Finding Label and Model Errors in Perception 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× higher precision compared to recent work on model assertions and standard techniques such as uncertainty sampling.

AIDB Workshop Reference Format:
Daniel Kang, Nikos Arechiga, Sudeep Pillai, Peter Bailis, Matei Zaharia. Finding Label and Model Errors in Perception Data With Learned Observation Assertions. *AIDB* 2021.

1 Introduction

Machine learning (ML) is increasingly being deployed in complex applications with real-world consequences. For example, ML models are being deployed to make predictions over perception data in autonomous vehicles (AVs) [11], with potentially fatal consequences for errors, such as striking pedestrians [24]. Thus, quality assurance and testing of ML pipelines is of paramount concern [1, 15, 25, 26].

A critical component of ML deployments is the curation of *high-quality* training data, in which crowd workers produce labels. Similar to how errors in tabular data results in downstream errors in query results, 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.

Unfortunately, standard data cleaning techniques are not well suited for finding errors in ML pipelines. For example, while constraints work well on tabular data, they are less suited for perception data, e.g., pixels of an image. As such, it is necessary to develop new tools for finding errors in these pipelines.

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.

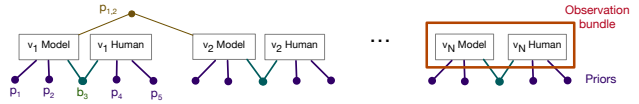
Recent work has proposed Model Assertions (MAs) that indicate when errors may be occurring [10]. MAs are black-box functions over model inputs/outputs that return a quantitative severity score indicating when an ML model or human-proposed label may have an error. For example, an MA may assert that a 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 [23] that are already present or collected: large amounts of ground-truth labels and existing ML models.

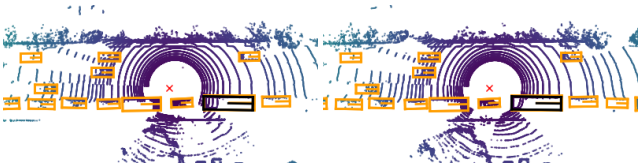
To address these challenges, we propose a probabilistic domain-specific language (DSL), *Learned Observation Assertions* (LOA), for specifying assertions, and methods for data-driven specification of severity scores that leverage existing resources in ML deployments. We implement LOA in a prototype system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

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, sensor data across short snippets of time (*scenes*) are sent to vendors 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:



(a) Schematic of a track that contains 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 ($p_{1,2}$).



(b) $t = 0$ **(c)** $t = 1$

Figure 2: Example of the graphical model and corresponding LIDAR point cloud data. The track is in black and other human-proposed labels are in orange for reference.

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 when searching for errors.

Our second contribution is methods of leveraging *organizational resources* [23], 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 [6], we find a number of labeling errors via Fixy in these datasets, some of which could cause safety violations (e.g., Figure 1). We further show that Fixy can outperform hand-crafted MAs and uncertainty sampling for finding errors in ML models.

2 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 Kaparthy [11].

Our AV deployment pipeline is a continuous process, in which ML models are trained, tested, and deployed on vehicles. Because ML 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 [2].

Label quality is of paramount concern: erroneous labels can lead to downstream errors and 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 in labeling.

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 enables ranking datapoints that are likely to be erroneous and allows better utilization of auditing resources.

3 System Overview

Fixy aims to find errors in human-proposed labels or ML model outputs (we collectively refer to these as "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.

4 Learned Observation Assertions

The LOA DSL provides a simple 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 for allowing users to specify how observations interact with each other and how priors interact with observations. Our implementation of LOA is embedded in Python for ease of integration with ML packages. Since perception data often contains spatial and temporal components, we allow users to construct *observation bundles* within a single time step and *tracks* across time (collectively referred to as OBTs). Given these data elements, LOA then allows priors to be specified over any OBT. Finally, the user can specify AOFs over any prior.

Example. Consider the use case of finding errors in human labels of 3D bounding boxes over LIDAR data. We have two sources of observations: ML model predictions and labels from a human. We can then bundle observations from the ML model and human that overlap in a single time step, and form tracks for bundles that overlap across time. We show an example of a track in Figure 2.

Given these tracks, we can use a track prior and AOF to set the score of any track that contains a human observation to negative infinity to exclude tracks that already have human labels. We can then use existing labels to automatically generate priors, e.g., such as the likelihood of seeing a particular box volume or velocity. Fixy will then compile the specification to a graphical model (e.g., Figure 2a). Then, an auditor can check the ranked tracks for errors.

