mathbf{y}_{1:k}$.

into an output sequence $\mathbf{y}_{1:n}$, where $\mathbf{x}_i, \mathbf{y}_i \in \mathbb{R}^d, i \in \mathbb{N}_n$. We also introduce $\mathbf{X} = \begin{bmatrix}\mathbf{x}_1, \cdots, \mathbf{x}_n\end{bmatrix} \in \mathbb{R}^{d \times n}$ and similarly $\mathbf{Y} = \begin{bmatrix}\mathbf{y}_1, \cdots, \mathbf{y}_n\end{bmatrix} \in \mathbb{R}^{d \times n}$.

The first step is to compute three linear transformations of the input,

$$\mathbf{Q} = \mathbf{W}_Q \mathbf{X}, \ \mathbf{K} = \mathbf{W}_K \mathbf{X}, \ \mathbf{V} = \mathbf{W}_V \mathbf{X}, \tag{1}$$

referred to as queries, keys, and values, respectively, and where $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d \times d}$ are the learnable parameters of the self-attention layer. The output is constructed using the queries, keys, and values as

$$\mathbf{Y} = \mathbf{V} \cdot \text{Softmax-c}\left(\frac{\mathbf{K}^\top \mathbf{Q}}{\sqrt{d}}\right), \tag{2}$$

where Softmax-c is the column-wise application of the Softmax function, defined as

$$[\text{Softmax-c}(\mathbf{Z})]_{i,j} = \frac{e^{\mathbf{Z}_{i,j}}}{\sum_{k=1}^{d} e^{\mathbf{Z}_{k,j}}}; \quad i \in \mathbb{N}_d, \ j \in \mathbb{N}_n$$

for $\mathbf{Z} \in \mathbb{R}^{d \times n}$. Note that each output $\mathbf{y}_i$ of the self-attention layer directly depends on every input $\mathbf{x}_i$. This allows the model to learn about the pair-wise relations among all the elements of the input sequence, using a constant number of model parameters that does not depend on the sequence length. Furthermore, compound applications of the self-attention layer result in more complex and improved representations of each element in the sequence, allowing for more complicated relations to be learned. These properties are beneficial for tackling the data association problem in MTT, allowing the model to

leverage on complex, long-range patterns in the sequence of measurements when detecting objects.

In practice, Transformers use so-called multi-headed self-attention layers (shown in orange in Fig. 1), where $\mathbf{X}$ is fed to n_h different self-attention layers (with independent learnable parameters) in parallel, generating n_h different values $\mathbf{Y}_1, \cdots, \mathbf{Y}_{n_h}$. The final output is then computed by vertically stacking the results and applying a linear transformation to reduce the dimensionality:

$$\mathbf{Y} = \mathbf{W}^0 \begin{bmatrix} \mathbf{Y}_1 \\ \vdots \\ \mathbf{Y}_{n_h} \end{bmatrix} \tag{3}$$

where $\mathbf{W}^0 \in \mathbb{R}^{d \times dn_n}$ is a learnable parameter of the multi-head self-attention layer. Finally, $\mathbf{Y}$ is converted back to a sequence $\mathbf{y}_{1:n} = \mathrm{MultiAttention}(\mathbf{x}_{1:n})$. These multi-headed self-attention layers are then used to form the two main modules of a Transformer model: the encoder and the decoder.

B. Transformer Encoder

The encoder module of the Transformer model is the part responsible for transforming the input sequence into a sequence where each element has now a representation (embedding) that depends on all other elements in the sequence. Specifically, this is done by stacking N "encoder blocks" on top of each other, as shown in the left of Fig. 1. The output for encoder layer $l \in \mathbb{N}_N$ is computed as

$$\mathbf{t}_{1:n}^{(l)} = \mathrm{MultiAttention}(\mathbf{x}_{1:n}^{(l-1)}) \tag{4}$$

$$\tilde{\mathbf{t}}_{1:n}^{(l)} = \mathrm{LayerNorm}(\mathbf{x}_{1:n}^{(l-1)} + \mathbf{t}_{1:n}^{(l)}) \tag{5}$$

$$\mathbf{x}_{1:n}^{(l)} = \mathrm{LayerNorm}(\tilde{\mathbf{t}}_{1:n}^{(l)} + \mathrm{FFN}(\tilde{\mathbf{t}}_{1:n}^{(l)})) \ , \tag{6}$$

where $\mathbf{x}_{1:n}^{(0)}$ is the input sequence $\mathbf{x}_{1:n}$, MultiAttention is a multiheaded self-attention layer, FFN is a fully-connected feed-forward network, and LayerNorm is a Layer Normalization layer [12]. The output of the last layer $\mathbf{x}_{1:n}^{(N)}$ is the encoded input sequence, denoted $\mathbf{e}_{1:n}$ in Fig. 1. Once the encoder is trained, each $\mathbf{e}_i$, $i \in \mathbb{N}_n$ will contain an improved representation of $\mathbf{x}_i$ that potentially summarizes the relevant information from all other $\mathbf{x}_j$, $j \in \mathbb{N}_n \setminus i$, in a way that is helpful for the task at hand. In MTT, for example, it can contain relevant information about other measurements from the same object at different time steps.

