Why Good Detectors Can Make Bad Trackers: mAP Worse than Random Chance at Model Selection

Trenton Tabor[1], Tomasz Swierzewski[1], Casidhe Hutchison[1], Oren Wright[2], Eric Heim[2], and Christopher S. Timperley[1]

***Abstract*— Neural network models have emerged over the last decade as the standard for perception in many robotic systems. While powerful, these models are opaque and practitioners often rely on empirical evaluation using standard metrics and held-out test sets to understand their performance. However, such model-level testing ignores how the outputs of perception models will be used in downstream components within a robotic system. In this work, we create a new family of object detection metrics that can accurately evaluate performance when detectors are used for object tracking in a robotic system. We show that mean Average Precision (mAP), arguably the most common metric for evaluating object detectors, does not strongly correlate with tracking performance. We propose an alternative metric, MATCH'N METRIC, that makes few, simple assumptions about downstream tracking algorithms and does not require the implementation of a tracker to compute. We show empirically that across a number of object detection models and tracking algorithms that MATCH'N METRIC more accurately correlates with tracking performance than mAP. Equipped with this metric, practitioners can better evaluate object detectors for their ability to be used in robotic systems that require object tracking.**

I. INTRODUCTION

Perception in modern autonomous systems is largely driven by machine learning; specifically by deep neural network models. These models take high-dimensional sensor outputs, such as images, and produce inferences about those outputs, such as a labeling of all the pixels (semantic segmentation) or the location and class of objects (object detection) within the image. In turn, inferences made by ML models are consumed by other software components within the autonomous system. Components within the sense-plan-act cycle of modern autonomy [1], like planners and trackers, are commonly reliant on the outputs of ML-based perception models to perform reasoning tasks that are key to overall system performance.

Though ML models provide state-of-the-art perception capabilities, they also pose significant challenges in test and evaluation. Many of the standard software engineering approaches used to evaluate code (e.g., code review and static analysis) cannot be directly applied to neural network models whose parameters have no intuitive interpretation [2]. Instead, engineers often rely on empirical testing where models are evaluated using standard metrics against data held-out from training. While this gives quantitative evidence about how the model can perform, it does not inform how a model will affect downstream components. For instance, a model may fail to detect an object every other frame of a video, which would result in poor model evaluation, but a downstream tracker may still be able to accurately predict the track of the object. This highlights the potential for discrepancy between how the model was evaluated and its performance in the task it is meant to feed if larger system context is not considered.

In this work, we aim to more closely couple ML model evaluation with the downstream tasks that they are meant to facilitate. More specifically, we focus on the common case in robotics where object trackers are reliant on ML-based object detection models. We show that one of the most common metrics to evaluate object detection performance, Mean Average Precision (mAP), is often *not* correlated with tracking performance when model inferences are used as input to tracking algorithms. We propose an alternative metric based on very simple tracking assumptions that 1) correlates strongly with tracking performance, and 2) generalizes to different choices of downstream tracking algorithms. In doing so, we provide a new model evaluation metric, MATCH'N METRIC, that can be used to determine how fit an object detection model is for tracking, without burdening engineering teams with fully developing downstream tracking components or perform costly full system testing.

In this paper, we make the following contributions:

I. We perform an evaluation across 17 object detection models, 4 trackers, and 3 tracking metrics and show that common model-level object detection metrics, such as mAP, can result in a relative ranking of models that does not correlate to the relative performance of models when used for tracking.
II. Based on this observation, we develop a new family of metrics for tracker-free evaluation of object detection models in the context of object tracking. These metrics are based on few, simple assumptions about downstream trackers, but much more closely model how detectors will be used for tracking.
III. We empirically validate our metrics across the same object detection models, trackers, and tracking metrics as before and show that they are much better indicators of tracking performance than common object detection metrics.

Our approach facilitates model selection, development,

[1]School of Computer Science, Carnegie Mellon University, USA. {ttabor,tswierze,fhutchin}@nrec.ri.cmu.edu, ctimperley@cmu.edu

[2]Software Engineering Institute, Carnegie Mellon University, USA. owright@sei.cmu.edu, etheim@sei.cmu.edu

and refinement earlier in the engineering process (before a tracker has been selected and developed), without the inaccuracy that comes from the conventional approach of using model-level metrics.

Code used in our experiments is available at `https://github.com.squaresLab/MatchnMetric/tree/main`.

II. Background & Related Work

A. Object Detection and Tracking

The system class that we focus on is object tracking from a monocular camera system. We briefly provide an explanation of this system class and related work in this section.

Object Detection: A standard machine learning component is an object detector. As their input, detectors consume images, generally grids of floating point values representing the scene in front of a camera or other sensor. They produce lists of bounding boxes, axis-aligned rectangles in image pixel coordinates that are intended to fully enclose the pixels representing an object of interest. These bounding boxes may be extended with additional values indicating a classification and/or confidence. In this work, we filter out classifications to only our class of interest, `Person`. We also make use of a floating point confidence value, which represents something akin to likelihood of being this class under the training data distribution.

As a standard component, there are libraries of object detection models (e.g., [3], [4]) that have been pretrained on different public datasets (e.g., COCO [5]). While these pretrained models can be used off-the-shelf to detect certain classes of objects (represented in their training data) in general images, they often require fine-tuning to be suitable for many real-world applications.