Code examples. We first show an observation prior on 3D box volume, which specifies the likelihood of seeing a box volume.

```
class VolumePrior(ObsPrior):
  def score(self, box):
    vol = obs.width * obs.height * obs.length
    # Wrapper to a density estimator. It returns
    # a probability of observing the volume.
    return self.KDEPrior(vol)
```

We also show an example of associating observations across time to form tracks, which associates two boxes in a track if their intersection over union (IOU) is at least 0.5.

```
class TrackBundler(Bundler):
  def is_associated(self, box1, box2):
    return compute_iou(box1, box2) > 0.5
```

Generating graphical models. Fixy will compile the scene, priors, and AOFs to a graphical model, which is used to score groups of observations. Fixy uses these scores to flag potential errors.

To compile a scene, Fixy will create nodes for each observation and prior. Then, Fixy will create edges between each prior and the observation it applies over. 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.

5 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. While transition priors can be implemented as track priors, we provide a syntatic element for ease of use. 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).

Learning priors. Given the features, Fixy automatically fits priors over existing training datasets. To fit priors, Fixy takes a function that accepts a list of scalars/vectors and returns a fitted distribution. By default, Fixy uses a kernel density estimator (KDE). In some cases, other types of distributions are appropriate (e.g., discrete distributions): the user can override our default in these cases.

To learn priors given a set of scenes, Fixy first exhaustively generates the features over the data and collects the scalar or vector values. Then, for each prior, Fixy executes the fitting function over the scalar/vector values.

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.

Name	Type	Description
Volume	Obs.	Class-conditional box volume
Distance	Obs.	Distance to AV
Model only	Bundle	Selects bundles with model predictions only
Velocity	Trans.	Class-conditional object velocity
Count	Track	Filters tracks with two or fewer obs.

Table 1: Description of priors we used in this evaluation.

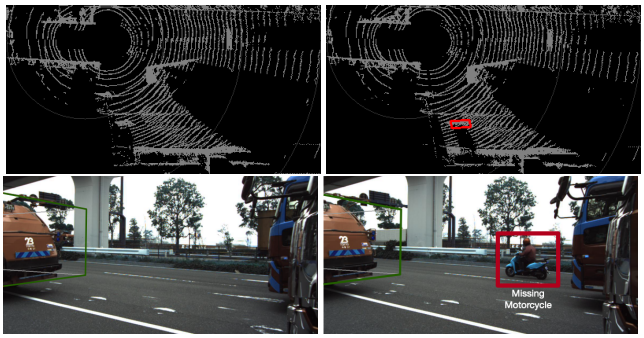


Figure 3: Example of a motorcycle (red) missed by human proposals. We show the LIDAR view (top) and the camera view (bottom). The motorcycle is occluded, so only appears for <1 second. Nonetheless, it is important to predict.

6 Evaluation and Applications

6.1 Experimental Setup

Datasets. We evaluated Fixy on two AV perception datasets: an internal dataset from our organization and the Lyft Level 5 perception dataset [12], which has been used to develop models [27] and host competitions. Both datasets consists of LIDAR and camera data that were densely labeled with 3D bounding boxes by leading external vendors for human labels ("human-proposed labels").

Observations. We used three sources of observations: human-proposed labels, LIDAR ML model labels [14], and expert auditor labels. All sources predict 3D bounding boxes. We focus on the common classes of car, truck, pedestrian, and motorcycle.

Baselines. We compared against manually designed, ad-hoc MAs developed by Kang et al. [10] and uncertainty sampling. The ad-hoc MAs were designed to find errors in similar settings to ours. Uncertainty sampling is commonly used in active learning [22]. Furthermore, both datasets were vetted for errors by leading vendors. Thus, we find errors that were not found in an external audit.

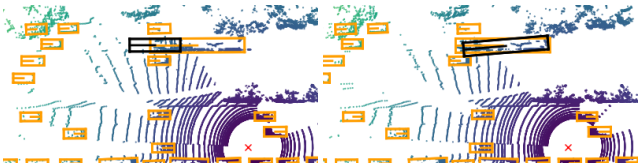
Priors. We use priors automatically learned from data (volume, velocity, count) in addition to priors for selecting more egregious errors (distance, model only), as shown in Table 1.

6.2 Fixy can Find Missing Tracks

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.

Method	Dataset	Precision at top 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 2: Precision at top 10 of Fixy and ad-hoc MA baselines for finding tracks missed by humans. Fixy used priors and ad-hoc MAs used the consistency assertion described by Kang et al. [10]. Fixy outperforms baselines by up to 2×.



(a) $t = 0$ **(b)** $t = 1$

Figure 4: Example of a model error in the Lyft dataset not found by ad-hoc MAs. The prediction overlaps across frames, but does is not consistent. Fixy can find such errors as they produce unlikely values under learned priors.

Experimental setup. We deployed the priors described above. The AOF zeros out tracks containing any human proposal. The remaining tracks contain only model predictions. For the ad-hoc MAs, we used the "consistency" assertion [10]. We ordered the ML model predictions randomly and by model confidence.