As one can infer from this type of architecture, the encoder blocks are permutation-equivariant, and can be seen as learning mappings between input and output sets. However, in many applications (e.g. translation, sentiment analysis, MTT), there is important information encoded in the order of the elements in the input sequence. To allow the Transformer model to leverage on such information, one usually adds a "positional encoding" $\mathbf{q}_{1:n}$ vector to the input of every encoder (and often decoder) layer, using $\mathbf{x}_{1:n}^{(l)} + \mathbf{q}_{1:n}$, in the subsequent computations instead of $\mathbf{x}_{1:n}$, where $\mathbf{q}_i = f_p(i)$. The function f_p can either be fixed, usually with sinusoidal components [10], or learnable [13].

C. DETR Decoder

The Transformer decoder is the module responsible for using the embeddings computed by the encoder to predict the output sequence $\mathbf{y}_{1:k}$. Different Transformer decoder structures have been proposed for different problems [10], [14], [15], and the one used in this paper is similar to the one proposed in [13]. Specifically, instead of performing predictions autoregressively as in [10], the DEtection TRansformer (DETR) [13] decoder takes as input the encoded input sequence and a sequence of "object queries" $\mathbf{o}_{1:k}$ (learnable vectors), and computes all of its predictions in parallel. Each object query $\mathbf{o}_i$ learns to attend to aspects of the embeddings $\mathbf{e}_{1:n}$ which are helpful for predicting the output $\mathbf{y}_i$. In MTT, for example, an object query $\mathbf{o}_i$ can potentially learn to only attend to the embeddings of the true measurements from a certain object, making it possible for the decoder self-attention layers to perform the easier task of regression.

As shown in Fig 1, the decoder is comprised of M "decoder blocks" stacked on top of each other, where the input of the next block is the output of the previous one. Similarly to the encoder, the input is processed by layers of self-attention followed by an FFN, with normalization layers in-between. The Cross-attention layer shown in Fig 1 is a regular multi-head attention layer as described in section III-A, with the difference that the matrices $\mathbf{K}$ and $\mathbf{V}$ in (1) are respectively computed by multiplying $\mathbf{W}_K$ and $\mathbf{W}_V$ with $\mathbf{e}_{1:n}$, while $\mathbf{Q}$ is computed as usual, by multiplying $\mathbf{W}_Q$ with the output from the previous normalization layer. All of the subsequent computations are the same as regular multiheaded self-attention.

IV. MTT using Transformers

This section describes our proposed transformer architecture MT3: MultiTarget Tracking with Transformers. We approach the task of model-based MTT using deep learning, where the available transition and observation models are used for generating unlimited training data. In comparison to traditional MTT algorithms, we sidestep the need for using recursive pruning strategies that impair estimation performance by proposing a network structure able to implicitly reason about the data association hypotheses across the entire sequence, given all measurements.

A. MT3 Architecture

MT3 uses an encoder-decoder architecture for processing the input measurement sequence, as shown in Fig. 2. The idea is that the Transformer encoder can process the measurement sequence, generating a new, improved representation for each measurement capable of encoding helpful properties such as which measurements come from the same object, which are clutter measurements, etc. We feed these new representations into a modified DETR decoder, which can then leverage on them for implicitly performing soft data associations using cross-attention, and subsequent state estimation with the decoder self-attention.

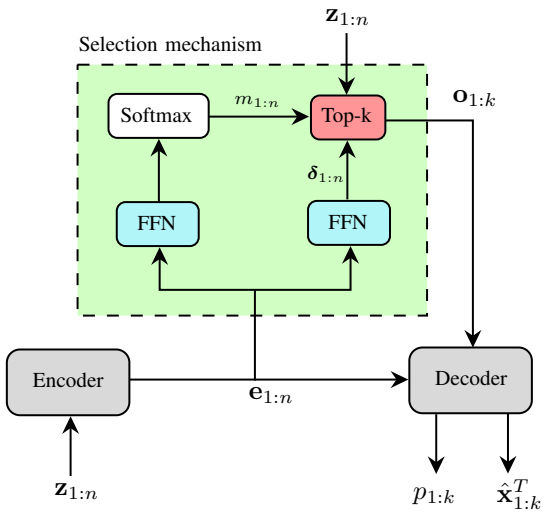

Fig. 2. High-level diagram of the MT3 architecture. The measurements $\mathbf{z}_{1:n}$ are processed by the encoder, producing embeddings $\mathbf{e}_{1:n}$. These embeddings are used by a selection mechanism for generating object queries $\mathbf{o}_{1:k}$. Lastly, $\mathbf{e}_{1:n}$ and $\mathbf{o}_{1:k}$ are used by the decoder for predicting the object states, $\hat{\mathbf{x}}^T_{1:k}$, and corresponding existence probabilities $p_{1:k}$.

Concretely, the measurements $[\mathbb{Z}^{T-\tau}, \cdots, \mathbb{Z}^T]$ are collected in a sequence

$$\mathbf{z}_{1:n} = [\mathbf{z}_1^{T-\tau}, \cdots, \mathbf{z}_{n^{T-\tau}}^{T-\tau}, \cdots, \mathbf{z}_1^T, \cdots, \mathbf{z}_{n^T}^T], \tag{7}$$

where $n^t \doteq |\mathbb{Z}^t|$, $n = \sum_{i=T-\tau}^{T} n_z^i$, and the elements of each $\mathbb{Z}^i$ are added in random order to the sequence. As shown in Fig. 2, the sequence $\mathbf{z}_{1:n}$ is fed to a Transformer encoder (N blocks), which generates embeddings for each measurement, resulting in a sequence $\mathbf{e}_{1:n}$. The sequence of embeddings $\mathbf{e}_{1:n}$ is then fed to a Transformer decoder (containing M encoder blocks), together with object queries $\mathbf{o}_{1:k}$ produced by a selection mechanism (explained in Section IV-B), to produce the output: estimated states for the objects present at time-step T, $\hat{\mathbf{x}}^T_{1:k}$, and corresponding existence probabilities, $p_{1:k}$.