Multi-Object Tracking: Object detectors are frequently used as components in a Multi-Object Tracking (MOT) system. MOT systems generally consume sensor data over time and produce a list of object detections for each sensor reading with an association between the most recent detections and previous detections (i.e., a tracking of objects over time). We focus on MOT in videos from monocular cameras, but many other sensors are commonly used for the detector, such as radar. Generally, these algorithms are expected to describe when a previously unseen object enters view, where it moves in the view, and when it leaves the view. We focus on trackers that rely on object detection bounding boxes (i.e., tracking by detection), but trackers that rely on images directly or on other ML components also exist [6].

B. Perception Evaluation

In this section we collect some of the related work in evaluation, including a brief overview of detector evaluation, tracker evaluation, and overall perception system evaluation.

1) Detection Evaluation: As a relatively mature component, object detection models are generally evaluated against metrics on data similar to their expected deployment domain. In the absence of a deployment domain, they are evaluated on standard benchmarks like COCO [5]. There are four common errors for an individual labeled bounding box that metrics are designed to summarize.

Missed Detections: no bounding box produced for object.

Poor Localization Accuracy: bounding box produced near object, with no other object it matches.

False Positive: bounding box produced with no nearby object present.

Misclassification: bounding box produced, but identified as different class with no instance of the class nearby.

These types of failures are summarized by metrics that are computed and compared to evaluate model performance. These metrics are often defined as part of an open challenge in object detection performance, such as the PASCAL Visual Object Classes Challenge [7] and COCO [5]. Each of these challenges came with a definition of performance based on an area under the Precision-Recall curve for detections matched to ground truth labels, referred to as a mean Average Precision (mAP). These metrics were not designed around a particular application.

In this work, we also discuss the notion of robustness in object detection. We consider robustness to refer to the worst-case performance under a list of possible image perturbations or degradations. Previously, we looked at robustness in object detection in the context of collision avoidance systems [8]. Others have also been examining robustness for detection in autonomous vehicle contexts [9], [10].

2) MOT Evaluation: MOT, as a superset of object detection, is able to make the same errors and additionally make association failures. Association failures include:

False Association: declaring a detection to be associated with a past object where it should not.

Missed Association: declaring a detection to have no association with past objects.

This set of 6 error types are summarized into different MOT metrics, such as HOTA, MOTA, and IDF1 [6]. Prior work has shown that these three metrics may only be weakly correlated with each other, as they each trade off these error types differently [6]. Additionally, prior work has also shown that when choosing a detector in a MOT system, the ranking for models under detector metrics may not be preserved when using those models in the same tracker under tracking metrics.

3) Other Work in System Level Perception V&V: One group used human in the loop testing, where humans could correct model output in order to estimate possible improvent to system-level performance [11]. In previous work, we have shown that, for autonomous systems performing object detection for collision avoidance in autonomy, model-level failure rates do not necessarily correlate with system/task failure rates [8]. We also demonstrated a similar result for keypoint detection for pose estimation [12]. Finally, we examined system level performance in the context of semantic segmentation model testing in simulation [13].

III. Approach

In this work, we examine the quality using model-level metrics (e.g., COCO mAP) to inform model selection for

the downstream task of tracking. We compare it to the ranking of models when used in actual trackers, and we also demonstrate an example of evaluating only a model with a simple heuristic to get much better model ranking on tracking performance. We introduce the simple idea of a MATCH'N METRIC, a metric that uses additional information in the ground truth in order to evaluate what a machine learning model is capable of in the context of system-level performance.

Our approach is straightforward: assume a perfect tracker. Like in VOC mAP [7] detector evaluation, we match the ground truth labels to the detections. We use standard Hungarian matching with an IoU threshold of 0.5 to match detection bounding boxes to label bounding boxes, maximizing total confidence. Instead of computing a detector score with these matched detections, we copy the ground truth track IDs into the matched detections (discarding any detections without a ground truth match). We can then calculate a tracker level metric (e.g., HOTA [6]) directly on the result. Figure 1 illustrates the data flow for our approach, compared to detector or full tracker evaluation. Our approach gives an optimistic/ideal performance of a tracking system using a given detector model. While this strongly over-estimates the performance of the tracking system, it does so in a consistent manner across models, resulting in an accurate proxy metric.

This performs better than standard detector model metrics at estimating relative system performance; detector metrics often lead to poor model selection. Additionally, this approach generalizes across any tracker metric, as it creates an intermediate that looks like a tracker output. This allows for better (early) predictions of task performance in the absence of implementation details (i.e., which tracker is used).

While the driving motivation behind MATCH'N METRIC is to eliminate the dependency between model development and tracker development during engineering of the system, it is notable that MATCH'N METRIC also provides a computational advantage (c.f. full tracker evaluations), which can be be particularly consequential if a large number of models are being evaluated. For example, for HOTA, MATCH'N METRIC is between 25 to 60 times faster than running the tracker and computing HOTA.

IV. EVALUATION

To evaluate the effectiveness of our approach, we answer the following research questions:

RQ1 Do model-level object detection metrics effectively predict tracking performance?

