Finding Label Errors in Autonomous Vehicle Data With Learned Observation Assertions

Daniel Kang[†,*], **Nikos Arechiga**[†], **Sudeep Pillai**[†], **Peter Bailis**[*], **Matei Zaharia**[*]
[†]Toyota Research Institute, [*]Stanford University

Abstract

ML is being deployed in complex, real-world scenarios where errors have impactful consequences. As such, thorough testing of the ML pipelines is critical. A key component in ML deployment pipelines is the curation of labeled training data, which is assumed to be ground truth. However, in our experience in a large autonomous vehicle development center, we have found that labels can have errors, which can lead to downstream safety risks in trained models.

To address these issues, we propose a new abstraction, *learned observation assertions*, and implement it in a system, FIXY. FIXY leverages existing organizational resources, such as existing labeled datasets or trained ML models, to learn a probabilistic model for finding errors in labels. Given user-provided features and these existing resources, FIXY learns priors that specify likely and unlikely values (e.g., a speed of 30mph is likely but 300mph is unlikely). It then uses these priors to score labels for potential errors. We show FIXY can automatically rank potential errors in real datasets with up to $2\times$ higher precision compared to recent work on model assertions and standard techniques such as uncertainty sampling.

1 Introduction

ML is being deployed in complex applications with real-world consequences. For example, they are deployed in autonomous vehicles (AVs) [1], with potentially fatal consequences for errors, such as striking pedestrians [2]. Thus, quality assurance and testing of ML pipelines is vital [3, 4, 5].

A critical component of ML deployments is the curation of *high-quality* training data. Erroneous training data (e.g., Figure 1) can lead to subsequent safety repercussions for trained models. As such, finding these errors is critical to these pipelines.

Recent work has proposed Model Assertions (MAs) to indicate when errors may be occurring [6]. MAs are black-box functions over model inputs/outputs that return a severity score indicating when an ML model or human label may have an error. For example, an MA may assert that a prediction of a box of a car should not appear and disappear in subsequent frames of a video. MAs can be used to monitor the ML models in deployment, and to flag problematic data to label and retrain the model.

However, in our experience deploying MAs in a real-world AV perception pipeline, we have found several major challenges. First, users must manually specify MAs, which can be difficult for complex ML deployments. Second, calibrating severity scores to correctly indicate severity can be challenging. This is especially important as organizations have limited resources to evaluate potential errors. Third, ad-hoc specification of severity scores ignores organizational resources [7] that are already present or collected: large amounts of ground-truth labels and existing ML models.

To address these challenges, we propose a probabilistic domain-specific language (DSL), *Learned Observation Assertions* (LOA), for specifying assertions, and methods for data-driven specification of severity scores that leverage existing resources in ML deployments. We implement LOA in a prototype system (FIXY), embedded in Python to easily integrate with ML systems.

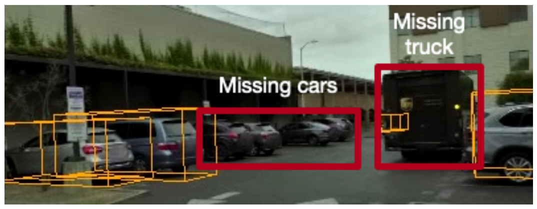
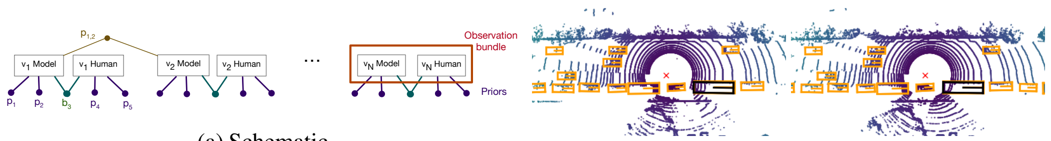


Figure 1: Example of human labels (orange) and missing labels (red) in the Lyft Perception dataset. The black truck highlighted is within 25m of the AV. Such errors can cause downstream issues with perception and planning systems.



(a) Schematic (b) $t = 0$ (c) $t = 1$

Figure 2: Example of the graphical model and corresponding LIDAR point cloud data. (a) is a schematic of a track with observations from LIDAR ML model predictions and from human-proposed labels. We show examples of observation priors (p_1, p_2, p_4, and p_5), bundle priors (b_3), and transition priors ($p_{1,2}$). The track is in black and other human-proposed labels are in orange for reference.

Our first contribution, LOA, allows users to specify properties of interest for perception tasks. LOA contains three components: data associations, priors over features, and application objective functions (AOFs). LOA can be used to specify assertions without ad-hoc code and ad-hoc severity scores by automatically transforming the specification into a probabilistic graphical model and scoring data components, producing statistically grounded severity scores.