B. Selection Mechanism

In order to produce the object queries $\mathbf{o}_{1:k}$, a selection mechanism similar to the two-stage encoder proposed in [16] is used: instead of using learned object queries that are the same for all sequences $\mathbf{z}_{1:n}$, the k most promising measurements in $\mathbf{z}_{1:n}$ are used as starting points, in order to facilitate the task of the decoder. To that end, each embedding $\mathbf{e}_i$ is fed to two different FFN layers, see Fig. 2. To select the most promising measurements, scores $m_i = \text{Softmax}(\text{FFN}_1(\mathbf{e}_i)) \in [0, 1],\ i \in \mathbb{N}_n$ are computed, and the measurements with the top-k-scores are then selected as candidates for the object queries. In order to allow more flexibility, the second FFN computes offsets $\boldsymbol{\delta}_i = \text{FFN}_2(\mathbf{e}_i) \in \mathbb{R}^{d_z},\ i \in \mathbb{N}_n$, which are added to the chosen measurements:

$$\tilde{\mathbf{z}}_i = \mathbf{z}_{r_i} + \boldsymbol{\delta}_{r_i}, \quad i \in \mathbb{N}_k\ . \tag{8}$$

where $\mathbf{r} = \text{argsort}(m_{1:n})$, and argsort is a function that returns the indices that would sort the input array (i.e., $m_{r_i} \geq m_{r_j} \iff i > j$, for $r = \text{argsort}(m_{1:n})$). The sequence $\tilde{\mathbf{z}}_{1:k}$ is then fed to an FFN layer, producing the object queries $\mathbf{o}_{1:k}$ for the decoder

$$\mathbf{o}_i = \text{FFN}(\tilde{\mathbf{z}}_i), \quad i \in \mathbb{N}_k\ , \tag{9}$$

where each element $\mathbf{o}_i \in \mathbb{R}^{d'}$, and $d' > d_z$ is a hyperparameter. Changing the value of the hyperparameter k then decides the trade-off between model capacity and computational requirements.

C. Iterative Refinement of State

We also adopt the idea of iterative refinement [16]–[19]. Instead of directly predicting the quantity of interest, each layer in the decoder predicts adjustments to the predictions made in the previous layer. In MT3, the output $\mathbf{y}^l_{1:k}$ from each decoder layer $l \in \mathbb{N}_M$ is fed through an FFN, producing adjustments $\boldsymbol{\Delta}^l_{1:k}$ for each of the k objects. The estimated position at layer L is then computed as

$$\hat{\mathbf{x}}_i^{T,L} = \tilde{\mathbf{z}}_i + \sum_{l=1}^{L} \boldsymbol{\Delta}_i^l\ , \quad i \in \mathbb{N}_k\ . \tag{10}$$

The existence probabilities, however, are not iteratively refined, and are computed directly by feeding the output of each decoder layer l through an FFN:

$$p_i^l = \text{FFN}(\mathbf{y}_i^l), \quad i \in \mathbb{N}_k\ . \tag{11}$$

Therefore, at each decoder layer l, estimates $\hat{\mathbf{x}}^{T,l}_{1:k}$ and existence probabilities $p^l_{1:k}$ are produced. The final output of MT3, shown in Fig. 2, is defined as the estimates computed at the last decoder layer M: $\hat{\mathbf{x}}^T_{1:k} = \hat{\mathbf{x}}^{T,M}_{1:k}$ and $p_{1:k} = p^M_{1:k}$.

D. Loss Function

We use the same loss function as defined in [13], which is based on the localization error and probabilities of missed and false targets, given the best match between ground-truth and predictions. Instead of applying such loss only to the final outputs $(\hat{\mathbf{x}}^T_{1:k}, p_{1:k})$, we apply it to the estimates produced at all the decoder layers, summing all contributions together. This was shown to improve performance for deep Transformer decoder structures, accelerating learning [13], [20].

Given a prediction made by the Transformer at decoder layer l:

$$\mathbf{a}^l_{1:k} = \left((\hat{\mathbf{x}}_1^{T,l}, p_1^l), ..., (\hat{\mathbf{x}}_k^{T,l}, p_k^l)\right)\ ,$$

and ground-truth states for all objects present at time-step T:

$$\mathbf{g}_{1:|\mathbb{X}^T|} = \left(\mathbf{x}_1^T, ..., \mathbf{x}_{|\mathbb{X}^T|}^T\right)\ ,$$

we assume[2], in the same fashion as [13], that $k \geq |\mathbb{X}^T|$ and then pad $\mathbf{g}_{1:|\mathbb{X}^T|}$ with $\varnothing$ elements so that $k = |\mathbb{X}^T|$. The best match is then computed as

$$\sigma^l_* = \arg\min_{\sigma} \sum_{i=1}^{k} \mathcal{L}_m(\mathbf{a}_i^l, \mathbf{g}_{\sigma(i)}) \tag{12}$$