RQ2 Is MATCH'N METRIC effective at informing detector model selection in the absence of tracker?

RQ3 Is MATCH'N METRIC an effective predictor of tracking metrics in the presence of perturbations?

To answer our questions, we assessed the nominal and robust performance of 4 trackers (OC-SORT [14], PKF [15], SORT [16], and ByteTrack [17]) combined with 17 pretrained detector models[1] on 74 videos from PersonPath22 [18], a labeled, multi-person tracking dataset containing static camera footage from predominantly urban scenes. We describe our methodology below.

Methodology: Nominal Performance

To assess the effectiveness of proxy measures under nominal conditions, we first obtained bounding box predictions from each detector model across all frames in the dataset. We then used those predictions to measure the model-level performance of each detector in terms of COCO style mAP, PASCAL VOC mAP, and a modified ROC curve (ROC_A) [8]. Next, we fed the predictions from each model into the different trackers to assess the task-level (i.e., tracking) performance of 72 detector–tracker pairs according to three metrics: HOTA (Higher Order Tracking Accuracy), MOTA (Multi-Object Tracking Accuracy), and IDF1 (Identification F1 Score) [6].

To determine the best performance of each detector–tracker pair on each task-level metric, we swept several values of the detector confidence threshold to find the highest score for each model. Each evaluation of a metric requires running the tracker on all bounding boxes above that confidence. These confidence values were generated by taking 20 equal sized quantiles of the set of all confidence values produced by that model run on all images in the dataset. This provided a reasonabl tradeoff between resolution and time to complete. Since OC-SORT, ByteTrack, PKF have constraints on the minimum allowed confidence threshold, we discard of the confidence values less than 0.1 as a post-process. The final score for each detector–tracker pair is the maximum value achieved across this set of minimum detection confidence parameters.

Methodology: Robust Performance

To assess model and task robust performance under perturbed conditions, we used Gorgon, a custom image mutation library, to mutate the dataset. We generated 19 variants of the dataset by applying the mutations described in Table I. Figure 2 shows an example of different Gorgon mutations applied to a single frame from the dataset. We then followed the same experimental procedure used to determine nominal performance on each mutated dataset.

We use the results from each dataset to compute robust versions of each detector and task-level metric. We measure the area under the Worst-Case curve [8] for metrics involving two competing signals: ROC_A and VOC-mAP. For COCO-mAP, we modify this procedure to account for the different IoU thresholds; we compute the robust mAP score for each IoU, then average them, same as nominal COCO-mAP. For ROC_A, VOC-mAP and COCO-mAP, we generate the robust area under the curve by finding the worst of each score (across nominal and all perturbations) at each evaluation

[1]Pretrained detector models were sourced from `https://github.com/Deeplite/deeplite-torch-zoo` [3] and `https://github.com/ultralytics` [4].

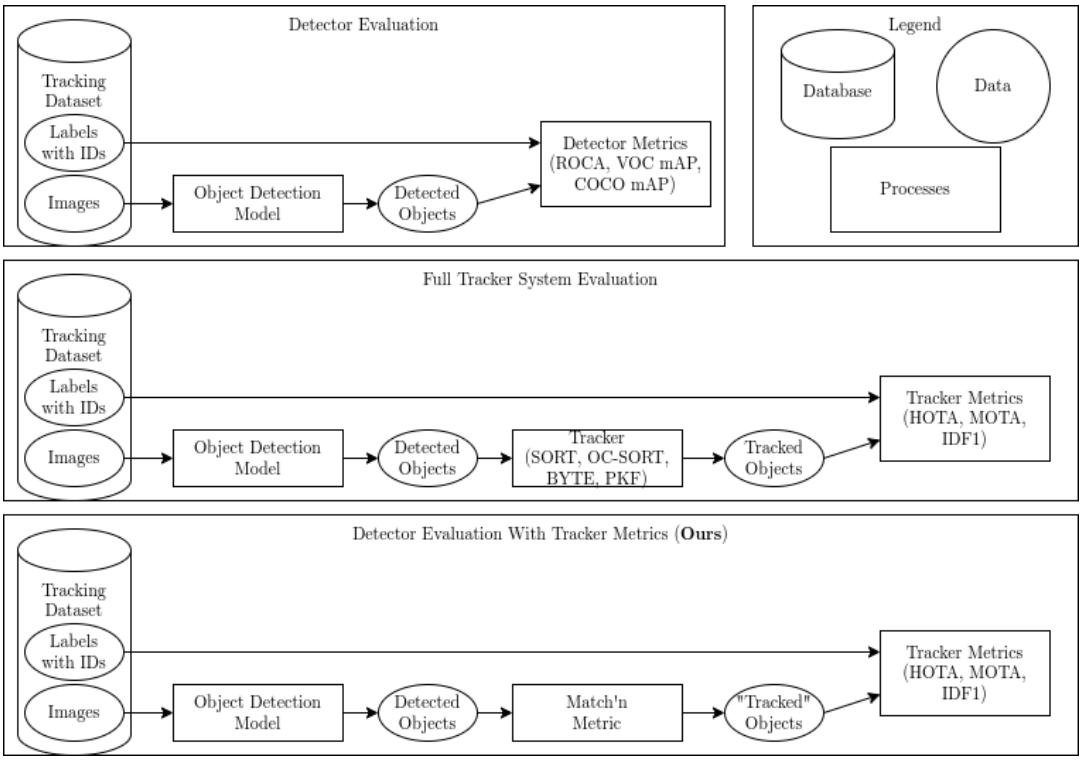

Fig. 1: Illustration of the different dataflow options for evaluating a detector for a tracking system. Either evaluation is on the detector itself, evaluation is on the final tracker result based on the detector, or in MATCH'N METRIC the evaluation is on detector results with additional ground truth information.

