Data Management for ML-based Analytics and Beyond

DANIEL KANG, Stanford University, USA
JOHN GUIBAS, Stanford University, USA
PETER BAILIS, Stanford University, USA
TATSUNORI HASHIMOTO, Stanford University, USA
YI SUN, University of Chicago, USA
MATEI ZAHARIA, Stanford University, USA

The increasing capabilities of machine learning (ML) has enabled the deployment of ML methods in a variety of applications, ranging from unstructured data analytics to autonomous vehicles. Due to the volumes of data ML is deployed over, it is infeasible for humans to monitor deployments: the Tesla fleet of vehicles produces exabytes of data and millions of hours of video per day. As a result, ML deployments can fail in unexpected and catastrophic ways.

In this work, we highlight three important, but underlooked aspects of ML deployment pipelines: 1) managing *high-quality* training data, 2) monitoring ML errors at deployment time, and 3) connecting end use to deployment algorithms. We first demonstrate that training labels are often erroneous, contrary to standard practice, *even when labeled by leading vendors*. We then demonstrate that standard methods of deploying ML methods can lead to downstream errors. As a first step towards addressing these issues, we review and contextualize two abstractions for finding errors in training data and deployments. We further describe how to improve algorithms for analytics queries as a case study for optimizing ML pipelines end-to-end.

1 INTRODUCTION

The applications of machine learning (ML) have only expanded as more capable models and algorithms have been created. As ML improves in capabilities, it is deployed on more volumes of data and for more tasks. For example, ML is being used in applications ranging from autonomous vehicles, detecting hummingbirds in videos to study microcosms [26], and find novel earthquakes [45]. These applications produce large volumes of data: a single autonomous vehicle can produce over 4 TB of data in a day and ~200 days worth of ecological video is ~9 TB.

Due to the nature of ML-based deployments, it is infeasible for human to exhaustively monitor this data (limited "human attention") [5]: the Tesla fleet of vehicles sees more data per day than the largest labeled image dataset.[1] As a result, ML is increasingly being deployed where only a small fraction of the data could possibly be monitored.

As a result, ML-based deployments have already been involved in negative outcomes, including severe accidents such as autonomous vehicles striking pedestrians [22]. Beyond autonomous vehicles, ML models have also been involved in confidently overestimating flu case counts [35], have failed to generalize in network congestion control algorithms [52], have failed to generalize in predicting unemployment claims from Twitter data [3], etc. These errors are not limited to resource-constrained organizations: Google, Uber, and other organizations have been involved with such errors [22, 35].

To address these issues, the ML and systems community has largely focused on three areas: increasing model capacity [17, 18, 23], improved training methods [13, 14, 50], and validation methods (e.g., statistical validation or "verification") [40, 51]. For example, on the model front, recent work on transformers have not only revolutionized natural language processing, but also

[1]Tesla has produced over 1.9M vehicles [16]. Assuming each vehicle drives even an hour a day, the fleet will produce 6.8B images at one frame per second, both of which are gross underestimates. The largest labeled image dataset is 300M images [49].

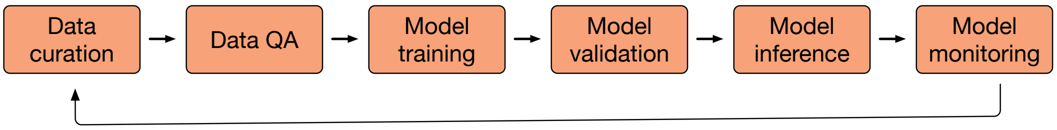
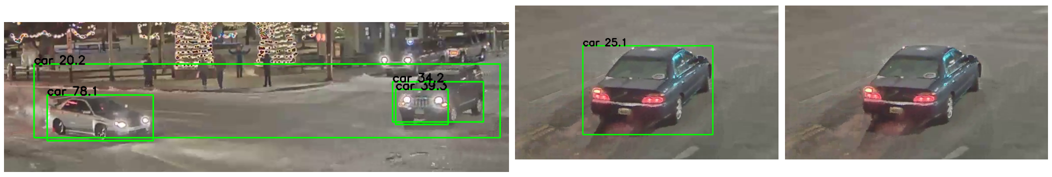


Fig. 1. Steps present in most ML deployments. Most of the literature focuses on model training, validation, and inference. Furthermore, most of the data assumes the pipeline is fixed, as opposed to ongoing. We argue for the importance of the underlooked components of the ML deployment pipeline: data curation, data quality assurance, and model monitoring.



(a) Unrealistic predicted boxes overlapping. (b) Correctly predicted frame ($t = 0$). (c) Incorrectly predicted frame immediately adjacent to the correctly predicted frame ($t = 1$).

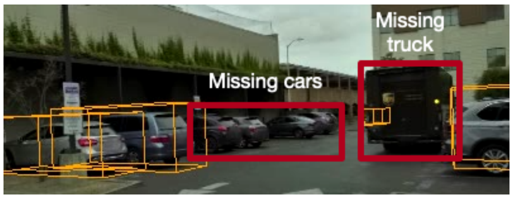
Fig. 2. Examples of errors from SSD deployed on a street camera video. These errors are easily specified by domain experts. Taken from Kang et al. [32].

have shown great progress on vision tasks with potential for improved semantic understanding of images [17, 18, 23]. On the training front, work on self- and semi-supervised learning have the potential to allow understanding of large datasets with few labels [13, 14, 50]. On the validation front, work aims to catch overflow errors at training time [40], "verify" that neural networks satisfy certain properties [51], etc.

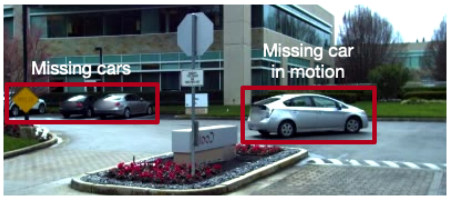
Unfortunately, the ML deployment stack has steps beyond training ML models that have been understudied: monitoring deployed ML models, the curation of *high-quality* training data, and directly leveraging end use cases to ML models. We show a typical ML model deployment stack in Figure 1. Much of existing work focuses on the training, validation, and inference stages, with little work on the remaining stages.

In this work, we argue for the importance of the ML deployment stack beyond training, validation, and inference. In particular, we argue for the importance of monitoring, vetting/curating training data, and end-to-end optimization of ML deployments. We describe our experience in ML deployments and highlight areas that have been neglected in existing work. We then review and extend work to address these areas.

Monitoring ML deployments. We first show that common methods of deploying ML methods can lead to large downstream errors, e.g., in aggregation query results. As an example, a common method of performing analytics over unstructured data is to deploy a pre-trained model on a new dataset. Consider deploying SSD [37], a single-shot object detector, pre-trained on MS-COCO [36] for traffic analysis. We deployed SSD on a commonly-studied traffic dataset denoted `night-street` [25, 27, 29]. As shown in Figure 2, SSD makes a numerous errors, despite the camera being in a fixed position. The literature has many more examples of such discrepancies: simple transformations of images can cause errors [4]. As a result, we argue it is critical to monitor ML models at deployment time.



(a) Example of missing vehicles within 25m of the autonomous vehicle.



(b) Example of missing vehicles in motion close to the autonomous vehicle.

Fig. 3. Examples of errors from the publicly available Lyft Level 5 dataset. This dataset was produced by a leading labeling vendor. Taken from Kang et al. [24].

To begin to monitor ML deployments, we review and extend model assertions [32]. Model assertions are opaque functions that allow domain experts to specify when errors may be occurring. For example, a traffic analyst can specify that cars should not appear and disappear too rapidly in video. We review how model assertions can be used and extend its results to other domains, including for tabular data.

Curation and vetting of high-quality training data. We then show that the common practice of assuming human labels are correct is in fact flawed. A number of publicly available datasets contain egregious errors, including datasets used to train autonomous vehicles. We show several examples of errors from the Lyft Level 5 perception dataset [33], which has been used to host competitions [48] and develop models [53], in Figure 3. While seemingly simple, similar errors were involved in Uber's fatal autonomous vehicle incident: "[a]s the [automated driving system] changed the classification of the pedestrian several times–alternating between vehicle, bicycle, and an other–the system was unable to correctly predict the path of the detected object" [22], highlighting the need to avoid such errors. Furthermore, the utility of a given additional training data item varies greatly depending on the applications [26, 32].

To address the quality and utility of training data, we first describe learned observation assertions (LOA), which aids users in determining when training data may contain errors. LOA leverages existing training data and existing ML models to highlight such errors. We further describe how model assertions and LOA can be used to select training data more efficiently than standard methods of selecting training data.