[2]We choose a value of k which is large in comparison to the generative model, and enforce the inequality constraint during training by not adding more than k objects in a scene, k being the number of object queries of the model being trained. This restriction is only applied during training, during inference this loss needs not be computed.

where σ is a permutation function:

$$\sigma : \mathbb{N}_k \to \mathbb{N}_k \mid \sigma(i) = \sigma(j) \Rightarrow i = j \ ,$$

and $\mathcal{L}_m$ is the matching loss, defined as

$$\mathcal{L}_m(\mathbf{a}_i^l, \mathbf{g}_j) = \begin{cases} 0 & \text{if } \mathbf{a}_j^l = \varnothing \\ \|\hat{\mathbf{x}}_i^{T,l} - \mathbf{x}_i^T\| - p_i^l & \text{otherwise.} \end{cases} \tag{13}$$

The optimization problem in (12) can be efficiently solved using the Hungarian algorithm [21]. Given the best match σ_*^l at each layer l, the final loss used to train the model is:

$$\mathcal{L}_T(\mathbf{a}_{1:k}^{1:M}, \mathbf{g}_{1:k}) = \sum_{l=1}^{M} \sum_{i=1}^{k} \mathcal{L}(\mathbf{a}_i^l, \mathbf{g}_{\sigma_*^l(i)}) \tag{14}$$

where $\mathcal{L}$ is defined as

$$\mathcal{L}(\mathbf{a}_i^l, \mathbf{g}_j) = \begin{cases} -\log(1 - p_i) & \text{if } \mathbf{g}_j = \varnothing \\ \|\hat{\mathbf{x}}_i^{T,l} - \mathbf{x}_i^T\| - \log(p_i) & \text{otherwise.} \end{cases} \tag{15}$$

E. Contrastive Auxiliary Learning

Training on auxiliary tasks often helps the training process and the generalization performance of the final model, by leveraging on information helpful for the task which is present in the inputs, but not being directly used by the primary loss chosen for training [22]. In MTT, this could be the information of whether a measurement is clutter or not, or if it comes from the same object as another measurement. Although such information is helpful for solving the task, it is not directly used by the loss chosen in Section IV-D. In order to steer the model into quickly learning how to perform this task, we use an idea inspired by Supervised Contrastive Learning [23] to create the contrastive loss $\mathcal{L}_c$, with the aid of object labels for each of the measurements. The idea is to encourage the encoder to generate embeddings of measurements which are similar for measurements coming from the same object, but dissimilar for measurements coming from different objects. False measurements are treated as coming all from the same dummy object (and therefore their encoded representations should be similar to each other, but dissimilar to the representations of all other measurements).

Concretely, for a given sequence of measurements $\mathbf{z}_{1:n}$, let b_i be the object from which measurement $\mathbf{z}_i$ came from, $i \in \mathbb{N}_n$. We define $\mathbb{P}_i = \{j \in \mathbb{N}_n \mid j \neq i \ , \ b_i = b_j\}$, that is, $\mathbb{P}_i$ is the set of indices of measurements that came from the same object as $\mathbf{z}_i$. The auxiliary loss $\mathcal{L}_c$ is then defined as:

$$\mathcal{L}_c(\mathbf{u}_{1:n}, \mathbb{P}_{1:n}) = \alpha \sum_{i=1}^{n} \frac{-1}{|\mathbb{P}_i|} \sum_{i^+ \in \mathbb{P}_i} \log \frac{e^{\mathbf{u}_i^\top \mathbf{u}_{i^+}}}{\sum\limits_{j \in \mathbb{N}_n \setminus i} e^{\mathbf{u}_i^\top \mathbf{u}_j}} \tag{16}$$

where, $\mathbf{u}_{1:n}$ is a sequence of vectors computed by processing each $\mathbf{e}_i, i \in \mathbb{N}_n$ (embeddings of $\mathbf{z}_{1:n}$ computed by the encoder) with an FFN layer and normalizing to unit length, and $\alpha \geq 0$ is a hyperparameter used to control the importance of this task. Minimizing this loss can be understood as making $\mathbf{u}_i^\top \mathbf{u}_j$ large when $\mathbf{z}_i$ and $\mathbf{z}_j$ are from the same object, but small when they are from different objects. Training on the sum of this auxiliary loss and the loss defined in (14) accelerated learning and improved the final performance of the trained model in both tasks considered in Section V.

F. Preprocessing

We perform three steps of preprocessing: normalization, dimensionality augmentation, and addition of a temporal encoding, whose main aspects are explained in this subsection. For additional information we refer the reader to our code. For preprocessing, we first normalize all dimensions of each measurement vector $\mathbf{z}_i$ in $\mathbf{z}_{1:n}$ to be in $[0, 1]$, using the known field-of-view dimensions. Then, to use self-attention layers of higher dimensionality, we increase the dimension of each measurement vector using a linear transformation

$$\mathbf{z}_i' = \mathbf{W}\mathbf{z}_i + \mathbf{b}, \quad i \in \mathbb{N}_n \ , \tag{17}$$

where $\mathbf{z}_i' \in \mathbb{R}^{d'}$ is the dimensionality augmented measurement vector, d' was defined in Section IV-B, and $\mathbf{W} \in \mathbb{R}^{d' \times d_z}$, $\mathbf{b} \in \mathbb{R}^{d'}$ are learnable parameters. Lastly, we add positional encodings to the inputs of all the encoder and decoder layers, with learnable f_p as described in section III-B, similar to [13]. However, instead of using the position of the elements as input to f_p, we use the corresponding time-step for that position in the sequence.