#	Mutation Name	Parameters
1	No Mutation	—
2	BrightnessContrast	Contrast: 2.0x, Brightness: +100
3	BrightnessContrast	Contrast: 1.2x, Brightness: +50
4	BrightnessContrast	Contrast: 1.2x, Brightness: 0
5	BrightnessContrast	Contrast: 0.8x, Brightness: 0
6	ChannelDrop	Blue Channel
7	ChannelDrop	Green Channel
8	ChannelDrop	Red Channel
9	WhiteBalance	Red Magnification: 0.8
10	WhiteBalance	Red Magnification: 1.2
11	GaussianBlur	Blur Strength: 1.5
12	GaussianBlur	Blur Strength: 2.0
13	GaussianBlur	Blur Strength: 3.0
14	GaussianNoise	Seed: 1, Intensity: 10
15	GaussianNoise	Seed: 2, Intensity: 10
16	GaussianNoise	Seed: 3, Intensity: 10
17	GaussianNoise	Seed: 1, Intensity: 50
18	GaussianNoise	Seed: 2, Intensity: 50
19	GaussianNoise	Seed: 3, Intensity: 50
20	LensScratchMutator	—

TABLE I: Mutations applied to the dataset.

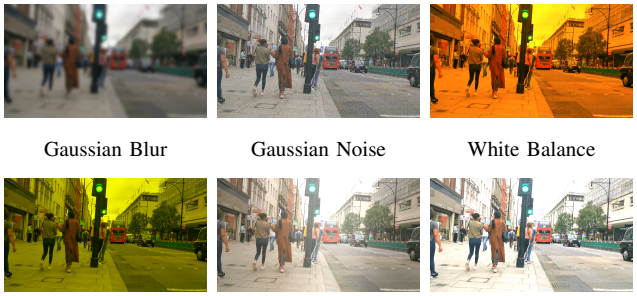

Fig. 2: Representative examples of different mutations applied to a single frame from the dataset.

threshold, creating a new tradeoff curve, then taking the area under that curve.

For the tracker metrics, since they combine multiple error types into a single value, we compute the robust version of the metric as the max-min value over all possible detector confidence thresholds. This is the maximum score under the worst case, where you do not know which environment you will deploy the model in. Fig. 3 shows an illustration of how this max-min can differ from the max nominal score.

Limitations and Threats to Validity

There are limitations to this evaluation. Firstly, we only demonstrate the issues with mAP on a single dataset of mono RGB video. Additionally our evaluation only uses a single class. Both of these could limit the scope of the conclusion, though we have no reason to believe that any of these limitations impact the validity of the results.

We note that 15 of the 17 models from are experiment are YOLO variants. This is because the sources of publicly available models that we used primarily provided YOLO-based models. For the two non-YOLO models (DETR variants) in

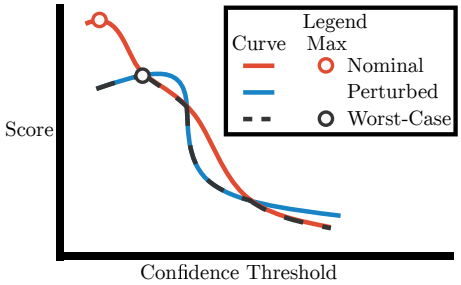

Fig. 3: Visual definitions for applications of different metrics in this work. Curves are not representative of any actual metric. Similar to our previous work, we define a robust version of a metric curve as the minimum over the range of shared confidence values [8]. With compound metrics, like HOTA, MOTA, and IDF1, the scores are plotted against confidence directly, instead of constructing a tradeoff between two independent metrics. The robust score is the Max of the Worst-Case Curve (i.e., the Max-Min over all curves).

our dataset, our conclusions remain the same, but results may differ for other families of object detectors.

Similarly, all of the trackers in our evaluation are based on SORT. Due to the limitations of our evaluation environment, we used only trackers that do not consume images as input and instead only used detections (i.e., tracking-by-detection trackers).

V. Results

Figure 4 plots the performance of every detector–tracker pair on the dataset according to different tracking metrics (y-axis) and proxies (x-axis). Table II measures the correlation between each tracking metric and its proxies (i.e., how well each proxy predicts tracking performance) according to its coefficient of determination (r^2) and Kendall rank correlation coefficient (τ).

A. RQ1: Model Performance vs. Task Performance

Across all detector–tracker pairs, we find that conventional model-level performance metrics (COCO mAP, VOC mAP, ROC_A) are weak predictors of tracking performance. The correlation (r^2) between these model level metrics and the performance of a tracker using that model is no better than 0.20 marginally (i.e. correlation for a single metric over all trackers simultaneously).