In our labeling deployment, data across short snippets of time (*scenes*) are sent for labeling. These scenes are then audited for missing labels. These errors are difficult to find via ad-hoc MAs, so LOA supports associating observations together: across observation sources (*observation bundles*, i.e., predictions from different ML models or sensors) and across time (*tracks*, i.e., predictions of the same object as it moves). These associated observations can then be considered jointly for finding errors.

Our second contribution is methods of leveraging *organizational resources* [7], i.e., existing labels and ML models, to automatically specify severity scores via LOA. Users specify features over data, which are used to generate *priors*, and *application objective functions* to guide the search for errors. Priors take sets of observations and output a probability of seeing a feature of the input, e.g., the likelihood of encountering the volume of a 3D bounding box of a car. AOFs transform prior values for the application at hand. For example, if we wish to find unlikely tracks (e.g., a "ghost" track that an ML model erroneously predicts), the AOF could return one minus the prior's value. Importantly, our approach automatically learns priors, so users need not manually specify severity scores.

We evaluate FIXY on two real-world AV datasets annotated by leading commercial labeling vendors. Despite vendor best efforts [8], we find a number of labeling errors via FIXY in these datasets, some of which could cause safety violations (e.g., Figure 1).

2 FIXY

Example and background. As an example, we describe the ML deployment pipeline for our AVs, focusing on labeling data for perception. Other organizations deploy similar pipelines, e.g., as documented by [1]. Our AV deployment pipeline is continuous, in which ML models are trained, tested, and deployed on vehicles. Because models are continuously exposed to new and different scenarios, we continuously collect and label data, which is subsequently used to develop and retrain ML models [9].

Label quality is of paramount concern: erroneous labels can lead to downstream safety violations. Vendors that provide labels are not always accurate, which is in contrast to the large body of work that assumes datasets are "gold." The most egregious errors are when objects are entirely missed.

To address label quality issues, our organization has expert auditors who audit the vendor-provided labels. Unfortunately, it is too expensive to audit every data point, so we have developed FIXY, which ranks datapoints that are likely to be erroneous and allows better utilization of auditing resources.

Overview. FIXY aims to find errors in human-proposed labels or ML model outputs ("observations"). As input, FIXY takes user-defined features over observations, which return scalar values, and AOFs. FIXY aims to rank observations or sets of observations as likely errors.

FIXY consists of: a DSL for specifying relations between observations and priors, a component to learn priors, a scoring component, and a runtime engine. FIXY's DSL allows users to specify how priors and observations interact. Its prior learning component fits distributions over existing observations. Its scoring component scores observations or groups of observations by likelihood. Finally, its runtime engine ranks observations or groups of observations.

2.1 Learned Observation Assertions

The LOA DSL provides a means of specifying associations between observations and priors. We show an example of a compiled LOA graph and corresponding sensor data observations in Figure 2.

Overview. LOA contains elements to specify observation/observation and observation/prior interactions. Since perception data often contains spatial and temporal components, LOA has *observation bundles* within a single time step and *tracks* across time (collectively referred to as OBTs). LOA then allows priors to be specified over any OBT. The user can specify AOFs over any prior.

Example. Consider finding errors in human labels of 3D bounding boxes. We have two sources of observations: ML model predictions and human labels. We bundle observations from the ML model and human that overlap in a single time step, and form tracks for bundles that overlap across time (Figure 2). Given these tracks, we use a track prior and AOF to set the score of any track that contains a human label to negative infinity to exclude these tracks. We then use existing labels to automatically generate priors, e.g., the likelihood of seeing a particular box velocity. FIXY then compiles the specification to a graphical model (e.g., Figure 2a). Then, an auditor checks the ranked tracks for errors.

Generating graphical models. FIXY will compile the scene, priors, and AOFs to a graphical model, which is used to score potential errors. FIXY will create nodes for each observation and prior and edges between priors/observations. If a prior applies to a single observation, FIXY will create a single edge. If a prior applies to a group of observations (e.g., an observation bundle or track), FIXY will create one edge between each observation in the group and the prior. Once the graphical model is constructed, FIXY can then score any OBT by the negative log-likelihood implied by priors. The score of a group of observations is the sum of the scores of the observations, normalized by the number of priors.

2.2 Priors

Overview. A key component to scoring OBTs are the priors. Both our AV deployment and other organizations deploying ML collect large amounts of training data. This training data contains labels (potentially with errors), which can be used to fit empirical distributions to the priors. We leverage these existing labels in this work, as they come at no additional cost.