V. Results

We evaluate the performance of MT3 in two tasks of different complexity using synthetic data, and compare it against two state-of-the-art Bayesian RFS filters: the PMBM filter [5], [24], and the δ-GLMB filter [6], both with linear/Gaussian implementation. Furthermore, we perform a series of ablation studies to evaluate the importance of different components of the MT3 architecture.

A. Definition of the Tasks

We compare the performance of the three trackers under two different variations, henceforth referred to as Task 1 and Task 2. For both tasks, $\tau = 20$, $\lambda_b(\mathbf{x}) = 10^{-3}$, $p_s(\mathbf{x}) = 0.95$, the field of view is the 2D square $[-10, 10] \times [-10, 10]$, and we use Poisson models with parameter λ_0 for the initial number of objects. The motion model used is the nearly constant velocity model, defined as:

$$f(\mathbf{x}^{t+1}|\mathbf{x}^t) = \mathcal{N}\left(\begin{bmatrix} \mathbf{I} & \mathbf{I}\Delta_t \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \mathbf{x}^t \ , \ \sigma_q^2 \begin{bmatrix} \mathbf{I}\frac{\Delta_t^3}{3} & \mathbf{I}\frac{\Delta_t^2}{2} \\ \mathbf{I}\frac{\Delta_t^2}{2} & \mathbf{I}\Delta_t \end{bmatrix}\right) \tag{18}$$

where $\mathbf{x}^{t+1}, \mathbf{x}^t \in \mathbb{R}^{d_x}$, $d_x = 4$ represents target position and velocity in 2D, $\Delta_t = 0.1$ is the sampling period, σ_q controls the magnitude of the process noise, and the state for newborn objects is sampled from $\mathcal{N}(\mathbf{0}, 3\mathbf{I})$. The linear Gaussian measurement model is used with measurement likelihood

$$g\left(\mathbf{z}^t \mid \mathbf{x}^t\right) = \mathcal{N}(\mathbf{H}\mathbf{x}^t, \mathbf{I}\sigma_z^2) \tag{19}$$

where $\mathbf{H}$ selects the position components from $\mathbf{x}^t$, and σ_z controls the magnitude of the measurement noise.

Task 1 has $\lambda_0 = 4$, $p_d = 0.9$, $\lambda_c = 0.05$, $\sigma_q = 0.5$, and $\sigma_z = 0.1$. We expect the model-based Bayesian trackers

to be able to approximate well the optimal solution in this context, making them a strong benchmark for evaluating the Transformer-based model. Task 2 has $\lambda_0 = 6$, $p_d = 0.8$, $\lambda_c = 0.075$, $\sigma_q = 0.9$, and $\sigma_z = 0.3$. The lower signal-to-noise ratio in this task makes it considerably harder for conventional MTT algorithms to perform well with a feasible computational complexity, since the number of probable hypothesis to keep track of grows considerably.

B. Algorithms

1) PMBM: The PMBM filter provides a closed-form solution for multiple point object tracking with standard multitarget models with Poisson birth. The unknown data associations lead to an intractably large number of terms in the PMBM density. To achieve computational tractability of the PMBM filter, it is necessary to reduce the number of PMBM parameters. First, gating is used to remove unlikely measurement-to-object associations, by thresholding the squared Mahalanobis distance, where the gating size is 20. Second, we use Murty's algorithm [25] to find up to 200 best assignments. Third, we prune multi-Bernoullis with weight smaller than 10^{-4}, Bernoulli components with probability of existence smaller than 10^{-5} and Poisson mixture components with weight smaller than 10^{-5}. Object state estimation is performed by obtaining the global hypothesis with highest weight at time-step T and reporting the means of the Bernoullis whose existence is above 0.5 (Estimator 1 in [5]). PMBM was run on a machine with 2x16 core Intel Xeon Gold 6226R CPU, amounting to average inference times per sample of 2.4s and 13.2s for Task 1 and Task 2, respectively.

2) δ-GLMB: The δ-GLMB filter provides a closed-form solution for MTT when the object birth model is a multi-Bernoulli (mixture). To achieve computational tractability of the δ-GLMB filter, gating and Murty's algorithm are implemented to remove unlikely measurement-to-object associations as in the PMBM filter. In addition, we prune multi-Bernoullis with weight smaller than 10^{-4}. Object states are extracted from the global hypothesis with the highest weight. Average inference times per sample on the same machine as above were 16.8s and 23.3s for Task 1 and Task 2, respectively