That is, simply looking at the general purpose performance of a detector outside of the context of tracking leads to poor estimation of how well a tracking system using that detector will perform.

Our approach of calculating tracking metrics directly by simulating a perfect tracker is a significantly better estimator of task performance than conventional proxy measures (i.e., detector metrics). As shown in Figure 4 and Table II, across all trackers and tracker metrics, MATCH'N METRIC is the most strongly correlated proxy with marginal r^2 values of 0.41 or above, showing that it is significantly more correlated with tracking performance than model-level metrics such as

Metric	HOTA			
Proxy	VOC mAP	COCO mAP	ROC_A	HOTA*
Marginal	0.13 [-0.04]	0.14 [-0.08]	0.12 [-0.14]	**0.58 [0.78]**
sort	0.15 [-0.04]	0.16 [-0.09]	0.16 [-0.16]	**0.73 [0.76]**
byte	0.16 [-0.04]	0.18 [-0.09]	0.19 [-0.16]	**0.74 [0.79]**
pkf	0.18 [-0.03]	0.19 [-0.07]	0.12 [-0.12]	**0.73 [0.78]**
oc-sort	0.18 [-0.03]	0.19 [-0.07]	0.13 [-0.12]	**0.72 [0.78]**
Metric	MOTA			
Proxy	VOC mAP	COCO mAP	ROC_A	MOTA*
Marginal	0.17 [-0.11]	0.19 [-0.14]	0.14 [-0.22]	**0.78 [0.75]**
sort	0.19 [-0.09]	0.22 [-0.13]	0.22 [-0.26]	**0.89 [0.72]**
byte	0.14 [-0.15]	0.15 [-0.16]	0.04 [-0.06]	**0.84 [0.72]**
pkf	0.21 [-0.09]	0.23 [-0.13]	0.20 [-0.26]	**0.90 [0.75]**
oc-sort	0.21 [-0.10]	0.24 [-0.15]	0.22 [-0.28]	**0.90 [0.79]**
Metric	IDF1			
Proxy	VOC mAP	COCO mAP	ROC_A	IDF1*
Marginal	0.10 [0.05]	0.11 [0.01]	0.16 [-0.18]	**0.41 [0.79]**
sort	0.13 [0.03]	0.14 [-0.01]	0.28 [-0.21]	**0.58 [0.79]**
byte	0.20 [0.00]	0.22 [-0.04]	0.28 [-0.21]	**0.67 [0.82]**
pkf	0.14 [0.09]	0.15 [0.04]	0.17 [-0.15]	**0.58 [0.79]**
oc-sort	0.12 [0.09]	0.14 [0.04]	0.18 [-0.15]	**0.55 [0.74]**

TABLE II: This table summarizes the results of our evaluation. The primary value is the r^2 coefficient of determination (1.0 is ideal), while the parenthetical value is the Kendall τ rank correlation (1.0 is ideal). The cells highlighted in bold show the best proxy for estimating the metric.

mAP. For a fixed tracker and metric, the contrast is even more pronounced, with VOC mAP having a mean r^2 correlation of 0.17, COCO mAP of 0.18 and ROC_A of 0.18, while MATCH'N METRIC has a mean r^2 of 0.74.

Insight: Model-level metrics are poorly correlated with tracker performance — model performance should not be relied on to determine downstream task performance.

B. RQ2: Decision Accuracy

Even if there is not a strong correlation between a tracking metric and a given proxy (as measured by r^2), the proxy can still be used to inform system design and development (i.e. model selection), provided that it still produces an accurate ordering of detectors.

The Kendall τ rank correlation scores associated with these proxies are in fact slightly negative, meaning that choosing a model based on the highest mAP will more often than not result in picking worse models over better ones. For example, we find that the lowest ranked models according to mAP score are the highest ranked in the actual tracking task performance, while the model with the worst tracking performance under HOTA and MOTA is in the top 3 recommended by mAP. On the other hand, our approach performs very well at model selection, with much higher mean scores across trackers (greater than 0.74 for each metric), indicating much better ranking performance, and therefore enabling better model selection.