To fit these priors, FIXY takes as input scalar or vector valued features over OBTs. For example, a feature over an observation may take a bounding box and return the volume of the box. The user may also manually specify priors to rank severity (e.g., distance of an object to the AV) or to filter certain instances (e.g., only search for errors in detecting pedestrians). Finally, FIXY takes an optional AOF, which can be applied per prior or over the resulting score.

Prior Types. FIXY contains priors over OBTs and transitions. Specifically, FIXY contains priors over 1) single observations (e.g., box volume), 2) observation bundles (e.g., consistency of labels across sensors), 3) observations or bundles in adjacent time steps within a track (e.g., box velocity), and 4) tracks (e.g., normalization across tracks).

Application Objective Functions. AOFs wrap priors to transform them in application-specific ways. They take scalar values and return scalar values. The most common operations are taking the inverse and setting the probability to 0/1. For example, when searching for likely tracks, the AOF may be the identity, but when searching for unlikely tracks, the AOF may invert the probability.

Method	Dataset	Precision at 10
FIXY	Lyft	**69%**
Ad-hoc MA (rand)	Lyft	32%
Ad-hoc MA (conf)	Lyft	39%
FIXY	Internal	**76%**
Ad-hoc MA (rand)	Internal	30%
Ad-hoc MA (conf)	Internal	60%

Table 1: Precision at top 10 of FIXY and ad-hoc MA baselines for finding tracks missed by humans. FIXY outperforms baselines by up to $2\times$.

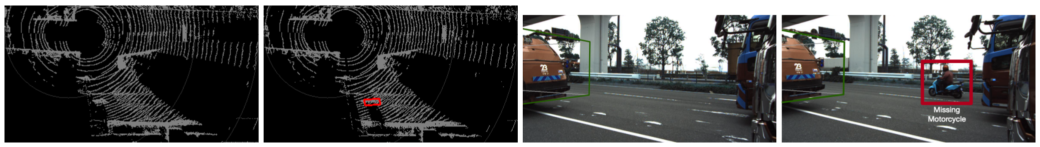


Figure 3: Example of a motorcycle (red) missed by human proposals. We show the LIDAR view (top) and the camera view (bottom). The motorcycle is occluded, so only appears for <1 second. Nonetheless, it is important to predict.

3 Evaluation

We investigated whether FIXY could find errors in vendor-provided human labels. We searched for tracks that were entirely missed by human proposals, as these errors are the most egregious.

Experimental setup. We evaluated FIXY on two AV perception datasets: an internal dataset and the Lyft Level 5 perception dataset [10], which has been used to develop models [11]. Both datasets consists of LIDAR and camera data that were densely labeled with 3D bounding boxes by leading external vendors for human labels. We used three sources of observations: human-proposed labels, LIDAR ML model labels [12], and expert auditor labels. We use priors automatically learned from data (volume, velocity, count) and priors for selecting more egregious errors (distance, model only).

We compared against manually designed MAs developed to find errors in a similar setting [6]. Both datasets were vetted for errors by leading vendors. Thus, we find errors that were not found in an external audit. We manually checked the top 10 potential errors ranked by FIXY and ad-hoc MAs. We measured precision, where a higher precision indicates that there are more errors. For the Lyft dataset, we measured the precision across every scene in the validation set with errors. For our internal dataset, we focused on a scene that failed audit.

Results. FIXY outperforms on finding errors in both datasets (Table 1) by aggregating information across observations in tracks, which is difficult to do with ad-hoc MAs. We show examples of errors FIXY found in Figures 1 and 3. Many of these errors are close to the AV and are clearly visible, which are particularly problematic.

To contextualize our results, FIXY uncovered an error missed by an internal audit (Figure 3). Given the short time period the motorcycle was visible, it can be difficult to find for both crowd workers and auditors. Nonetheless, it is critical to be accurately labeled. Furthermore, the open-sourced Lyft perception dataset has many vehicles that were not labeled. We found a number of inconsistencies (e.g., parked cars that were and were not labeled).

4 Related Work

Data cleaning. Work in the data systems community has focused on cleaning tabular data [13, 14]. This work focuses largely on detecting errors via constraints [15, 16, 17] and more recently machine learning [18, 19, 20]. Unfortunately, these techniques do not directly apply to the labels in many ML pipelines, thus necessitating the need for new abstractions and systems for ML data.