3) MT3: The architecture choice used for the encoder is the same as in [10], and for decoder the same as in [13]. We use 6 encoder and 6 decoder layers, and in all of them the multiheaded self-attention layers have 8 heads. All FFN layers in the encoder and decoder layers are comprised of 2048 hidden units, and we use a dropout rate of 0.1 during training. The increased state dimensionality d' is set to 256. The FFNs used for predicting measurement adjustments $\boldsymbol{\delta}_i$ in (8), state refinements $\boldsymbol{\Delta}^l_{1:k}$ in (10), object queries $\mathbf{o}_i$ in (9), and existence probabilities p^l_i in (11) are all comprised of single-layer neural networks with 128 hidden units, while the FFN used in the computation of $\mathbf{u}_i$'s in (16) has 256. Furthermore, in all our experiments we set the contrastive loss weight $\alpha = 4.0$, and we extract target estimates from $\hat{\mathbf{x}}^t_i$ with $p_i \geq 0.9$. MT3 was trained on a V100 GPU for approximately 110 hours on each task, amounting to 800k and 600k gradient descent steps for Task 1 and Task 2, respectively. The ADAM optimizer [26] was used, with a batch size of 32. Each batch of samples is generated using the known transition and observation models available for each task. The initial learning rate was set to $5 \cdot 10^{-5}$, and whenever the total loss value $\mathcal{L}_T$ did not decrease for 50k consecutive time-steps, the learning rate was reduced by a factor of 4. Once trained, inference times for MT3 on the same hardware were 0.032s and 0.035s per sample, for Task 1 and Task 2, respectively. The code used to define, train, and evaluate our model is made publicly available.

C. Performance Metrics

We evaluate the performance of each algorithm using the Generalized optimal sub-pattern assignment (GOSPA) metric [27] with $\alpha = 2$ between predictions $\hat{\mathbb{X}} = \{\hat{\mathbf{x}}^T_1, \cdots, \hat{\mathbf{x}}^T_{|\hat{\mathbb{X}}|}\}$ and the ground-truth states $\mathbb{X} = \{\mathbf{x}^T_1, \cdots, \mathbf{x}^T_{|\mathbb{X}|}\}$, defined as

$$d_p^{(c,2)}(\hat{\mathbb{X}}, \mathbb{X}) = \min_{\gamma \in \Gamma} \left(\underbrace{\sum_{(i,j)\in\gamma} d(\hat{\mathbf{x}}^T_i, \mathbf{x}^T_j)^p}_{\text{Localization}} + \underbrace{\frac{c^p}{2}(|\hat{\mathbb{X}}| - |\gamma|)}_{\text{False detections}} + \underbrace{\frac{c^p}{2}(|\mathbb{X}| - |\gamma|)}_{\text{Missed detections}} \right)^{\frac{1}{p}} \tag{20}$$

where the minimization is over assignment sets between the elements of X and Y, such that $\gamma \subseteq \{1, \cdots, |\hat{\mathbb{X}}|\} \times \{1, \cdots, |\mathbb{X}|\}$, while $(i,j), (i,j') \in \gamma \implies j = j'$, and $(i,j), (i',j) \in \gamma \implies i = i'$. In all our experiments we use $c = 2.0$, $p = 1$, and

$$d(\hat{\mathbf{x}}^T_i, \mathbf{x}^T_i) = \|\mathbf{H}\hat{\mathbf{x}}^T_i - \mathbf{H}\mathbf{x}^T_i\| \ . \tag{21}$$

We evaluate the estimation performance of the three trackers using Monte Carlo simulation with 1000 runs.

D. Task 1 Results

The resulting average GOSPA scores for Task 1 are shown in Table I, along with corresponding decompositions. The localization decomposition is provided after normalizing it by the number of detected objects, so as to provide an easier comparison between methods with large differences in missed detection rates.d In terms of total GOSPA error, PMBM performs best in this task, with MT3 achieving a close second place. The δ-GLMB filter attains worse performance than PMBM in this task, mainly because PMBM has a more efficient hypothesis structure than δ-GLMB [28]. MT3 obtains the lowest missed and false detection rates, which implies that it performs the data association task the best.

TABLE I
GOSPA AND ITS DECOMPOSITIONS FOR TASK 1.

Method	GOSPA	Localization	Missed	False
PMBM	**1.267**	0.102	0.632	0.195
δ-GLMB	1.863	**0.098**	1.137	0.335
MT3	1.277	0.141	**0.528**	**0.094**

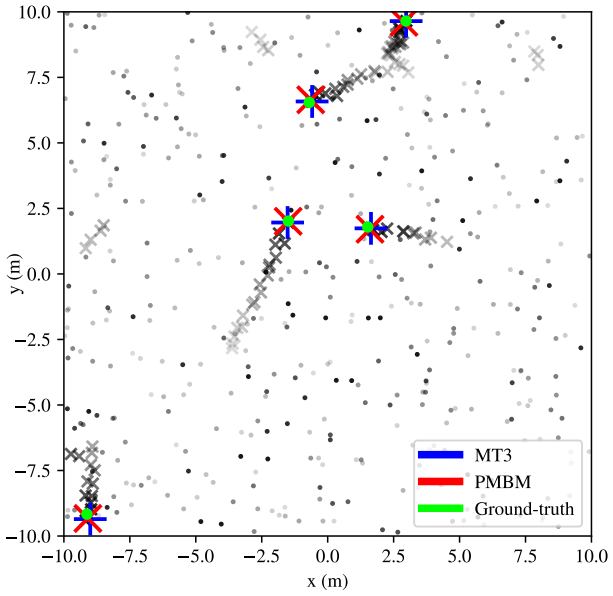

Fig. 3. Evaluation sample for Task 1. The measurements generated during the $\tau = 20$ simulation time-steps are shown in black, with their transparency illustrating the time-step in which they were taken (more opaque x's correspond to more recent measurements, closer to $t = T$). Clutter is depicted as small black circles and true measurements as black crosses.