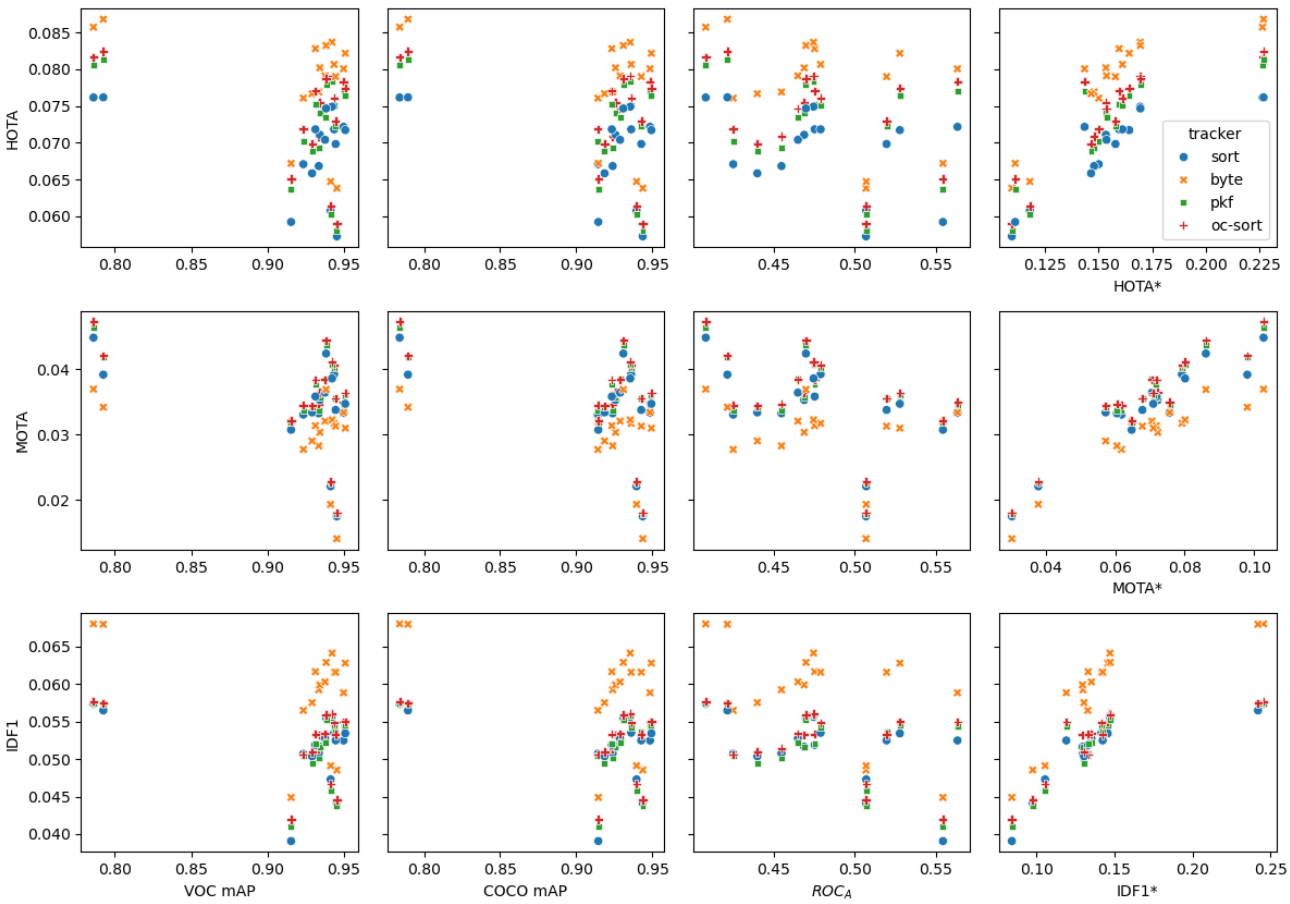

Fig. 4: Graph of correlation between tracker performance metrics (ground truth) and different proxy metrics. Each row of plots represents a different tracker performance metric (y-axis). Each column of plots represents a different proxy metric (x-axis). Each plot shows the correlation between a given tracker and proxy metric for all detector–tracker pairs.

Insight: Detector-level metrics can lead to poor selection. MATCH'N METRIC is much more effective at model selection.

C. RQ3: Robustness

We perform the same analysis of metric-proxy correlation in the robust context and find that, while conventional proxies improve somewhat, MATCH'N METRIC remains a significant improvement.

For HOTA and IDF1, the conventional proxies show improved r^2 correlation and Kendall τ, indicating increased model selection ability. In MOTA, only ROC_A showed substantial improvement. On the other hand, our approach shows *increased* performance under robustness analysis, which improvements on correlation for HOTA and IDF1 and rank score on all three tracking metrics. It remains significantly ahead of conventional proxies in both performance estimation and model selection.

Insight: Our approach performs even better as a decision metric when looking at robust system performance.

VI. CONCLUSION

We draw several key conclusions from our results.

First, conventional approaches to model selection do not perform well. AP and ROC_A do a poor job predicting performance of the eventual system (regardless of the tracking metric of interest), and do a poor job ranking models. This means that, if a system designer simply uses the model with the best AP off of the COCO leaderboards, or even downloads, trains, and evaluates models on their data, they may end up selecting a sub-optimal model.

Secondly, a tracker is not needed to assess tracker-level performance. The proposed approach of modeling a perfect tracker and evaluating tracker metrics is effective both as an estimator of system performance as well as a ranking criterion for model comparison and selection. This approach is more flexible by allowing evaluation of any tracking metric (beyond the three explored in this work), as it does not rely on any inherent properties of the metric. This allows developers to evaluate models in the system context much earlier in the project development lifecycle. It also allows for flexibility in the architecture of the system, letting developers swap out the tracker or model independently