Guaranteeing performance. Finally, we show a range of methods for analytics deployments can catastrophically fail, ranging from selection to aggregation queries. For example, when selecting rare events with a recall target of 90%, common methods of selection can return sets with recalls below 20%. Furthermore, methods for accelerating aggregation queries, when naively applied, can return higher errors while simultaneously being slower than alternative methods. These errors are particularly problematic for scientific analyses or for decision making, as these methods also do not specify their errors.

To address the issues that arise in analytics, we describe and contextualize methods for achieving statistical guarantees on query accuracy. These methods largely leverage sampling. However, in contrast to standard AQP, the fields to be aggregated on are not present ahead of time, so standard indexing/preprocessing methods cannot be used. We describe challenges that arise in this setting and methods to address this issue.

Component interface	Description
`Train(ModelArchitecture, TrainingData, TrainingAlgorithm) -> TrainedModel`	Model trainin
`Validation(TrainedModel, ValidationData, ValidationAlgorithm) -> Statistics`	Model valida
`Inference(TrainedModel, UnseenData) -> StructuredOutputs`	Executing a t
`EndUse(StructuredOutputs, UnseenData) -> Action | Analysis`	Use in the en
`Curation(TrainedModel, StructuredOutputs, UnseenData) -> LabeledData`	Curation of t

Table 1. Example interfaces for various aspects of the ML deployment as described in Figure 1. As shown, many ML models produce *structured outputs*, which are subsequently consumed for downstream tasks. Additionally, many components have well defined interfaces, which allow teams of engineers, data scientists, and domain experts to work together.

In the remainder of the paper, we describe methods for monitoring ML deployments at inference time, curation and quality control of ML data, and an example of optimizing end-to-end ML deployments for analytics. Throughout, we specify when results are novel to this work.

2 BACKGROUND

In this section, we outline the ML deployment stack. Our description is generated from a range of industry collaborations and conversations, our own experiences in deploying ML models, and the literature. Our collaborators include large autonomous vehicle organizations, ML model providers partnering with numerous Fortune 500 companies, scientists, and others. We additionally highlight differences between our observations and standard academic practice.

2.1 Overview

ML deployments vary widely in end use case and resources. Their use cases range from analyzing video to study hummingbird feeding patterns, finding earthquakes, and driving vehicles autonomously. Organizations that deploy ML range from single individuals with no coding experience to hundreds of engineers with years of experience. As a result, these deployments vary widely. Nonetheless, we aim to outline common properties of these deployments.

All ML deployments we are aware of contain a subset of the following: training models, validating models, deployment/inference in its end use case, and the curation of training data. We describe each step in detail below.

As different individuals and teams have specific concerns, we outline interfaces of different parts of the ML deployment stack. At large organizations, each part has one or more teams dedicated to optimizing that part of the stack. Even for individuals or smaller teams, different parts are often out-sourced (e.g., AutoML for training).

We describe the interfaces in Table 1. Much of existing work has focused on `Train`, `Validation`, and `Inference`, with less work on the remaining parts. In many ML deployments, a large number of models are trained and deployed. We describe the interfaces for a single model, although models are sometimes jointly trained.

An important feature we highlight is that ML models are often used to take *unstructured data* (e.g., text, video, images, audio) to *structured outputs*. Unstructured data is often difficult and computationally intensive to process. In contrast, structured outputs are much simpler to process. We note that a number of deployments do not fall under this setting (e.g., generation).

2.2 Training

The first step of ML model deployment is training a model. In this step, training algorithm takes a model architecture and training data as input and outputs a trained model. The most common training algorithm is stochastic gradient descent (SGD). We highlight popular model architectures for common tasks/modalities:

(1) BERT/GPT-based architectures are most commonly used for tasks over text (e.g., sentiment analysis, topic classification, etc.).
(2) Convolution-based architectures (e.g., ResNet) are most commonly used for image classification.
(3) RCNN (e.g., Mask R-CNN, Faster R-CNN) or single-shot (e.g., SSD, YOLO) backbones with convolutional backbones are most commonly used for object detection.

A large body of existing work focuses on improving training algorithms and model architectures.

2.3 Validation

Given a trained ML model, the next step is to validate the model. In this step, data (ideally) unseen at training time and reflective of deployment is used to produce statistics about the model. The most common method is to produce an average validation statistic over the dataset, such as the mean accuracy for classification or mean average precision for object detection. New models are typically not deployed unless a majority of validation metrics have improved.

An emerging body of work is demonstrating the dangers of using a fixed validation set. Even leading experts in ML funded by industry titans can be prone to such errors. For example, Google flu trends worked well on in-distribution data, but catastrophically failed when deployed in the wild [35]. These errors are wide ranging: experts in network congestion control algorithms [52] to flu predictions by Google [35] have also encountered similar issues.

As a result, we believe it is critical to collect data in an ongoing manner to ensure that validation results still hold. This is in stark contrast to the literature that measures model performance on fixed validation sets.

2.4 Inference and End Use

Once a model is validated, it is deployed for its end use.

Inference. One key aspect of this process is model inference. In model inference, the set of inputs is transformed into a “useful” output. This is most commonly a *structured* output(s). For example:

(1) Sentiment analysis transforms unstructured text to a sentiment (e.g., positive or negative).
(2) Object detection transforms unstructured images to a set of boxes and object categories.

While seemingly obvious, this distinction is critical for a range of techniques to monitor ML model deployments.

Much work in the systems and ML communities focuses on improving inference performance [11, 15, 31, 43]. For example, the MLPerf inference benchmark measures the throughput and latency of accelerators on a range of tasks [43]. While important, improvements to inference do not always translate to improvements to the end use.

End use. Once the outputs of the model are generated they are then used for some end use case. In this work, we focus on models that produce structured outputs. A range of generative tasks (e.g., image editing, chatbots) do not produce structured outputs, which we do not focus on in this work.

The structured outputs are used to take actions or perform analyses. For example, they can be used to decide on control actions for an autonomous vehicle. They can also be used to do business analytics, e.g., analyzing which customers are most likely to churn.

ML deployments are also increasingly being used in scientific analyses. These analyses often have strict requirements on accuracy *with respect to ground truth.* In particular, validation accuracy does not provide any guarantees on accuracy on the deployed data, as we describe above.

2.5 Curation of Training Data

A critical part of many ML deployments is the *ongoing* curation of *high quality* training (and validation) data. Many ML deployments are constantly exposed to new types of data, so require new training data to adapt to new scenarios. For example, the Tesla Cybertruck is visually distinct from older trucks. The failure of Google flu trends to generalize across years also demonstrates the importance of ongoing curation of training data. We highlight two aspects of training data curation of importance.

The first is that training data curation is an *ongoing* process for many organizations. In particular, datasets and ML deployments are not static, so organizations must collect new data. This is of particular importance to safety-critical ML deployments (e.g., autonomous vehicles).

The second is that training data must be *high quality* and *high value.* In contrast to a large body of work that assumes given labels are correct, real-world labels vary widely in accuracy. For example, the Lyft Level 5 Perception dataset is rife with errors. We show examples of missing vehicles in Figure 3. Furthermore, the marginal value of data varies greatly depending on the end task, which we highlight in Section 4.

2.6 Examples

We provide two examples of end-to-end ML deployments.

Autonomous vehicles. Autonomous vehicles deploy a number of ML models in the vehicle. In this example, we focus on LIDAR perception models, which are used to detect 3D boxes of objects (vehicles, pedestrians, etc.). The primary goal of the model builders and team is to create a model that results in no accidents. As this objective is difficult to measure, a common goal is to create a model as accurate as possible. We contrast this goal with the goal of the ecological analysis use case below.

Teams of ML researchers focus on developing new models and training algorithms for 3D LIDAR detection. The state-of-the-art has progressed from 44.9 NDS [34] to 75.0 NDS [38] on the nuScenes dataset [12], a remarkable increase in performance in only two years. Given trained models, they are evaluated on a held out set of data, which is separate from the training set. The most common metric for 3D box detection is mean average precision (mAP).

If the model is suitable for deployment, then it is deployed in the car, where it sees new data. Due to the large volumes of data, it is infeasible to even store all of the data seen by the models. As a result, selecting which data to keep and which data to label is challenging.

A separate team typically manages data curation, management, and quality control. Data curation is a difficult and often ad-hoc process. Data is typically sampled at random until problematic events are surfaced. Then, data scientists manually search for events that are similar to problematic events.

Ecological analysis. Our collaborators in the Stanford biology department wish to find instances of hummingbirds feeding in field video for downstream manual analysis. The dataset is approximately 9TB and 200 continuous camera-days of video.

