MODEL ASSERTIONS FOR MONITORING AND IMPROVING ML MODELS

Daniel Kang [*1] **Deepti Raghavan** [*1] **Peter Bailis** [1] **Matei Zaharia** [1]

ABSTRACT

Machine learning models are increasingly deployed in mission-critical settings such as vehicles, but unfortunately, these models can fail in complex ways. To prevent errors, ML engineering teams monitor and continuously improve these models. We propose a new abstraction, *model assertions*, that adapts the classical use of program assertions as a way to monitor and improve ML models. Model assertions are arbitrary functions over the model's input and output that indicates when errors may be occurring. For example, a developer may write an assertion that an object's class should stay the same across frames of video. Once written, these assertions can be used both for runtime monitoring and for improving a model at training time. In particular, we show that at runtime, model assertions can find *high confidence* errors, where a model returns the wrong output with high confidence, which uncertainty-based monitoring techniques would not detect. We also propose two methods to use model assertions at training time. First, we propose a bandit-based active learning algorithm that can sample from data flagged by assertions and show that it can reduce labeling costs by up to 33% over traditional uncertainty-based methods. Second, we propose an API for generating "consistency assertions" (e.g., the class change example) and weak labels for inputs where the consistency assertions fail, and show that these weak labels can improve relative model quality by up to 46%. We evaluate both algorithms on four real-world tasks with video, LIDAR, and ECG data.

1 INTRODUCTION

Machine learning is increasingly deployed in complex mission-critical contexts from autonomous vehicles (AVs) to precision medicine. However, ML models can misbehave, with disastrous consequences. For example, AVs have accelerated toward highway lane dividers (Lee, 2018) and can rapidly change their classification of objects over time, causing erratic behavior (Coldewey, 2018). As a result, quality assurance (QA) of models, including continuous monitoring and improvement, is of paramount concern.

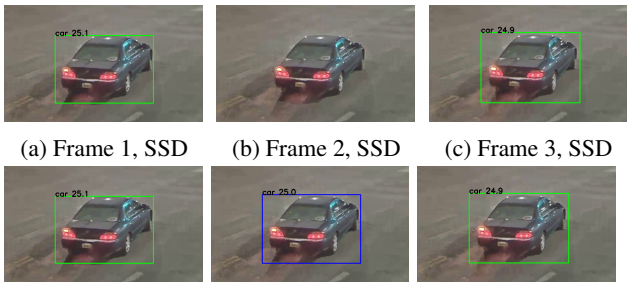
Unfortunately, performing QA for complex, real-world ML applications is challenging. Existing solutions that focus on verifying training, including formal verification (Katz et al., 2017), whitebox testing (Pei et al., 2017), monitoring training metrics (Renggli et al., 2019), and validating training code (Odena & Goodfellow, 2018), *only give guarantees on a test set and perturbations thereof*, so models can still fail on the huge volumes of deployment data that are not part of the test set (e.g., billions of images per day in an AV fleet). Validating input schemas (Polyzotis et al., 2019; Baylor et al., 2017) does not work for applications with unstructured inputs that *lack meaningful schemas*, e.g., images. Solutions that check whether model performance remains consistent over time (Baylor et al., 2017) only apply to deployments that have ground truth labels, e.g., predicting ad click-through rates, but not to deployments that *lack labels*.

As a step towards more robust QA for complex ML applications, we have found that many ML developers can often specify *systematic* errors made by ML models. Specifically, certain classes of errors are repetitive and can be checked automatically, via code. For example, a researcher developing a video analytics engine noticed that object detection models can identify boxes of cars that flicker rapidly in and out of the video (Figure 1), indicating some of the detections are likely wrong. Likewise, our contacts at an AV company reported that LIDAR and camera models sometimes disagree. These systematic errors can arise for diverse reasons, including domain shift between training and deployment data (e.g., still images vs. video), incomplete training data (e.g., no instances of snow-covered cars), and noisy inputs.

To leverage the systematic nature of these errors, we propose *model assertions*, a simple abstraction to monitor and improve ML model quality. Model assertions are inspired by program assertions (Goldstine et al., 1947; Turing, 1949), one of the most common ways to monitor software. A model assertion is an arbitrary function over the model's input and output that returns a Boolean or continuous severity score to indicate when faults may be occurring. For example, a model assertion that checks whether an object flickers in

[*]Equal contribution [1]Stanford University. Correspondence to: Daniel Kang <ddkang@stanford.edu>.

Proceedings of the 2^{nd} SysML Conference, Palo Alto, CA, USA, 2019.