ML testing. A recent survey [3] shows that ML testing focuses on pipelines where schemas have meaningful information (categorical or numeric data) [21, 22, 23]. While important, they do not apply to the settings we consider. Other work considers statistical measures of accuracy [4], numeric fuzzing [24], worst case perturbations [5], and other techniques [25]. These approaches are complementary to FIXY. In this work, we focus on finding errors in complex perception training data and model errors. To our knowledge, MA are closest line of work [6]. Unfortunately, users must manually specify MAs and severity scores, which can be challenging and miss important classes of errors.

References

[1] Andrej Kaparthy. Building the software 2.0 stack. 2018.

[2] Daisuke Wakabayashi. Self-driving uber car kills pedestrian in arizona, where robots roam. `https://www.nytimes.com/2018/03/19/technology/uber-driverless-fatality.html`, 2018.

[3] Jie M Zhang, Mark Harman, Lei Ma, and Yang Liu. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*, 2020.

[4] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 291–300. IEEE, 2019.

[5] Weiming Xiang, Patrick Musau, Ayana A Wild, Diego Manzanas Lopez, Nathaniel Hamilton, Xiaodong Yang, Joel Rosenfeld, and Taylor T Johnson. Verification for machine learning, autonomy, and neural networks survey. *arXiv preprint arXiv:1810.01989*, 2018.

[6] Daniel Kang, Deepti Raghavan, Peter Bailis, and Matei Zaharia. Model assertions for monitoring and improving ml model. *MLSys*, 2020.

[7] Sahaana Suri, Raghuveer Chanda, Neslihan Bulut, Pradyumna Narayana, Yemao Zeng, Peter Bailis, Sugato Basu, Girija Narlikar, Christopher Ré, and Abishek Sethi. Leveraging organizational resources to adapt models to new data modalities. *arXiv preprint arXiv:2008.09983*, 2020.

[8] Chiao-Lun Cheng. Training data - quantity is no panacea. 2019.

[9] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, et al. Tfx: A tensorflow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1387–1395, 2017.

[10] R. Kesten, M. Usman, J. Houston, T. Pandya, K. Nadhamuni, A. Ferreira, M. Yuan, B. Low, A. Jain, P. Ondruska, S. Omari, S. Shah, A. Kulkarni, A. Kazakova, C. Tao, L. Platinsky, W. Jiang, and V. Shet. Lyft level 5 perception dataset 2020. `https://level5.lyft.com/dataset/`, 2019.

[11] Benjin Zhu, Zhengkai Jiang, Xiangxin Zhou, Zeming Li, and Gang Yu. Class-balanced grouping and sampling for point cloud 3d object detection. *arXiv preprint arXiv:1908.09492*, 2019.

[12] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12697–12705, 2019.

[13] Xu Chu, Ihab F Ilyas, Sanjay Krishnan, and Jiannan Wang. Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 international conference on management of data*, pages 2201–2206, 2016.

[14] Erhard Rahm and Hong Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[15] Leopoldo Bertossi. Consistent query answering in databases. *ACM Sigmod Record*, 35(2):68–76, 2006.

[16] George Beskales, Ihab F Ilyas, and Lukasz Golab. Sampling the repairs of functional dependency violations under hard constraints. *Proceedings of the VLDB Endowment*, 3(1-2):197–207, 2010.

[17] Philip Bohannon, Wenfei Fan, Michael Flaster, and Rajeev Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 143–154, 2005.

[18] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J Franklin, and Ken Goldberg. Activeclean: Interactive data cleaning while learning convex loss models. *arXiv preprint arXiv:1601.03797*, 2016.

[19] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. Holoclean: Holistic data repairs with probabilistic inference. *arXiv preprint arXiv:1702.00820*, 2017.

[20] Alireza Heidari, Joshua McGrath, Ihab F Ilyas, and Theodoros Rekatsinas. Holodetect: Few-shot learning for error detection. In *Proceedings of the 2019 International Conference on Management of Data*, pages 829–846, 2019.

[21] Nick Hynes, D Sculley, and Michael Terry. The data linter: Lightweight, automated sanity checking for ml data sets. In *NIPS MLSys Workshop*, 2017.

[22] Neoklis Polyzotis, Martin Zinkevich, Sudip Roy, Eric Breck, and Steven Whang. Data validation for machine learning. *MLSys*, 2019.

[23] Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. Data management challenges in production machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1723–1726, 2017.

[24] Augustus Odena, Catherine Olsson, David Andersen, and Ian Goodfellow. Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. In *International Conference on Machine Learning*, pages 4901–4911, 2019.

[25] Hung Viet Pham, Thibaud Lutellier, Weizhen Qi, and Lin Tan. Cradle: cross-backend validation to detect and localize bugs in deep learning libraries. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1027–1038. IEEE, 2019.