In this example, the primary goal is selection. Since exhaustive manual annotation is infeasible and the downstream analysis can tolerate some missing events, the goal is to select 80% of the video where hummingbirds appear (as measured by total time of appearance).

There are a number of differences with the autonomous vehicle setting. We have a limited amount of resources (computational, labeling, development) compared to autonomous vehicle companies. Furthermore, the goal is to *select* instances of hummingbirds as opposed to training a high quality detection method. As we describe, we can leverage the temporal nature of hummingbird visits to select difficult examples of hummingbirds. In contrast, this approach is not feasible for decision making in autonomous vehicles.

3 MONITORING ML DEPLOYMENTS

In this section, we describe and extend work on monitoring ML deployments.

A critical aspect of many ML deployments these monitoring methods leverage is that the *output* of many ML models is structured (e.g., boxes of cars). This is in contrast the to *unstructured* inputs to many ML models, e.g., images. Since structured records are both easier for domain experts to reason over and computationally cheaper to process, we believe leveraging these structured outputs is a promising way of monitoring ML deployments.

We describe two methods of incorporating domain knowledge to monitor ML deployments over the structured outputs of ML models. Both methods allow multiple users to add monitoring without being experts in ML model development. We have found this property to be useful for large teams.

3.1 Incorporating Domain Knowledge for Monitoring

As mentioned, the key insight we leverage is that the outputs of ML pipelines are *structured*, i.e., the outputs have a meaningful schema. Importantly, both labels and the outputs of ML models are structured in the settings we consider. As such, we can apply similar techniques to monitor both ML model outputs and labels. Furthermore, we can leverage labels to monitor ML model outputs and vice versa.

An important property of monitoring ML pipelines is that, increasingly, non-experts in ML are aiding in monitoring. As a result, we focus on simple interfaces that are agnostic to input data types and specific ML models.

3.2 Manually Specifying Assertions

Overview. We first describe model assertions [32], which allow domain experts to manually specify when errors in ML models may be occurring.

Model assertions are black-box functions provided by domain experts. As inputs, model assertions take one or more inputs to ML models and outputs of ML models. For temporal data, this is typically a recent history of ML model inputs and outputs. The output of a model assertion is real number, where a zero is an abstention.

The API for implementing model assertions with temporal information is as follows:

```
def Assertion(model_inputs: List[ModelInputs], model_outputs: List[ModelOutputs]) -> Float
```

and for point assertions is as follows:

```
def Assertion(model_inputs: ModelInputs, model_outputs: ModelOutputs) -> Float
```

We provide concrete instantiations of implementing assertions below.

Examples. We give several examples of model assertions. As we show, model assertions can be used across domains, tasks (including beyond simple classification), models, and modalities. In contrast, a range of existing techniques only applies to specific tasks (e.g., only classification [47] or only detection [10]), models (e.g., only linear models), etc.

Autonomous vehicles. Autonomous vehicles have a number of perception models that predict information from sensor data (e.g., from LIDAR point clouds, camera pixels, and sonar readings). The goal of these perception models is to produce accurate predictions about object types, positions, and other attributes (e.g., if a car is parked or not).

Since these predictions are about the real world, we have found that simple assertions about real-world behavior are surprisingly effective. For example, if a predicted box of a car is “flying,” it is unlikely to be a correct prediction (e.g., if its predicted z-coordinate is above 10 meters high). Similarly, a car with a velocity of over 120 mph is unlikely to be predicted correctly.

We provide a pseudocode implementation of the flying assertion:

```
def CarNotFlying(point_cloud: PointCloud, boxes: List[Box3D]):
  nb_violations = 0
  for box in boxes:
    if box.z > 10m:
      nb_violations += 1
  return nb_violations
```

As shown, these assertions are often simple to implement, requiring few lines of code.

Video analytics. Similarly for video analytics, many models are used to predict information about real-world behavior. Consider the case of using traffic camera data to perform analytics over traffic patterns. As shown in Figure 2, ML models can produce highly unrealistic predictions. As a concrete example, we can write a simple assertion that states that three boxes of cars should not overlap. To implement such an assertion, an analyst could provide the following code:

```
def NoMultipleOverlap(image: Image, boxes: List[Box2D]):
  nb_violations = 0
  # Iterate over the Cartesian product of the three lists
  for box1, box2, box3 in itertools.product(boxes, boxes, boxes):
    # If the boxes are the same box, skip this iteration
    if box1 == box2 or box2 == box3 or box1 == box3:
      continue
    # If the three boxes simultaneously overlap, we have a violation
    if box1.overlaps(box2) and box2.overlaps(box3) and box1.overlaps(box3):
      nb_violations += 1
  return nb_violations
```

Although this assertion can be rewritten for higher performance, we present the $O(N^3)$ version for clarity of presentation. As above, this assertion can be written in under 10 LOC.

In this example, such predictions can occur for other classes where the model was initially designed for. For example, boxes of humans or other animals can highly overlap in images. As a result, it is not desirable to apply such constraints in general, but only in an application-specific manner.

Translation. Consider an analyst deploying an ML translation model. We trained a model as suggested by the PyTorch tutorial on translation [44]. This example is novel to this work.

We constructed a simple assertion that asserts that two words should not repeat in the translated output:

```
def NoRepeatedWords(original: Word[], translation: Word[]):
  nb_repeats = 0
  for i in range(1, len(translation)):
    if translation[i - 1] == translation[i]:
      nb_repeats += 1
  return nb_repeats
```

After executing this assertion, we found that *6.4% of the test set triggered this assertion.* Thus, simple assertions can find errors in widely used models.

Housing price prediction. Consider an analyst predicting housing prices from features of the surrounding area. As an example, we took the popular `sklearn` Boston housing prices dataset [21]

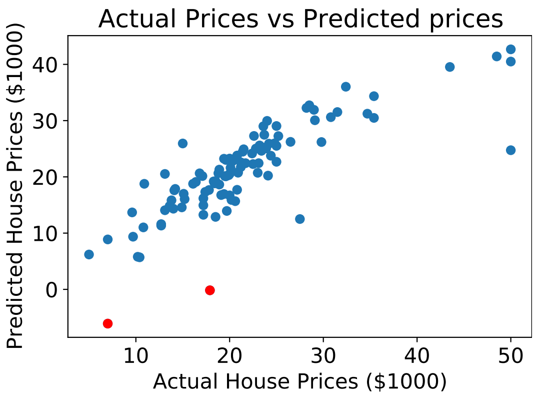


Fig. 4. Example of errors when predicting housing prices. As we see, two of the examlpes are predicted as having negative housing prices, which is infeasible.

and a standard example online training a linear regression model to predict the housing prices [20]. This example is novel to this work.

As we show in Figure 4, two of the test data points (i.e., houses) are predicted as having a *negative* housing value, which is not valid. To prevent such erroneous predictions, the analyst could write a simple model assertion that asserts predicted housing prices should be positive:

```
def OutputPositive(features: HousingFeatures, output: Float):
  if output < 0:
    return 1
  else:
    return 0
```

We note that this example is over structured inputs (i.e., tabular inputs) and structured outputs, showing that model assertions apply beyond unstructured inputs.

As we have shown, model assertions can be applied to a wide range of tasks, models, and modalities with few lines of code (<10 LOC in all our examples). Due to their simplicity, we have found it easy to integrate into existing pipelines.

3.3 Consistency Assertions

Overview. A large class of assertions that we have found to be useful across domains are consistency assertions. Consistency assertions check whether or not predictions are concordant across time, modalities, or views.

These assertions can be specified via a simple API. Given a set of associated data and predicted attributes, the model assertion can simply trigger when there are disagreements. The model assertion can also return the number of disagreements if desired.

An important application of consistency assertions are *finding errors in human labels*, which is of critical importance for high-stakes ML deployments (e.g., autonomous vehicle deployments).

We provide several examples of using consistency assertions to find errors in ML model predictions below. We discuss how to find errors in human labels in Section 4.2.

Examples. We describe several examples of consistency assertions.

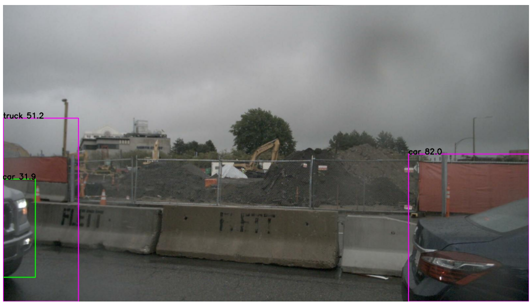


Fig. 5. Example of a model prediction error in the nuScenes autonomous vehicle dataset. This error was found via a consistency assertion between different sensor views of the same datas (3D point cloud and camera data). Taken from Kang et al. [32].