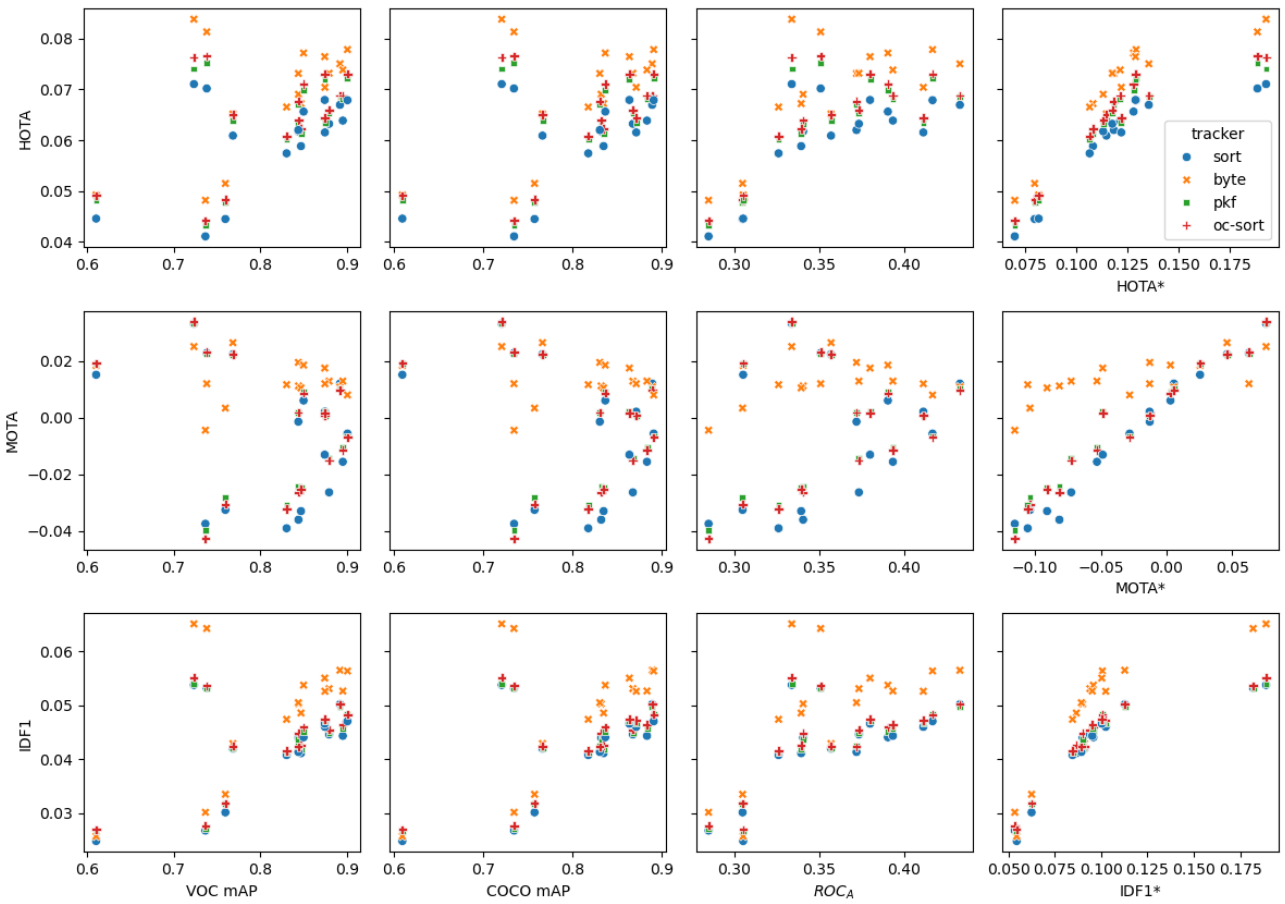

Fig. 5: Graph of correlation between tracker metric and proxies

Metric	HOTA			
Proxy	VOC mAP	COCO mAP	ROC_A	HOTA*
Marginal	0.23 [0.33]	0.23 [0.31]	0.44 [0.50]	**0.68 [0.86]**
sort	0.28 [0.32]	0.28 [0.31]	0.52 [0.50]	**0.74 [0.87]**
byte	0.28 [0.37]	0.28 [0.35]	0.47 [0.49]	**0.77 [0.82]**
pkf	0.25 [0.31]	0.25 [0.29]	0.50 [0.51]	**0.75 [0.88]**
oc-sort	0.22 [0.31]	0.22 [0.29]	0.47 [0.51]	**0.77 [0.88]**
Metric	MOTA			
Proxy	VOC mAP	COCO mAP	ROC_A	MOTA*
Marginal	0.06 [-0.10]	0.05 [-0.10]	0.07 [0.18]	**0.69 [0.85]**
sort	0.10 [-0.09]	0.09 [-0.07]	0.10 [0.21]	**0.98 [0.93]**
byte	0.01 [-0.15]	0.01 [-0.19]	0.05 [0.09]	**0.51 [0.54]**
pkf	0.08 [-0.09]	0.08 [-0.07]	0.10 [0.24]	**0.96 [0.96]**
oc-sort	0.08 [-0.09]	0.07 [-0.07]	0.10 [0.21]	**0.96 [0.96]**
Metric	IDF1			
Proxy	VOC mAP	COCO mAP	ROC_A	IDF1*
Marginal	0.24 [0.41]	0.24 [0.42]	0.39 [0.54]	**0.62 [0.88]**
sort	0.26 [0.40]	0.27 [0.41]	0.45 [0.54]	**0.70 [0.90]**
byte	0.27 [0.41]	0.27 [0.40]	0.41 [0.50]	**0.71 [0.82]**
pkf	0.27 [0.41]	0.28 [0.43]	0.47 [0.59]	**0.69 [0.91]**
oc-sort	0.26 [0.43]	0.27 [0.44]	0.45 [0.54]	**0.70 [0.90]**

TABLE III: This table summarizes the results of our evaluation in a robust context. The cells highlighted in bold show the best proxy for estimating the metric. Again, our approach significantly outperforms standard practice

without impacting the model evaluation results.