(a) Frame 1, SSD (b) Frame 2, SSD (c) Frame 3, SSD

(d) Frame 1, SSD (e) Frame 2, corrected from assertion (f) Frame 3, SSD

Figure 1. **Top row:** example of flickering in three consecutive frames of a video. The object detection method, SSD (Liu et al., 2016), failed to identify the car in the second frame. **Bottom row:** example of correcting the output of a model. The car bounding box in the second frame can be inferred using nearby frames based on a consistency assertion. Best viewed in color.

and out of video could return a Boolean value over each frame or the number of objects that flicker. While assertions may not be a complete specification of correctness, we have found that similar assertions are easy to specify in many domains (§2).

We explore several ways to use model assertions, both at runtime and training time:

- At runtime, we show that assertions can be used for **monitoring** to log unexpected behavior or automatically trigger corrective actions, e.g., shutting down an autopilot.
- At training time, we show that assertions can augment **active learning**. We propose a new *bandit-based active learning algorithm (BAL)* that can leverage the output of multiple assertions in addition to standard uncertainty methods to select data for humans to label, increasing model quality for the same number of labels.
- We show that model assertions can be used for **weak supervision**. In particular, we propose an API for writing *consistency assertions* about model outputs that can also impute weak labels for outputs that fail the assertion, improving model quality with no new human labels.

Our first application of model assertions is runtime monitoring. We show that simple model assertions can often find *high confidence* errors, where the model has high certainty in an erroneous output. Prior uncertainty-based monitoring would not flag these errors.

Our second application is active learning. Traditional active learning algorithms select data to label based on uncertainty, with the intuition that "harder" data where the model is uncertain will be more informative (Settles, 2009). Model assertions provide another natural way to find "hard" examples. However, using assertions in active learning presents a challenge: how should the active learning algorithm select between data when several assertions are used? Data can be simultaneously flagged by multiple assertions or a single assertion can flag multiple data points, in contrast to a single uncertainty metric. To address this challenge, we present a novel *bandit-based active learning algorithm (BAL)*. Given a set of data that have been flagged by potentially multiple model assertions, our bandit algorithm uses the assertions' severity scores as context (i.e., features) and maximizes the marginal reduction in the number of assertions fired (§3). We show that our bandit algorithm can reduce labeling costs by up to 33% over traditional uncertainty-based methods.

Our third application is weak supervision (Mintz et al., 2009; Ratner et al., 2017). We propose an API for writing *consistency assertions* about how attributes of a model's output should relate that can also provide weak labels for training. Consistency assertions specify that data should be consistent between attributes and identifiers, e.g., that a TV news host (identifier) should have consistent gender (attribute) in a single show, or that certain predictions should (or should not) exist in temporally related outputs, e.g., cars in adjacent video frames (Figure 1). We demonstrate that this API can apply to a range of domains, including medical classification and TV news analytics. These weak labels can be used to improve relative model quality by up to 46% with no additional human labeling.

We implement model assertions in a Python library, LIBMA, that can be used with existing ML frameworks. We evaluate assertions on four ML applications: understanding TV news, AVs, video analytics, and classifying medical readings. We implement assertions for systematic errors reported by ML users in these domains, including checking for consistency between sensors, domain knowledge about object locations in videos, and medical knowledge about heart patterns. Across these domains, we find that model assertions we consider can be written with at most 60 lines of code and with 88-100% precision, that these assertions often find high-confidence errors (e.g., top 90th percentile by confidence), and that our new algorithms for active learning and weak supervision via assertions improve model quality over existing methods.

In summary, we make the following contributions:

1. We introduce the abstraction of model assertions for monitoring and continuously improving ML models.
2. We show that model assertions can find high confidence errors, which would not be flagged by uncertainty metrics.
3. We propose a bandit algorithm to select data points for active learning via model assertions and show that it can reduce labeling costs by up to 33%.
4. We propose an API for consistency assertions that also automatically geneate weak labels for data where the assertion fails, and show that weak supervision via these labels can improve relative model quality by up to 46%.

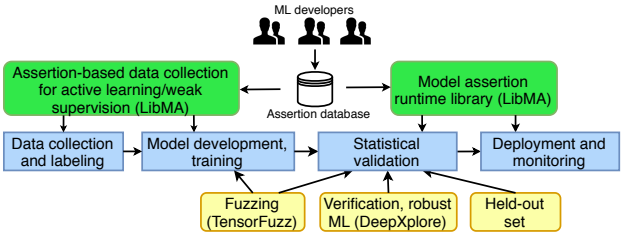


Figure 2. A system diagram of how model assertions can integrate into the ML development/deployment pipeline. Users can collaboratively add to an assertion database. We also show how related work can be integrated into the pipeline. Notably, verification only gives guarantees on a test set and perturbations thereof, but not on arbitrary runtime data.