Autonomous vehicles. We provide two examples of finding errors in 3D box prediction from LIDAR point clouds.

First, consider the setting where multiple models produce predictions over different sensors. For example, in addition to the 3D box prediction model, a 2D object detection might be executed over camera data. In this setting, an analyst can write consistency assertion between different views: the 2D projection of the 3D box via LIDAR should match with the 2D prediction from the camera. Similarly, we have found errors in the publicly available nuScenes dataset using this method [12]. We show an example of such an error in Figure 5.

Second, consider the setting of only 3D box predictions and a high enough time resolution (e.g., 10 Hz or higher). In this setting, the analyst can write a consistency assertion that specifies that a box in frame t and $t + 2$ that overlap with IOU at least 0.5 should have a box in frame $t + 1$. We have found that this assertion can also find errors in state-of-the-art ML models.

Video analytics. Consider the setting of performing traffic analysis from camera data where the frame rate is 30 frames per second. Suppose the analyst uses a 2D object detection model to localize and classify vehicles.

Similarly to the autonomous vehicle setting, the detection model is predicting properties of the real world via sensor data. As such, the analyst can deploy assertions that encode simple properties about the real world. For example:

(1) A time-consistency assertion as above (a box in frames t and $t + 2$ implies a box in frame $t + 1$).
(2) A separate time-consistency assertion asserts that a box of an object should not appear for only one frame (i.e., a box in frame t should have an overlapping box in frame $t - 1$ or $t + 1$).
(3) An assertion that says a continuously tracked object should have the same class prediction (e.g., a car should not switch to a truck).

Heart arrhythmia detection. Consider the setting where a model predicts heart arrhythmias from ECG time-series data [42]. These models are often trained on short snippets of ECG readings as these short snippets are the easiest for doctors to label. When deployed on real data, the models are expected to predict over continuous data.

As above, the analyst can deploy a simple time-consistency assertion. Heart arrhythmia events typically do not start and restart in under 30 seconds, so high frequency changes in model predictions are likely to be erroneous.

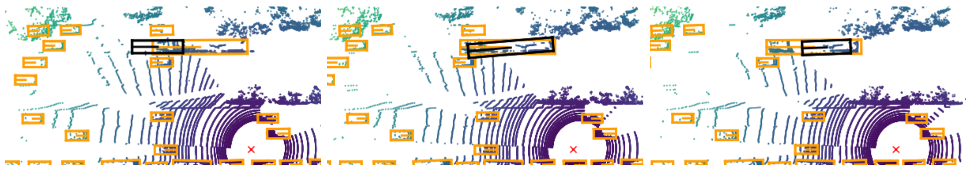


Fig. 6. Example of a model prediction error on the Lyft Level 5 dataset. While the predicted boxes (in black) overlap with the ground truth boxes (in orange), they are inconsistent over time and can be found with Fixy as a result. Taken from Kang et al. [24].

While many notions of consistency are easy to define (i.e., "hard" consistency), many notions of consistency are "fuzzy". As a result, we have extended model assertions to learn consistency from data.

3.4 Learning Assertions

Overview. While many notions of consistency can be encoded with exact rules (i.e., as described above), there are other notions of consistency which users may want to encode. For example, consider a model that predicts 3D boxes from LIDAR data in an autonomous vehicle which predicts the boxes in Figure 6. As shown, the boxes across three timesteps overlap, but in an unrealistic way. While such an assertion could be specified manually, it can be difficult to specify all such cases. In many cases, users wish to specify "soft" assertions, as opposed to "hard" ones.

As a result, we have developed learned observation assertions (LOA), which can find such "soft" violations as described above. Instead of users specifying each violation manually, they simply specify attributes over predictions (e.g., box volume). Then, our system that implements LOA, Fixy, will learn which observations are violations from existing predictions.

LOA takes advantage of two properties of ML deployments. First, many ML deployments collect human labels and older model predictions. These existing labels and predictions can be used to learn which subsequent predictions are possibly erroneous. Second, these human labels and ML model predictions are often correct on average. As a result, they can be used to learn reasonable distributions over subsequent model predictions.

LOA was designed for auditing autonomous vehicle labels. As such, it explicitly models temporal structure. However, we believe LOA can be applied to any setting with temporal structure beyond autonomous vehicle data.

Deploying LOA. We provide an overview of deploying LOA via Fixy and defer the formal specification of the language to Kang et al. [24].

To use Fixy, a user need only provide:

(1) Access to existing human labels.
(2) Simple functions to compute features and associations over predictions/labels.
(3) Whether or not to select likely or unlikely predictions/labels.

Given these inputs, Fixy will automatically rank which predictions or labels are most likely to be erroneous.

We provide an example of deploying LOA via Fixy for autonomous vehicle data for finding erroneous ML model predictions. We describe how to use Fixy to find missing human labels in Section 4.2.

To find erroneous ML model predictions, the analyst can use the class-conditional features of box volume, object velocity, and track length. Then, the analyst specifies to Fixy that they are looking for *unlikely* tracks.

Given the specification, Fixy will look for unlikely tracks among the ML model predictions. We show an example in Figure 6, where the predicted truck box is highly inconsistent across frames. As a result, Fixy will flag such a track highly.

We show quantitative results of Fixy finding errors in human labels in Section 4.

4 CURATION AND QUALITY CONTROL OF LABELED DATA

A critical component of ML deployments is the curation of *high-quality* training data. The curation of this data involves two steps: 1) selecting which data to label and 2) ensuring the labels have high quality.

ML deployments have limited resources to label data: human labels are orders of magnitude more expensive than ML model inference. As a result, ML pipeline deployers must select which data to label.

In contrast to standard assumptions that human-labeled data is ground truth, an emerging body of work has shown that datasets are rife with errors. These errors can cause cascading failures in ML deployments and are thus of critical importance to find. Quantitatively, recent work has shown that label errors can cause a model capacity to degrade by up to 3× compared to training in clean data [39].

As such, we describe how assertions can be used to select and perform quality assurance over labeled data.

4.1 Selecting Data

Overview and comparison to active learning. Selecting data to label is closely related to active learning [47]. In active learning, data is labeled and added to a training set to improve model quality. However, ML deployments often have requirements outside of the constraints of traditional active learning:

Standard active learning aims to add data to the training set, as it assumes the validation/test sets are representative. In contrast, we aim to select data to label for the training, validation, and test sets. This is especially important as validation and test datasets are often selected specifically to *not* reflect the input distribution, but to cover rare events as well.

Active learning also aims to connect specific objectives (e.g., high binary classification accuracy) to the data collection process. In contrast, a wide range of ML deployments we have encountered are interested in finding examples where the ML model produces *incorrect predictions*. Incorrect predictions are important for several reasons. First, analysts often wish to perform manual analysis on data where the model is underperforming. These analyses often inform decisions beyond the scope of traditional active learning, such as changing the model architecture or changing control algorithms to avoid such situations. Second, knowing where the model is failing is critical for liability reasons. Third, data where the model is incorrect is often more valuable *for improving model quality* than standard theory may suggest for complex deep learning models.

Finally, active learning is often specific to at least one of task, model type, and training objective. For example, active learning techniques for classification [47] do not apply to object detection and vice versa [10]. Other active learning techniques are specific to the form of the model (e.g., logistic regression models) [46]. Since tasks and model types are constantly emerging, we have focused on constructing APIs that are agnostic to the task, model type, and training objective.

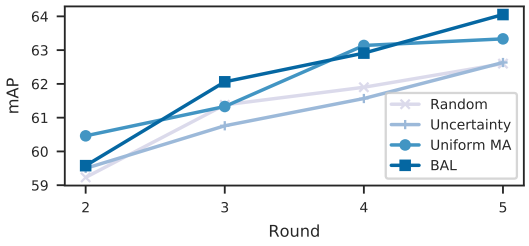


Fig. 7. Performance of active learning when using random sampling, uncertainty sampling, and model assertions. BAL is the bandit active learning algorithm that leverages model assertions. The x-axis is the round of training and the y-axis is the mAP. As shown, BAL can achieve the 62 mAP target with 40% fewer labels compared to baselines. Taken from Kang et al. [32].

Selecting incorrect predictions. To select data for labeling, we use the assertions as described in Section 3. As described, the assertion interface is agnostic to data type (as assertions primarily operate on the structured outputs), task, and model type.

Given a set of inputs that assertions trigger on, an important question is how to select which data points to label given limited resources. We note that many ML deployments have objectives outside of maximizing average accuracy. For example, certain classes of errors might be more important to reduce for liability or business reasons (e.g., errors in detecting objects closer to an autonomous vehicle). As a result, end users may be interested in custom weightings for selecting data.