A conclusion that one might draw from this data is that the choice of tracker doesn't seem to effect which model is the best (or indeed impact the ranking much at all). However, the selection of trackers here is limited, and they are closely related algorithms. While this may not be conclusive, it is still suggestive, and warrants further investigation. The model choice also appears to make much more difference than the choice of tracker further highlighting the impact of traditional evaluation approaches selecting poor models.

Finally, this work highlights the importance of metrics that include the system task, and the effectiveness of using ground truth as a proxy for the system's software components. Our previous work [8], which considered similar properties for autonomous ground navigation, found similar results, though the approach there was more limited in terms of mocking software with ground truth. Our results provide further evidence to suggest that this approach generalizes to different ML system applications and tasks. Which is to say, we believe it likely that by using ground truth to simulate perfect behavior of the software components of an ML system, we can effectively evaluate the ML component in the system context, and get both more accurate estimates of performance and more accurate comparisons and selection

of models. We plan to explore our hypothesis in additional contexts in future work.

ACKNOWLEDGEMENT

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

REFERENCES

[1] C. L. Goues, S. Elbaum, D. Anthony, Z. B. Celik, M. Castillo-Effen, N. Correll, P. Jamshidi, M. Quigley, T. Tabor, and Q. Zhu, "Software engineering for robotics: Future research directions; report from the 2023 workshop on software engineering for robotics," *arXiv preprint arXiv:2401.12317*, 2024.

[2] C. Kastner, *Machine learning in production: from models to products*. MIT Press, 2025.

[3] I. Lazarevich, M. Grimaldi, R. Kumar, S. Mitra, S. Khan, and S. Sah, "Yolobench: benchmarking efficient object detectors on embedded systems," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 1169–1178.

[4] G. Jocher, J. Qiu, and C. Ayush, "Ultralytics yolo," 2023, accessed: Nov, 2024. [Online]. Available: https://github.com/ultralytics/ultralytics/tree/main

[5] 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: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer, 2014, pp. 740–755.

[6] 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*, vol. 129, pp. 548–578, 2021.

[7] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, pp. 303–338, 2010.

[8] Z. Pezzementi, T. Tabor, J. K. Chang, C. Tiernan, B. Drozd, E. M. Sample, C. Hazard, M. Wagner, and P. Koopman, "Perception robustness testing at different levels of generality." *Field Robotics*, vol. 1, no. 1, pp. 253–286, 2021.

[9] H. Gupta, O. Kotlyar, H. Andreasson, and A. J. Lilienthal, "Robust object detection in challenging weather conditions," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2024, pp. 7523–7532.

[10] J. Liu, Z. Wang, L. Ma, C. Fang, T. Bai, X. Zhang, J. Liu, and Z. Chen, "Benchmarking object detection robustness against real-world corruptions," *International Journal of Computer Vision*, pp. 1–19, 2024.

[11] B. Nushi, E. Kamar, E. Horvitz, and D. Kossmann, "On human intellect and machine failures: Troubleshooting integrative machine learning systems," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[12] X. Luo, T. Wei, S. Liu, Z. Wang, L. Mattei-Mendez, T. Loper, J. Neighbor, C. Hutchison, and C. Liu, "Certifying robustness of learning-based keypoint detection and pose estimation methods," 2024. [Online]. Available: https://arxiv.org/abs/2408.00117

[13] T. Tabor, F. Bowen, M. Kassel, E. Graupe, R. Saxena, C. S. Timperley, and C. Le Goues, "The surprising effectiveness of contrastive unpaired translation for test input generation," *SSRN 4879828*, 2024. [Online]. Available: https://papers.ssrn.com/sol3/Delivery.cfm?abstractid=4879828

[14] J. Cao, J. Pang, X. Weng, R. Khirodkar, and K. Kitani, "Observation-centric sort: Rethinking sort for robust multi-object tracking," 2023. [Online]. Available: https://arxiv.org/abs/2203.14360

[15] H. Cao, G. J. Pappas, and N. Atanasov, "Pkf: Probabilistic data association kalman filter for multi-object tracking," 2024. [Online]. Available: https://arxiv.org/abs/2411.06378

[16] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, Sep. 2016. [Online]. Available: http://dx.doi.org/10.1109/ICIP.2016.7533003

[17] 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," 2022. [Online]. Available: https://arxiv.org/abs/2110.06864

[18] B. Shuai, A. Bergamo, U. Buechler, A. Berneshawi, A. Boden, and J. Tighe, "Large scale real-world multi-person tracking," in *European Conference on Computer Vision*. Springer, 2022, pp. 504–521.