2 Model Assertions

In this section, we introduce model assertions, describe how model assertions integrate into the ML development/deployment cycle, and describe several domains where model assertions apply.

2.1 Model Assertions Interface

In this section, we formalize the model assertions interface. Model assertions are arbitrary functions that can indicate when an error is likely to have occurred. They take as input a list of inputs and outputs from one or more ML models. They return *severity score*, a continuous value that indicates the severity of an error of a specific type. By convention, the `0` value represents an abstention. Boolean values can be implemented in model assertions by only returning `0` and `1`. The severity score does not need to be calibrated, as our algorithms only uses the relative ordering of scores.

As a concrete example, consider an AV with a LIDAR sensor and camera and object detection models for each sensor. To check that these models agree, a developer may write:

```
def sensor_agreement(lidar_boxes, camera_boxes):
  failures = 0
  for lidar_box in lidar_boxes:
    if no_overlap(lidar_box, camera_boxes):
      failures += 1
  return failures
```

Notably, LIBMA takes arbitrary Python functions.

2.2 Using Model Assertions for QA

We describe how model assertions can be integrated with ML development and deployment pipelines. Importantly, model assertions are complementary to a range of other ML QA techniques, including verification, fuzzing, and statistical techniques, as shown in Figure 2.

First, model assertions can be used for monitoring and validating all parts of the ML development/deployment pipeline. Namely, model assertions are agnostic to the source of the output, whether it be an ML model or a human labeler. Concretely, model assertions can be used to validate human labels (data collection) or historical data (validation), and to monitor deployments (e.g., to populate dashboards).

Second, model assertions can be used at training time to select which data points to label in active learning. We describe BAL, our algorithm for data selection, in §3.

Third, model assertions can be used to generate weak labels to further train ML models without additional human labels. We describe how LIBMA accomplishes this via consistency assertions in §4. Users can also register their own weak supervision rules.

2.3 Example Use Cases and Assertions

In this section, we provide use cases for model assertions that arose in discussions with industry and academic contacts, including AV companies and academic labs. We show example of errors caught by the model assertions described in this section in Appendix A and describe how one might look for assertions in other domains in Appendix B.

Our discussions revealed two key properties in real-world ML systems. First, ML models are deployed on orders of magnitude more data than can reasonably be labeled, so a labeled sample cannot capture all deployment conditions. For example, the fleet of Tesla vehicles will see over 100× more images in a day than in the largest existing image dataset (Sun et al., 2017). Second, ML deployments are complex and developed by teams, so the domain experts who monitor ML models may not know how to fix errors in deployment. As a result, it is critical for different teams to be able to collaborate in monitoring these deployments.

Analyzing TV news. We spoke to a research lab that studies bias in media via automatic analysis. This lab has collected over 10 years of TV news (billions of frames) and executed face detection every three seconds. These detections are subsequently used to identify the faces, detect gender, and classify hair color using ML models. Currently, the researchers have no method of identifying errors and manually inspect data. However, they additionally compute scene cuts. Given that most TV new hosts do not move much between scenes, we can assert that the identity, gender, and hair color of faces that highly overlap within the same scene are consistent (Figure 6, Appendix). We further describe how model assertions can be implemented via our consistency API for TV news in §4.

Autonomous vehicles (AVs). AVs are required to execute a variety of tasks, including detecting objects and tracking lane markings. These tasks are accomplished with ML models from different sensors, such as visual, LIDAR, or ultrasound sensors (Davies, 2018). For example, a vision

model might be used to detect objects in video and a point cloud model might be used to do 3D object detection.

Our contacts at an AV company noticed that models from video and point clouds sometimes disagree. We implemented a model assertion that projects the 3D boxes onto the 2D plane of the camera to check for consistency. If the assertion triggers, then at least one of the sensors returned an incorrect answer.

Video analytics. Many modern, academic video analytics systems use an object detection method (Kang et al., 2017; 2018; Hsieh et al., 2018; Jiang et al., 2018; Xu et al., 2019; Canel et al., 2019) trained on MS-COCO (Lin et al., 2014), which is a corpus of still images. These still image object detection methods are then deployed on video for detecting objects. Currently, none of these systems aim to detect errors in object detection.

Researchers developing such a system noticed that objects flickered in and out of the video (Figure 1) and that vehicles overlapped in unrealistic ways (Figure 7, Appendix). We implemented assertions to detect these.