However, in many cases, we have found that reducing the fraction of data that assertions trigger on is often a good starting heuristic. To understand why, consider the setting where assertions have perfect precision (i.e., if an assertion triggers, the data point has an erroneous ML prediction). In this setting, if no assertions trigger, then the model entirely avoids the classes of errors the assertions specify.

Evaluating assertions for training. To evaluate assertions for training quality improvements, we compared assertions to standard baselines of random sampling and uncertainty sampling. We performed five rounds of active learning on a video analytics dataset for 2D box detection. We measured the mean average precision (mAP), a standard metric for 2D object detection.

We show results in Figure 7. As shown, using assertions outperforms baselines. At a fixed accuracy, they can require up to 40% fewer labels.

4.2 Quality Control of Labeled Data

In contrast to a wide range of work that assumes labeled data is ground truth, we have found widely used datasets are rife with errors. An emerging body of work also observes similar issues with popular datasets [39].

While these errors may seem innocuous, they are critical to find. Many mission-critical ML deployments have strong legal liability reasons to find errors in labeled data. Furthermore, recent work has shown that 6% label noise can effectively reduce model capacity by over 3× [39] (e.g., making a ResNet-50 perform at the same level as a ResNet-18 on clean data).

As such, we describe methods that leverage assertions to find errors in labeled data.

4.2.1 Overview. Errors in human-labeled “ground truth” data can degrade model quality (if in training data) and lead to misleading metrics (if in validation or test data). As such, examining human labels for errors is of increasing importance.

Since ML deployments are continuous, ML models are often correct on average. As a result, predictions from ML models can be used to vet human labels. In particular, the simple procedure of selecting the highest confidence model predictions disagreeing with human labels can find numerous errors in production datasets.

While seemingly simple, *even production autonomous vehicle datasets* do not leverage this idea. For example, the Lyft Level 5 dataset is rife with errors, as shown in Figure 3. We describe the setting and provide pseudocode in detail below.

4.2.2 Finding errors via manual consistency assertions. We consider the setting of finding errors in 3D box predictions from LIDAR point cloud and camera data. The most important errors to discover are ones where the human labeler entirely missed the object: this is for both liability and model performance reasons.

To find such errors, the analyst can take a trained 3D box prediction model (e.g., PointPillars [34]) and produce predictions over the labeled data. Then, the analyst can filter the predictions and select only those that do not overlap with an existing human label. Finally, the analyst can rank order the remaining predictions and manually analyze them for missing labels.

We show that, even when trained on noisy data, ML models can learn sufficiently well to find examples of missing human labels.

4.2.3 Finding errors with learned assertions. As with consistency assertions, we leverage existing ML models present in ML deployments. Instead of returning individual boxes with highest confidence not overlapping a human-labeled box, we use Fixy to find continuous *tracks* of boxes that are likely to be missing objects.

To do so, the analyst can specify the following associations and features. First, the analyst associates boxes with overlap across adjacent frames to form tracks. Then, the analyst can specify the following class-conditional features over the data: box volume, object velocity, and track length. Finally, the analyst specifies that they are looking for *likely* tracks.

Given these inputs, Fixy will rank order the tracks by those likely to be consistent, and therefore objects missed by the human labeler. We show several examples of missing tracks found by Fixy in Figure 3. We emphasize that these labels were generated by a leading label vendor and a well funded engineering organization.

4.2.4 Examples of errors. To show the utility of assertions for finding errors in human labels, we applied assertions to two autonomous vehicle datasets. We are unaware of other techniques to find missing human labels in detection datasets.

The datasets we evaluated on were the publicly available Lyft Level 5 perception dataset and an internal dataset at the Toyota Research Institute (TRI). The Lyft Level 5 dataset was labeled by a leading vendor for human labels: Waymo, Cruise, Uber, and Lyft have all used labels from this vendor [1].

We trained 3D object detection models on the datasets respectively and searched for errors in cars, trucks, and motorcycles.

For errors in human labels, we are primarily concerned with discovering missing human labels. This is for two reasons: 1) finding missing human labels is the most important for liability and performance reasons and 2) auditing existing labels is much less costly.

Results: recall. A critical question when applying assertions is the *recall* of finding errors, i.e., the total number of missing human labels found when using assertions. Many such errors are apparent when found, e.g., the errors highlighted in Figure 3. Unfortunately, evaluating recall is difficult, as it requires finding all errors in data. As a result, we present targeted experiments on evaluating recall.

Method	Dataset	Precision at top 10	Precision at top 5	Precision at top 1
Fixy	Lyft	**69%**	**70%**	**67%**
Consistency assertions (rand)	Lyft	32%	30%	24%
Consistency assertions (conf)	Lyft	39%	40%	39%
Fixy	Internal	**76%**	**100%**	**100%**
Consistency assertions (rand)	Internal	49%	64%	66%
Consistency assertions (conf)	Internal	71%	86%	66%

Table 2. Precision at top 10 of Fixy and manual consistency assertions for finding tracks missed by humans. Assertions can achieve precision as high as 100% and remains high even when searching for 10 errors per scene. Taken from Kang et al. [24].

We first evaluated the recall of LOA exhaustively on an 15 second scene from the internal TRI dataset. The vendor provided labels that missed 24 objects. Using LOA, we found 75% (18) of the errors in the top 10 errors per class.

We next evaluated the recall of LOA on the Lyft Level 5 dataset. We manually examined every scene in the validation dataset (i.e., those not seen by the model at training time) and discovered 32 scenes with errors, out of 53 total scenes.

Namely, *70% of scenes in the Lyft Level 5 validation set contained at least one error*. Impressively, LOA found at least one error among *every validation scene that contained an error*. In other words, LOA had a 100% recall on scenes with errors in the Lyft Level 5 dataset.

Results: precision. In addition to measuring recall, we measured the precision of LOA and manual consistency assertions. As we show, leveraging learned assertions can improve precision beyond simple, manually consistency assertions.

We show the precision of manual consistency assertions and LOA for finding errors in the Lyft Level 5 dataset and our internal dataset in Table 2. As we see, the precision of consistency assertions can be as high as 100% (at top one). The precision remains high when searching for the top 10 errors per scene.

5 ANALYTICS WITH GUARANTEES

Finally, as a case study of an end-to-end ML pipeline, we describe how to leverage ML to perform analytics with *guarantees on accuracy with respect to an expert human labeler*. Many analytics systems assume that ML models return correct answers [2, 6, 27], which is an invalid assumption [35]. As a result, these systems focus on optimizing the runtime of executing analytics queries with respect to some ML model.

We describe methods of combining sampling from expert human labelers while leveraging ML models to improve sampling efficiency. Namely, ML models augment human effort. Throughout this section, ML models are used as a *proxy*, or approximation, for expert human labels.

We describe several use cases for analytics with ML-augmentation and describe how to perform selection queries with guaranteed recall and aggregation queries with guaranteed error bounds.

5.1 Use Cases

Many applications that require accuracy with respect to an expert human labeler are scientific in nature. In particular, many scientific claims are frequentist in nature, e.g., that the null hypothesis is rejected with some failure probability.

We describe two analytics queries with several examples of applications for each. These applications feature the common property that the datasets are far too large to manually annotate. As a result, analysts and scientists turn to sampling methods, which we show can be augmented with ML.

Aggregation queries. A range of applications aim to compute linear statistics over large datasets (e.g., counts, sums, averages). For example, a traffic analyst may wish to compute the average number of cars per frame of a video to compare how busy two different intersections are. A social scientist studying the effect of newspaper media on public views on government program may compute the average sentiment of opinion articles discussing the New Deal.

As these queries are used for scientific applications, the scientists often desire frequentist guarantees on coverage of confidence intervals over the reported statistic.

Selection queries. Other applications aim to select events from large datasets. These events are typically rare and are used in downstream manual analysis. For example, our collaborators in the Stanford biology department desire to find 80% of the visits of hummingbirds to a bush in field video, as measured by time. An urban planner may be interested in finding 80% of instances of events where a car cuts off a cyclist in traffic camera video.

5.2 Aggregation with Guarantees

5.2.1 Query semantics. Consider a dataset $\mathcal{D} = \{x\}$ where $|\mathcal{D}|$ is large. Consider an oracle function $\mathcal{O}(x) \in \mathbb{R}$, which computes the per-example statistic of interest (e.g., number of cars via an expensive detection model).