We manually checked up to 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 within the top 10 proposals. For the Lyft dataset, we measured the precision across every scene in the validation set with errors. For our internal dataset, we focused on the scene that failed audit.

Results. Fixy outperforms on finding errors in both datasets (Table 2) 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).

6.3 Fixy can Find Novel ML Prediction Errors

We investigated whether or not Fixy can find errors in model predictions, e.g., in Figure 4, without using human-proposed labels.

Experimental setup. Unlike for finding errors in human-proposed labels, ad-hoc MAs can achieve high precision for errors in ML model predictions. As such, we deployed three ad-hoc MAs as used in Kang et al. [10] (`appear`, `flicker`, and `multibox`). In addition to ad-hoc MAs, we compared to uncertainty sampling, in which we sampled predictions around a confidence threshold.

We then deployed Fixy to find errors in ML model prediction after *excluding the errors found by these ad-hoc MAs*. We searched for both localization and classification errors. For Fixy, we deployed the same priors as above with the exception of distance and model only. We additionally deployed a track prior over the total number of observations. We measure the precision of the top 10 potential errors over 5 scenes in the Lyft dataset.

Results. Fixy achieves a precision at 10 of 82% *while excluding errors found by ad-hoc MAs* and uncertainty sampling achieved a precision of 42%. Many of these errors have tracks without missing time stamps and are longer than two observations, so will not trigger the ad-hoc MAs. For example, the predictions overlap across frames but in an unlikely way in Figure 4.

Furthermore, in contrast to uncertainty sampling, Fixy uncovers errors with high model confidence. Fixy discovered errors with confidences as high as 95%, which uncertainty sampling missed.

7 Related Work

Data cleaning. Work in the data systems community has focused on cleaning tabular data [7, 19]. This work focuses largely on detecting errors via constraints [3–5] and more recently machine learning [8, 13, 21]. 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.

Enriching data and weak supervision. Other work aims to clean or enrich data, often to train ML models. For example, HoloClean automatically aggregates noisy cleaning rules in a statistical fashion [8, 21]. Other systems allow users to specify labeling functions [20] or use organizational resources [23], e.g., embeddings, to train models. Mission-critical models must be validated with a high-quality dataset, so require labels [20]. We leverage different organizational resources to find errors in labels, as opposed to training models with fewer labels.

ML testing. A recent survey [26] shows that existing work in ML testing focuses on pipelines where schemas have meaningful information (categorical or numeric data) [9, 17, 18]. While important, they do not apply to the settings we consider. Other work considers statistical measures of accuracy [1], fuzzing for numeric errors [15], worst case perturbations [25], and other techniques [16]. 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 [10]. Unfortunately, users must manually specify MAs and severity scores, which can be challenging and miss important classes of errors.

8 Conclusion

To address the problem of finding errors, we propose LOA and implement Fixy. LOA allows users to specify data-driven priors to indicate which data points are potentially erroneous. Fixy, leverages existing organizational resources (trained ML models and labeled data) to find errors. We show that Fixy can find errors in human labels up to 2× more effectively than prior research work.

References

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

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

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

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

[5] Philip Bohannon, Wenfei Fan, Michael Flaster, and Rajeev Rastogi. 2005. 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*. 143–154.

[6] Chiao-Lun Cheng. 2019. Training Data - Quantity is no Panacea. (2019). https://scale.com/blog/training-data-quantity-is-no-panacea

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

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

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

[10] Daniel Kang, Deepti Raghavan, Peter Bailis, and Matei Zaharia. 2020. Model Assertions for Monitoring and Improving ML Model. *MLSys* (2020).

[11] Andrej Kaparthy. 2018. Building the Software 2.0 Stack. (2018).

[12] 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. 2019. Lyft Level 5 Perception Dataset 2020. https://level5.lyft.com/dataset/.

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

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

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

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

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

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

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

[20] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2020. Snorkel: rapid training data creation with weak supervision. *The VLDB Journal* 29, 2 (2020), 709–730.

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

[22] Burr Settles. 2009. *Active learning literature survey*. Technical Report. University of Wisconsin-Madison Department of Computer Sciences.

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

[24] Daisuke Wakabayashi. 2018. Self-Driving Uber Car Kills Pedestrian in Arizona, Where Robots Roam. https://www.nytimes.com/2018/03/19/technology/uber-driverless-fatality.html.

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

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

[27] Benjin Zhu, Zhengkai Jiang, Xiangxin Zhou, Zeming Li, and Gang Yu. 2019. Class-balanced Grouping and Sampling for Point Cloud 3D Object Detection. *arXiv preprint arXiv:1908.09492* (2019).