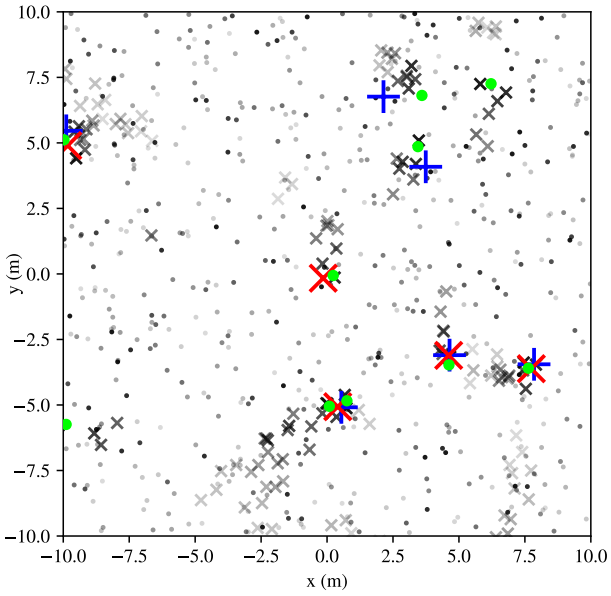

Fig. 4. Evaluation sample for Task 2. Same symbol convention from Fig. 3. Note the increased number of false and missed measurements, coupled with more measurement and motion noise, making the tracking considerably harder.

As a simple visual assessment of the task complexity as well as the trackers' performance, an evaluation sample from this task is plotted in Fig. 3. We show only the predictions for the best and second-best performing trackers to reduce visual clutter. From the figure we can see that neither PMBM nor MT3 had difficulty in finding all targets among the clutter. Note that the faint black crosses in the borders of the image correspond to measurements generated by objects which left the scene at a time-step prior to $t = T$, so no predictions are made for them.

E. Task 2 Results

The resulting average GOSPA scores for Task 2 are shown in Table II, along with corresponding decompositions. In this setting, MT3 outperforms the alternatives, having not only considerably lower missed detection rates, but also the lowest overall GOSPA score. Traditional methods have more difficulty in keeping track of all possible data association hypotheses, and approximations must be made so that the posterior computation remains feasible [11]. We hypothesize the Transformer model, in contrast, can efficiently learn to leverage complex time correlations in the input sequence to directly predict the object states, thus sidestepping the complicated posterior computation and resulting in better performance.

TABLE II
GOSPA AND ITS DECOMPOSITIONS FOR TASK 2.

Method	GOSPA	Localization	Missed	False
PMBM	4.075	0.3025	3.225	**0.163**
δ-GLMB	4.450	**0.280**	3.515	0.323
MT3	**3.662**	0.3767	**1.995**	0.364

An example evaluation sample from this task is shown in Fig 4, together with the predictions for the best and second-best performing trackers. In it, we can see that MT3 was still able to predict reasonable estimates for the position of most objects, and PMBM presented more missed detections, especially for newborn objects. For some detected objects (e.g., at approximately $(3, 7)$), although MT3 was able to detect it while PMBM failed to do so, MT3's state estimate seems suboptimal, judging from the trend in the measurements. This type of prediction may explain MT3's larger localization errors. From the results, we hypothesize that the Transformer architecture as used in MT3 is good at reasoning probable data association hypotheses, but future work could improve it for better state regression.

F. Ablation studies

To shed light on the importance of the design choices for the MT3 algorithm, we conducted a series of ablation studies of MT3 in Task 2:

1) *No contrastive loss*: the term α in (16) was set to zero.
2) *No iterative refinement*: $\mathbf{\Delta}^l_{1:k}$ in (10) was set to zeros for all $l \neq M$.
3) *No selection mechanism*: $\mathbf{o}_{1:k}$ in (9) is set to a learned lookup table (similar to [13]), no dependence on $\mathbf{z}_{1:n}$ nor $\mathbf{e}_{1:n}$.
4) *No intermediate decoder losses*: $\mathcal{L}(\mathbf{a}^l_i, \mathbf{g}_j)$ in (14) is set to zero for all $l \neq M$.

All other hyperparameters were kept constant, and the ablations were trained on exactly the same data, for the same number of gradient steps (600k). The average GOSPA scores

TABLE III
ABLATION STUDIES FOR THE MT3 ARCHITECTURE IN TASK 2.

Ablation	**GOSPA**
MT3 (original)	3.662
MT3, no contrastive loss	3.729
MT3, no iterative refinement	4.094
MT3, no selection mechanism	4.587
MT3, no intermediate decoder losses	5.055

for the original architecture and the ablations are shown in Table III, computed using 1000 Monte Carlo simulations. The results show not only that all of the considered components are indeed beneficial in terms of final GOSPA score, but also their relative importances. Removing the intermediate decoder losses has the strongest effect.

VI. CONCLUSION

In this work, we performed a comparison of two state-of-the-art model-based Bayesian methods for multitarget tracking to a deep learning approach based on the Transformer architecture. The evaluation showed that the proposed architecture, MT3, performs competitively to the model-based Bayesian methods, while being able to outperform them in more complicated tasks. Our results provide evidence that deep-learning models are suitable for the MTT task also in the model-based regime, where model-based Bayesian trackers have long been regarded as state of the art.