The goal is to compute approximately compute $\mu = \mathbb{E}_{\mathcal{D}}[\mathcal{O}(x)]$ with as few invocations of $\mathcal{O}(x)$ as possible. Furthermore, we aim to return a confidence interval $[\underline{\mu}, \bar{\mu}]$ that is as tight as possible subject to a coverage guarantee. The coverage guarantee is specified as a user-provided failure probability δ.

We are given access to a proxy $\mathcal{P}(x)$ that approximates $\mathcal{O}(x)$, i.e., that $\mathcal{P}(x) \approx \mathcal{O}(x)$.

5.2.2 Prior algorithms. Prior work on aggregation queries optimizes approximate aggregation via sampling. As a naive baseline, random sampling can already reduce computational costs by over 100×, depending on the dataset size and error tolerance.

However, random sampling does not leverage that many applications have proxies that are easy to construct. We defer discussions of constructing these proxies to other manuscripts [25, 30]. To leverage proxies for sampling, the BlazeIt system uses them as a variance reduction technique in the form of control variates. Specifically, given the quantities above, we can form the unbiased estimator

$$\mathcal{O}^* = \mathcal{O} + c(\mathcal{P} - \mathbb{E}[\mathcal{P}]) \tag{1}$$

where c is any arbitrary constant. The variance minimizing c is equal to $-\frac{\mathrm{Cov}(\mathcal{O},\mathcal{P})}{\mathrm{Var}(\mathcal{P})}$.

Finally, to achieve the error guarantees, BlazeIt uses an always-valid stopping rule, EBS sampling. Intuitively, EBS sampling "uses" a small portion of the failure budget at every sample to ensure validity. EBS sampling makes no assumptions on the statistic except that it lies within some range.

As we show below, this can result in suboptimal stopping, given minor assumptions about the relationship between the proxy and oracle.

5.2.3 Optimized algorithm. We show that leveraging reasonable priors about how the proxy is correlated with the oracle statistic can result in suboptimal sampling. To address this, we present BlazeIt++, an optimized aggregation algorithm.

Overview. To understand the intuition behind BlazeIt++, consider Equation 1. In particular, the variance of $\mathcal{O}^*$ is equal to:

$$\mathrm{Var}(\mathcal{O}^*) = (1 - \rho^2_{\mathcal{O},\mathcal{P}})\mathrm{Var}(\mathcal{O})$$

Algorithm 1 Optimized aggregation algorithm, BlazeIt++. If the proxy is correlated with the oracle, the control variates estimator will have variance lower than in the worst case. As a result, the assumptions for EBS sampling are pessimistic. To address this, BlazeIt++ first estimates the variance before proceeding. Here, Z is the two-sided z-value function. W is the Lambert W function.

```
1:  function BlazeIt++($\mathcal{O}, \mathcal{P}, \epsilon, \delta$)
2:      $S \leftarrow$ ControlVariates($\mathcal{O}, \mathcal{P}$, RandomSample($\mathcal{D}$, 1000))
3:      $\hat{V} \leftarrow \mathrm{Var}(S)$
4:      $\hat{V}_{\mathrm{UB}} \leftarrow \frac{(1000-1)\hat{V}}{\chi^2_{\delta/2}}$                ▷ Chi-Squared Tail Bound
5:      $i \leftarrow 0$
6:      $\phi \leftarrow \frac{\delta}{4}$
7:      $n \leftarrow \frac{Z(\phi)^2 \hat{V}_{\mathrm{UB}}}{\epsilon^2}$
8:      while true do
9:          $S \leftarrow$ ControlVariates($\mathcal{O}, \mathcal{P}$, RandomSample($\mathcal{D}, n$))
10:         $\hat{\mu},\ \hat{\mu}_{\mathrm{LB}},\ \hat{\mu}_{\mathrm{UB}} \leftarrow$ EstimateWithCI($S, \hat{V}_{\mathrm{UB}}, \phi, \max(\mathcal{P})$)
11:         if $\hat{\mu}_{\mathrm{LB}} + \epsilon < \hat{\mu}_{\mathrm{UB}} - \epsilon$ then
12:             break
13:         $\phi \leftarrow \frac{\phi}{2}$
14:         $n \leftarrow 2n$
15:     return $\hat{\mu}$
16:
17: function EstimateWithCI($S$, $\hat{V}_{\mathrm{UB}}$, $\delta$, $C$)
18:     $\hat{\mu} \leftarrow \mathbb{E}[S]$
19:     $t \leftarrow \frac{-C^2 \log(\delta)}{\hat{V}_{\mathrm{UB}}}$                ▷ Bennet's Inequality
20:     $t \leftarrow e^{W_0(\frac{h-1}{e}+1)} - 1$                ▷ Lambert W Function
21:     $t \leftarrow \mathrm{Real}(t\hat{V}_{\mathrm{UB}}/C)/|S|$
22:     return $\hat{\mu}, \hat{\mu} - t, \hat{\mu} + t$
```

where $\rho_{\mathcal{O},\mathcal{P}}$ is the correlation between $\mathcal{O}$ and $\mathcal{P}$. Namely, the more correlated the proxy is with the oracle, the lower the variance of the control variate estimator.

We assume a natural property of the proxy: that $\mathcal{P}$ is non-trivially correlated with $\mathcal{O}$. However, we note that *this assumption is only necessary for the performance, not correctness* of our algorithm.

We present BlazeIt++ in Algorithm 1. It operates by forming an estimate of the total number of samples necessary to achieve the error target given the assumptions above. Namely, it first estimates the variance of the control variate sampler using 1,000 random samples. It then estimates the total number of samples necessary to achieve the error bound given the estimated variance. If the sample size is too small, it iteratively doubles the sample size until the error target is achieved.

To compute the confidence intervals, we invert Bennett's inequality [8], which provides sub-Gaussian, finite-sample tail, bounds. As a result, we recover the same guarantees as EBS sampling.

Correctness. We prove correctness by first showing that the algorithm terminates and then by showing that the algorithm is valid at any termination point.

Termination follows since the dataset is finite and the number of samples increases at every iteration.

Correctness follows by the union bound. Suppose the algorithm terminates at step i. Then, by the union bound, the probability that the confidence interval is valid is at least $\sum_{j=0}^{i} \frac{\alpha}{2^j} \leq \alpha$.

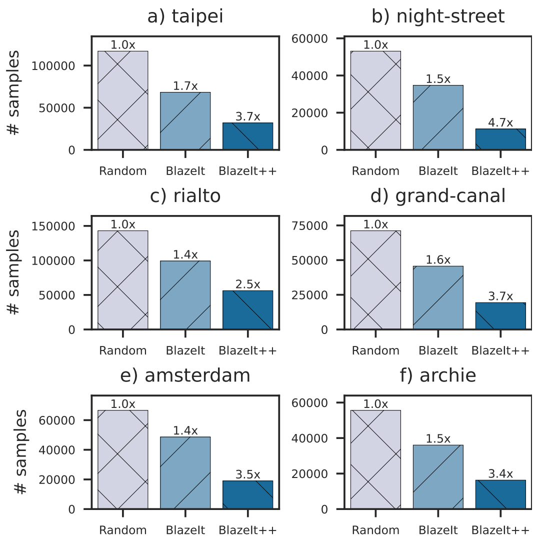


Fig. 8. Performance of BlazeIt and BlazeIt++ on the datasets considered by Kang et al. [25]. The y-axis is the number of samples required to achieve the error target. We show the improvement over random sampling in the numbers above the bars. As shown, our improved algorithm outperforms on all datasets.

5.2.4 *Evaluation.*

To evaluate BlazeIt++, we used the datasets and proxy scores generated by Kang et al. [25] to evaluate the BlazeIt system. We then executed random sampling, the algorithm BlazeIt uses, and BlazeIt++. We targeted an error of 0.01 and a success probability of 95%.

We show results in Figure 8. As shown, BlazeIt++ outperforms on all datasets, outperforming random sampling by up to 4.7× and BlazeIt by up to 3.3×. Thus, by leveraging reasonable assumptions about the proxy and oracle, we can construct higher performing sampling algorithms.

5.3 Selection with Guarantees

5.3.1 *Query semantics.*

As before, consider a dataset $\mathcal{D} = \{x\}$ where $|\mathcal{D}|$ is large. Instead of considering a real valued oracle, consider an oracle $O(x) \in \{0, 1\}$, which represents the result of a boolean predicate. For example, $O(x)$ may represent if a car is present in a frame of a video.