Medical classification. Deep learning researchers have created deep networks that can outperform cardiologists for classifying atrial fibrillation (AF, a form of heart condition) from single-lead ECG data (Rajpurkar et al., 2019). Our researcher contacts mentioned that AF predictions can rapidly oscillate. The European Society of Cardiology guidelines for detecting AF require at least 30 seconds of signal before calling a detection (EHRA, 2010). Thus, predictions should not rapidly switch between two states. A doctor could specify this model assertion, which could be implemented to monitor ECG classification deployments.

2.4 Implementing Model Assertions in LIBMA

We implement a prototype library for model assertions, LIBMA, that works with existing Python ML training and deployment frameworks. Its primary functions are to apply assertions for monitoring and select data for active learning and weak supervision. We briefly describe LIBMA's implementation.

LIBMA logs user-defined assertions in an assertion database. The simplest way to add an assertion is through `AddAssertion(func)`, where `func` is a function of the inputs and outputs (see below). LIBMA also provides an API to add consistency assertions as described in §4. Given this database, LIBMA requires a callback after model execution that takes the model's input and output as input. Given the model's input and output, LIBMA will execute the assertions and record any errors. We assume the assertion signature is similar to the following; this assertion signature is for the example in Figure 1:

```
def flickering(recent_frames: List[PixelBuf],
  recent_outputs: List[BoundingBox]) -> Float
```

For active learning, LIBMA will take a batch of data and return indices for which data points to label. For weak supervision, LIBMA will take data and return weak labels where valid. Users can specify weak labeling functions associated with assertions to help with this.

In the following two sections, we describe two key methods that LIBMA uses to improve model quality: BAL for active learning and consistency assertions for weak supervision.

3 USING MODEL ASSERTIONS FOR ACTIVE LEARNING WITH BAL

We introduce an algorithm called BAL to select data for active learning via model assertions. BAL assumes that a set of data points has been collected and a subset will be labeled in bulk. We found that labeling services (sca, 2019) and our industrial contacts usually label data in bulk.

Given a set of data points that triggered model assertions, a key challenge is selecting which points to label. There are two key challenges which makes data selection intractable in its full generality. First, we do not know the marginal utility of selecting a data point to label without labeling the data point. Second, even with labels, estimating the marginal gain of data points is expensive to compute as training modern ML models is expensive.

To address these issues, we make two simplifying assumptions. We describe the statistical model we assume, the resource-unconstrained algorithm, and our simplifying assumptions and BAL.

Data selection as multi-armed bandits. We cast the data selection problem as a multi-armed bandit (MAB) problem (Auer et al., 2002; Berry & Fristedt, 1985). In MABs, a set of "arms" (i.e., individual data points) is provided and the user must select a set of arms (i.e., points to label) to achieve the maximal expected utility (e.g., maximize validation accuracy, minimize number of assertions that fire). MABs have been studied in a wide variety of contexts (Radlinski et al., 2008; Lu et al., 2010; Bubeck et al., 2009), but we assume that the arms have context associated with them (i.e., severity scores from model assertions), give submodular rewards (defined below), and are possibly time-varying. The following presentation is inspired by (Chen et al., 2018).

Concretely, we assume the data will be labeled in T rounds and denote the rounds $t = 1, ..., T$. We refer to the set of n data points as $N = \{1, ..., n\}$. Each data point has a d dimensional feature vector associated with it, where d is the number of model assertions. We refer to the feature vector as x_i^t, where i is the data point index and t is the round

index; from here, we will refer to the data points as x_i^t. Each entry in a feature vector is the severity score from a model assertion. The feature vectors can change over time as the model predictions, and therefore assertions, change over the course of training.

We assume there is a budget on the number of arms (i.e., data points to label), B^t, at every round and the user must select a set of arms $S^t = \{x_{s_1}, ..., x_{s_{B^t}}\}$ such that $|S^t| \leq B^t$. We further assume that the reward from the arms, $R(S^t)$, is submodular in S^t. Intuitively, submodularity implies diminishing marginal returns: adding the 100th data point will not improve the reward as much as adding the 10th data point. Formally, we first define the marginal gain of adding an extra arm:

$$\Delta R(\{m\}, A) = R(A \cup \{m\}) - R(A). \quad (1)$$

where $A \subset N$ is a subset of arms and $m \in N$ is an additional arm such that $m \notin A$. The submodularity condition states that, for any $A \subset C \subset N$ and $m \notin C$, that

$$\Delta R(\{m\}, A) \geq \Delta R(\{m\}, C). \quad (2)$$

Resource-unconstrained algorithm. Assuming an infinite labeling and computational budget, we describe an algorithm that selects data points to train on. Unfortunately, this algorithm is not feasible as it requires labels for every point and training the ML model many times. We describe a bandit algorithm under resource constraints below.