Interesting future research directions include developing training loss functions which are closer to the standard performance metric used in this setting (GOSPA), evaluating the quality of the uncertainties produced [29], and investigating the performance of MT3 and related algorithms under model mismatch. Furthermore, we note that MT3 can be easily generalized to other contexts outside of the model-based regime without drastic changes to the overall architecture.

ACKNOWLEDGMENT

This work was supported, in part, by a grant from the Chalmers AI Research Centre Consortium. Computational resources were provided by the Swedish National Infrastructure for Computing at C3SE, partially funded by the Swedish Research Council through grant agreement no. 2018-05973.

REFERENCES

[1] T. L. A. Nguyen, F. Bremond, and J. Trojanova, "Multi-object tracking of pedestrian driven by context," in *IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, 2016, pp. 23–29.

[2] A. Rangesh and M. M. Trivedi, "No blind spots: Full-surround multi-object tracking for autonomous vehicles using cameras and lidars," *IEEE Transactions on Intelligent Vehicles*, vol. 4, no. 4, pp. 588–599, 2019.

[3] Y. Li, S. Chen, and X. Jiang, "Multi-object tracking within air-traffic-control surveillance videos," in *Intelligent Visual Surveillance*. Springer Singapore, 2016, pp. 72–80.

[4] D. Walther, D. R. Edgington, and C. Koch, "Detection and tracking of objects in underwater video," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, 2004.

[5] Á. F. García-Fernández, J. L. Williams, K. Granström, and L. Svensson, "Poisson multi-Bernoulli mixture filter: Direct derivation and implementation," *IEEE Transactions on Aerospace Electronic Systems*, vol. 54, no. 4, pp. 1883–1901, 2018.

[6] B. Vo, B. Vo, and H. G. Hoang, "An efficient implementation of the generalized labeled multi-Bernoulli filter," *IEEE Transactions on Signal Processing*, vol. 65, no. 8, pp. 1975–1987, 2017.

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[8] P. Dendorfer, A. Osep, A. Milan, K. Schindler, D. Cremers, I. Reid, S. Roth, and L. Leal-Taixé, "MOTchallenge: A benchmark for single-camera multiple target tracking," *International Journal of Computer Vision*, pp. 1–37, 2020.

[9] G. Ciaparrone, F. L. 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, 2020.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, pp. 5998–6008, 2017.

[11] R. P. Mahler, *Advances in statistical multisource-multitarget information fusion*. Artech House, 2014.

[12] L. J. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," in *Neural Information Processing Systems Deep Learning Symposium*, 2016.

[13] 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*, vol. 12346. Springer, 2020, pp. 213–229.

[14] D. R. So, Q. V. Le, and C. Liang, "The evolved transformer," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 2019, pp. 5877–5886.

[15] H. Le, J. Pino, C. Wang, J. Gu, D. Schwab, and L. Besacier, "Dual-decoder transformer for joint automatic speech recognition and multilingual speech translation," in *COLING*. International Committee on Computational Linguistics, 2020, pp. 3520–3533.

[16] 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*, 2021.

[17] Z. Teed and J. Deng, "RAFT: Recurrent all-pairs field transforms for optical flow," in *European Conference on Computer Vision*, vol. 12347. Springer, 2020, pp. 402–419.

[18] R. N. Rajaram, E. Ohn-Bar, and M. M. Trivedi, "Refinenet: Iterative refinement for accurate object localization," in *IEEE Intelligent Transportation Systems Conference*, 2016, pp. 1528–1533.

[19] C. Zhang, G. Lin, F. Liu, R. Yao, and C. Shen, "Canet: Class-agnostic segmentation networks with iterative refinement and attentive few-shot learning," in *Conference on Computer Vision and Pattern Recognition*. Computer Vision Foundation / IEEE, 2019, pp. 5217–5226.

[20] R. Al-Rfou, D. Choe, N. Constant, M. Guo, and L. Jones, "Character-level language modeling with deeper self-attention," in *AAAI Conference on Artificial Intelligence*, 2019, pp. 3159–3166.

[21] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[22] R. Caruana, "Multitask learning," *Machine Learning*, vol. 28, no. 1, pp. 41–75, 1997.

[23] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," in *Advances in Neural Information Processing Systems*, vol. 33. Curran Associates, Inc., 2020, pp. 18 661–18 673.

[24] J. L. Williams, "Marginal multi-Bernoulli filters: RFS derivation of MHT, JIPDA, and association-based MeMber," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 51, no. 3, pp. 1664–1687, 2015.

[25] K. G. Murty, "An algorithm for ranking all the assignments in order of increasing cost," *Operations Research*, pp. 682–687, 1968.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.

[27] A. S. Rahmathullah, Á. F. García-Fernández, and L. Svensson, "Generalized optimal sub-pattern assignment metric," in *IEEE International Conference on Information Fusion (Fusion)*, 2017.

[28] Á. F. García-Fernández, Y. Xia, K. Granström, L. Svensson, and J. L. Williams, "Gaussian implementation of the multi-Bernoulli mixture filter," in *IEEE International Conference on Information Fusion (Fusion)*, 2019.

[29] J. Pinto, Y. Xia, L. Svensson, and H. Wymeersch, "An uncertainty-aware performance measure for multi-object tracking," *IEEE Signal Processing Letters*, 2021.