For $\mathcal{S} \subset \mathcal{D}$, define the recall and precision of $\mathcal{S}$:

$$P_{\mathcal{S}} = \frac{\sum_{x \in \mathcal{S}} O(X)}{|\mathcal{S}|} \tag{2}$$

$$R_{\mathcal{S}} = \frac{\sum_{x \in \mathcal{S}} O(X)}{\sum_{x \in \mathcal{D}} O(X)} \tag{3}$$

The goal is to return a set $\mathcal{S}$ such that $R_{\mathcal{S}}$ is above some recall target R and that $P_{\mathcal{S}}$ is maximized.

We are given a proxy $\mathcal{P}(x) \in [0, 1]$ that approximates $O(x)$.

5.3.2 *Prior algorithm.*

Suppose we are given a calibrated proxy $\mathcal{P}(x)$, i.e., a proxy such that

$$\mathcal{P}(x) = \Pr_{x \sim \mathcal{D}}[O(x) = 1 | \mathcal{P}(x)]. \tag{4}$$

Intuitively, if a calibrated proxy returns 0.9, then 90% of the time, $O(x)$ will evaluate to true.

Kang et al. [28] developed SUPG to perform approximate selection. To perform approximate selection, SUPG orders $\mathcal{D}$ by proxy score and returns all records above some threshold C, i.e., all records with $\mathcal{P}(x) > C$. To achieve the recall target, SUPG uses importance sampling to generate a

Algorithm 2 Optimized SUPG algorithm, SUPG++. SUPG++ forms an estimate of the total number of positive examples and up-weights the top ranked records by proxy score. It then leverages the standard SUPG algorithm.

```
1: function SUPG++($\mathcal{O}, \mathcal{P}, R, \delta$)
2:     $E_{\mathcal{P}} \leftarrow \mathbb{E}[\mathcal{P}(x)]$
3:     $C = 3E_{\mathcal{P}} \cdot |\mathcal{D}|$
4:     $\mathcal{P}$' $\leftarrow$ UpweightTopC($\mathcal{P}, C$)
5:     return SUPG($\mathcal{O}, \mathcal{P}$', $R, \delta$)
```

sample $\mathcal{S}$. Then, it forms estimates of the fraction of records above and below an adjusted recall cutoff. Given these estimates, SUPG forms an estimate of the cutoff C to achieve the recall target. Namely, given the empirical cutoff τ SUPG constructs the sets

$$Z_1 = \{1_{\mathcal{P}(x) \geq \tau}\mathcal{O}(x) : x \in \mathcal{S}\} \tag{5}$$

$$Z_2 = \{1_{\mathcal{P}(x) < \tau}\mathcal{O}(x) : x \in Set\} \tag{6}$$

and upper/lower bounds the mean over these sets to form an adjusted recall target:

$$\frac{\text{UB}(\hat{\mu}_{Z_1})}{\text{UB}(\hat{\mu}_{Z_1}) + \text{LB}(\hat{\mu}_{Z_2})} \tag{7}$$

Kang et al. [28] showed that to estimate $\mathbb{E}[\mathcal{O}(x)]$, the variance minimizing strategy given a calibrated proxy is to sample proportional to $\sqrt{\mathcal{P}(x)}$. Since this quantity is similar to the quantities estimated in Equation 7, SUPG samples according to $\sqrt{\mathcal{P}(x)}$.

5.3.3 Optimized algorithm. We propose a modified version of SUPG which leverages properties of proxies that often hold on real datasets.

Overview. The performance analysis of SUPG relies on two assumptions: 1) that $\mathcal{P}(x)$ is calibrated and 2) that minimizing the variance of the estimate of $\mathbb{E}[\mathcal{O}(x)]$ corresponds to minimizing estimates of the means of Equations 5 and 6.

However, these assumptions often do not hold. An emerging body of work in the ML literature shows that ML models are rarely calibrated, but are often monotonic in the proxy score. Furthermore, recall targets are often relatively high (e.g., 90%), so the estimator is more sensitive to Z_1 in practice.

As a result, we propose a modified version of SUPG, SUPG++. We provide pseudocode in Algorithm 2. SUPG++ first estimates the total number of positive examples using $A = \mathbb{E}[\mathcal{P}(x)] \cdot |\mathcal{D}|$. Then, SUPG++ up-weighs the top $3A$ records by proxy score by setting the proxy scores of these records to 1. Given these updated weights, SUPG++ uses the standard SUPG procedure.

Correctness. SUPG's sampling procedure and estimation is valid regardless of the proxy weights. As SUPG++ only modifies the proxy weights, SUPG++ is valid.

Intuition. To understand why SUPG++ performs well, we note that proxies are often *monotonic* although not calibrated. As a result, the rank ordering of records by proxy value is often "good," even though the absolute proxy scores may not be informative. As a result, in many real world settings, the positive records are clustered near the top of the rank ordering. Furthermore, for higher recall targets, the estimator is more sensitive to the total number of positive records sampled. As a result, up-weighting the top *ranked* proxy scores can boost performance.

5.3.4 Evaluation. To evaluate SUPG++, we used the datasets and proxy scores generated by Kang et al. [28] to evaluate SUPG. We evaluated against the importance sampling procedure of SUPG++

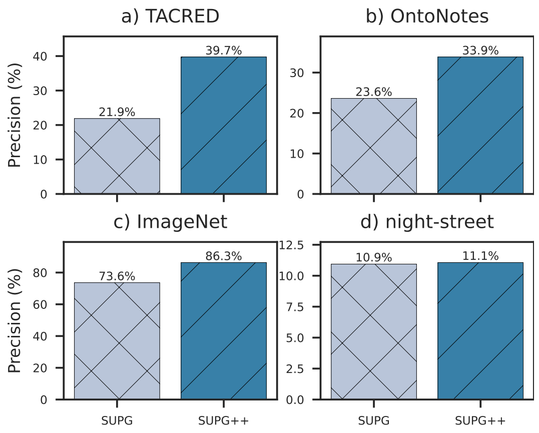


Fig. 9. Performance of SUPG and SUPG++ on the real-world datasets considered by Kang et al. [28] for the 90% recall target. As shown, our improved algorithm outperforms on all datasets.

and the modified importance sampling procedure of SUPG++. We targeted a recall target of 90% and a success probability of 95%. We measured precision, where a higher precision is better.

We show results in Figure 9. As shown, SUPG++ outperforms on all datasets by up to 18% in absolute precision points and 81% in relative terms. As before, we can construct higher performing sampling algorithms by leveraging reasonable assumptions about proxies.

6 RELATED WORK

Work from the ML to systems communities have focused on improving various aspects of ML deployments. Furthermore, an emerging body of work focuses on various aspects that we highlight (e.g. "data-centric AI"). We highlight several related areas of work. We note that this work draws heavily from and extends prior work [24, 25, 28, 32].

Monitoring ML. Monitoring ML has becoming increasingly important. One such line of work focuses on monitoring ML pipelines with semantically meaningful input schemas, i.e., tabular data [7]. This line of work focuses on validating inputs, e.g., that Boolean values are zeros or ones (as opposed to arbitrary values).

Another line of work focuses on monitoring statistical properties of inputs or outputs [9]. For example, if the proportion of a specific input field that is null dramatically increases, this likely signifies an upstream feature generation bug. Actions can then be taken as a result of finding such errors.

While this work is valuable for ML deployments with meaningful input schemas, they do not directly to apply to complex, unstructured data. Monitoring ML deployments over these data types are what we focus on in this work.

ML testing. Recent work aims to test ML systems, often focusing on a specific aspect of deployments. These aspects range from numeric errors (e.g., TensorFuzz [40]) to "formal verification" of properties [19]. These techniques are complementary to the ones we describe and an important part of ML deployments.

ML data curation. ML data curation is closely related to active learning. As discussed, active learning focuses on selecting data points to label to improve specific validation metrics [47]. ML deployments require curation of data for a range of purposes outside of improving validation metrics, including for legal liability, manual error analysis, test set curation, and creating new models. Assertion-based data curation can be used for these extended purposes.

ML data quality control. A recent line of work ("data-centric AI") focuses on the importance of data in ML deployments [41]. For example, Northcutt et al. [39] shows that ten commonly used *classification* ML datasets are rife with errors. In this work, we focus on allowing domain experts to specify expert knowledge to find errors, regardless of data type, model, or task.

Analytics with ML. ML offers a powerful primitive to turn unstructured data into structured data. As such, the analytics community has focused on leveraging ML in queries and optimizing the execution of such queries. Recent work has focused on optimizing selection queries [2, 27, 28], aggregation queries [25, 29], tracking queries [6], indexing [30], fast query execution [31] etc. In this work, we describe analytics with ML as an example of an end-to-end ML pipeline. We describe how to leverage the work from assertions and optimize the pipeline end-to-end with user constraints.

7 CONCLUSION

In this work, we describe commonalities between ML deployment pipelines and interfaces between various components. We describe two underlooked areas of ML deployments: monitoring ML deployments *at deployment time* and the curation of *high quality* training data. Perhaps surprisingly, we show that assertions can be used for both tasks effectively. We finally describe how to optimize ML-based analytics end-to-end as an example of an ML deployment, and how to leverage assertions for analytics.

REFERENCES

[1] Trixia Abrera. 2019. A 22 Year-Old CEO Can Affect The Safety Of Your Next Uber Ride. (2019). https://greyjournal.net/hustle/work-tech/a-22-year-old-ceo-can-affect-the-safety-of-your-next-uber-ride/

[2] Michael R Anderson, Michael Cafarella, Thomas F Wenisch, and German Ros. 2019. Predicate Optimization for a Visual Analytics Database. *ICDE* (2019).

[3] Dolan Antenucci, Michael Cafarella, Margaret Levenstein, Christopher Ré, and Matthew D Shapiro. 2014. *Using social media to measure labor market flows.* Technical Report. National Bureau of Economic Research.

[4] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. 2018. Synthesizing robust adversarial examples. In *International conference on machine learning.* PMLR, 284–293.

[5] Peter Bailis, Edward Gan, Samuel Madden, Deepak Narayanan, Kexin Rong, and Sahaana Suri. 2017. Macrobase: Prioritizing attention in fast data. In *SIGMOD.* 541–556.

[6] Favyen Bastani, Songtao He, Arjun Balasingam, Karthik Gopalakrishnan, Mohammad Alizadeh, Hari Balakrishnan, Michael Cafarella, Tim Kraska, and Sam Madden. 2020. MIRIS: Fast Object Track Queries in Video. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data.* 1907–1921.

[7] 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.

[8] George Bennett. 1962. Probability inequalities for the sum of independent random variables. *J. Amer. Statist. Assoc.* 57, 297 (1962), 33–45.

[9] Eric Breck, Neoklis Polyzotis, Sudip Roy, Steven Whang, and Martin Zinkevich. 2019. Data Validation for Machine Learning.. In *MLSys.*

[10] Clemens-Alexander Brust, Christoph Käding, and Joachim Denzler. 2018. Active learning for deep object detection. *arXiv preprint arXiv:1809.09875* (2018).

[11] Cristian Buciluă, Rich Caruana, and Alexandru Niculescu-Mizil. 2006. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining.* 535–541.

[12] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. 2020. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition.* 11621–11631.

[13] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *International conference on machine learning.* PMLR, 1597–1607.

[14] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. 2020. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297* (2020).
[15] Jack Choquette and Wish Gandhi. 2020. Nvidia A100 GPU: Performance & innovation for GPU computing. In *2020 IEEE Hot Chips 32 Symposium (HCS)*. IEEE Computer Society, 1–43.
[16] Brian Dean. 2021. Tesla Revenue and Production Statistics for 2021. https://backlinko.com/tesla-stats. (2021).
[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
[18] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).
[19] Ruediger Ehlers. 2017. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*. Springer, 269–286.
[20] Amit Gupta. 2019. Sklearn Linear Regression Tutorial with Boston House Dataset. (2019). https://amitg0161.medium.com/sklearn-linear-regression-tutorial-with-boston-house-dataset-cde74afd460a
[21] David Harrison Jr and Daniel L Rubinfeld. 1978. Hedonic housing prices and the demand for clean air. *Journal of environmental economics and management* 5, 1 (1978), 81–102.
[22] Andrew Hawkins. 2019. Serious safety lapses led to Uber's fatal self-driving crash, new documents suggest. https://www.theverge.com/2019/11/6/20951385/uber-self-driving-crash-death-reason-ntsb-dcouments. (2019).
[23] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. 2021. Masked autoencoders are scalable vision learners. *arXiv preprint arXiv:2111.06377* (2021).
[24] Daniel Kang, Nikos Arechiga, , Sudeep Pillai, Peter Bailis, and Matei Zaharia. 2022. Finding Label and Model Errors in Perception Data With Learned Observation Assertions. *SIGMOD* (2022).
[25] Daniel Kang, Peter Bailis, and Matei Zaharia. 2019. BlazeIt: Optimizing Declarative Aggregation and Limit Queries for Neural Network-Based Video Analytics. *PVLDB* (2019).
[26] Daniel Kang, Alex Derhacobian, Kaoru Tsuji, Trevor Hebert, Peter Bailis, Tadashi Fukami, Tatsunori Hashimoto, Yi Sun, and Matei Zaharia. 2021. Exploiting Proximity Search and Easy Examples to Select Rare Events. *NeurIPS DCAI workshop* (2021).
[27] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: optimizing neural network queries over video at scale. *PVLDB* 10, 11 (2017), 1586–1597.
[28] Daniel Kang, Edward Gan, Peter Bailis, Tatsunori Hashimoto, and Matei Zaharia. 2020. Approximate Selection with Guarantees using Proxies. *PVLDB* (2020).
[29] Daniel Kang, John Guibas, Peter Bailis, Tatsunori Hashimoto, Yi Sun, and Matei Zaharia. 2021. Accelerating Approximate Aggregation Queries with Expensive Predicates. *PVLDB* (2021).
[30] Daniel Kang, John Guibas, Peter Bailis, Tatsunori Hashimoto, and Matei Zaharia. 2022. Semantic Indexes for Machine Learning-based Queries over Unstructured Data. *SIGMOD* (2022).
[31] Daniel Kang, Ankit Mathur, Teja Veeramacheneni, Peter Bailis, and Matei Zaharia. 2021. Jointly Optimizing Preprocessing and Inference for DNN-based Visual Analytics. *PVLDB* (2021).
[32] Daniel Kang, Deepti Raghavan, Peter Bailis, and Matei Zaharia. 2020. Model assertions for monitoring and improving ML models. *MLSys* (2020).
[33] 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. Level 5 Perception Dataset 2020. https://level-5.global/level5/data/.
[34] 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/CVF Conference on Computer Vision and Pattern Recognition*. 12697–12705.
[35] David Lazer and Ryan Kennedy. 2015. What We Can Learn From the Epic Failure of Google Flu Trends. https://www.wired.com/2015/10/can-learn-epic-failure-google-flu-trends/. (2015).
[36] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *ECCV*. Springer, 740–755.
[37] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. Ssd: Single shot multibox detector. In *European conference on computer vision*. Springer, 21–37.
[38] Ramin Nabati and Hairong Qi. 2021. Centerfusion: Center-based radar and camera fusion for 3d object detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 1527–1536.
[39] Curtis G Northcutt, Anish Athalye, and Jonas Mueller. 2021. Pervasive label errors in test sets destabilize machine learning benchmarks. *arXiv preprint arXiv:2103.14749* (2021).
[40] Augustus Odena, Catherine Olsson, David Andersen, and Ian Goodfellow. 2019. Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. In *International Conference on Machine Learning*. PMLR, 4901–4911.

[41] Neoklis Polyzotis and Matei Zaharia. 2021. What can Data-Centric AI Learn from Data and ML Engineering? *arXiv preprint arXiv:2112.06439* (2021).

[42] Pranav Rajpurkar, Awni Y Hannun, Masoumeh Haghpanahi, Codie Bourn, and Andrew Y Ng. 2017. Cardiologist-level arrhythmia detection with convolutional neural networks. *arXiv preprint arXiv:1707.01836* (2017).

[43] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. 2020. Mlperf inference benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 446–459.

[44] Sean Robertson. 2022. NLP from Scratch: Translation with a Sequence to Sequence Network and Attention. (2022). https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

[45] Kexin Rong, Clara E Yoon, Karianne J Bergen, Hashem Elezabi, Peter Bailis, Philip Levis, and Gregory C Beroza. 2018. Locality-sensitive hashing for earthquake detection: A case study of scaling data-driven science. *PVLDB* (2018).

[46] Andrew I Schein and Lyle H Ungar. 2007. Active learning for logistic regression: an evaluation. *Machine Learning* 68, 3 (2007), 235–265.

[47] Burr Settles. 2009. Active learning literature survey. (2009).

[48] Vinay Shet. 2019. Lyft Level 5 Self-Driving Perception Dataset Competition Now Open. https://medium.com/wovenplanetlevel5/lyft-level-5-self-driving-dataset-competition-now-open-97493e9f154a. (2019).

[49] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. 2017. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*. 843–852.

[50] Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. 2018. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3733–3742.

[51] 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).

[52] Francis Y Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. 2020. Learning in situ: a randomized experiment in video streaming. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*. 495–511.

[53] 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).