If we assume that rewards for individual arms can be queried, then a recent bandit algorithm, CC-MAB (Chen et al., 2018) can achieve a regret of $O(cT^{\frac{2\alpha d}{3\alpha d}} \log(T))$. Here, α is an (unknown) smoothness parameter that determines the similarity between arms of similar contexts (formally, the α in the Hölder condition (Evans, 1998)). A regret bound is the (asymptotic) difference with respect to an oracle algorithm. Briefly, CC-MAB explores under-explored arms until it is confident that certain arms have highest reward. Then, it greedily takes the highest reward arms. Full details are given in (Chen et al., 2018) and summarized in Algorithm 1.

Unfortunately, CC-MAB requires access to an estimate of selecting a single arm. Estimating the gain of a single arm requires a label and requires retraining and reevaluating the model, which is computationally infeasible for expensive-to-train ML models, especially modern deep networks.

Resource-constrained algorithm. We make two simplifying assumptions and use these to modify CC-MAB for the resource-constrained setting. Our two simplifying assumptions are that 1) data points with similar contexts (i.e., x_i^t) are interchangeable and 2) data points with higher severity scores have higher expected marginal gain. We further assume that the goal is to reduce the number of data points that trigger assertions as much as possible.

Input: T, B^t, N, R
Output: choice of arms S^t at rounds $1, ..., T$
for $t = 1, ..., T$ **do**
  if *Underexplored arms* **then**
    Select arms S^t from under-explored contexts at random
  else
    Select arms S^t by highest marginal gain (Eq. 1):
    for $i = 1, ..., B^t$ **do**
      $S_i^t = \arg\max_{j \in N \setminus S_{i-1}^t} \Delta R(\{j\}, S_{i-1}^t)$
    end
  end
end

Algorithm 1: A summary of the CC-MAB algorithm. CC-MAB first explores under-explored arms, then greedily selects arms with highest marginal gain. Full details are given in (Chen et al., 2018).

Under these assumptions, we do not require an estimate of the marginal reward for each arm. Instead, we can approximate the marginal gain from selecting arms with similar contexts by the total number of these arms that were selected. This has two benefits. First, we can train a model on a set of arms (i.e., data points) in batches instead of adding single arms at a time. Second, we can select data points of similar contexts at random, without having to compute its marginal gain.

Input: T, B^t, N, R
Output: choice of arms S^t at rounds $1, ..., T$
for $t = 1, ..., T$ **do**
  if *t = 0* **then**
    Select data points uniformly at random from the d model assertions
  else
    Compute the marginal reduction r_m of the number of times model assertion $m = 1, ..., d$ triggered;
    if *all* $r_m < 1\%$ **then**
      Fall back to baseline method;
      continue;
    end
    for $i = 1, ..., B^t$ **do**
      Select model assertion m proportional to r_m;
      Select x_i that triggers m, sample proportional to severity score rank;
      Add x_i to S^t;
    end
  end
end

Algorithm 2: BAL algorithm for data selection for continuous training. BAL samples from the assertions at random in the first round, then selects the assertions that result in highest marginal reduction in the number of assertions that fire in subsequent rounds. BAL will default to random sampling or uncertainty sampling if none of the assertions reduce.

Leveraging these two assumptions, we can simplify Algorithm 1 to require less computation for training models and to not require labels for all data points. Our algorithm is described in Algorithm 2. Briefly, we approximate the marginal gain of selecting batches of arms and select arms proportional to the marginal gain. We additionally allocate 25% of the budget in each round to randomly sample arms that triggered different model assertions, uniformly. This ensures that no contexts (i.e., model assertions) are underexplored as training progresses. Finally, in some cases (e.g., with noisy assertions), it may not be possible to reduce the number of assertions that fire. In this case, BAL will default to random sampling or uncertainty sampling, as specified by the user.

4 Consistency Assertions and Weak Supervision

Although developers can write arbitrary Python functions as model assertions in LIBMA, we found that many assertions can be specified using an even simpler, high-level abstraction that we called *consistency assertions*. This interface allows LIBMA to generate multiple Boolean model assertions from a high-level description of the model's output, as well as automatic *correction rules* that propose new labels for data that fail the assertion to enable weak supervision.

The key idea of consistency assertions is to specify which attributes of a model's output are expected to match across many invocations to the model. For example, consider a TV news application that tries to locate faces in TV footage and then identify their name and gender (one of the real-world applications we discussed in §2.3). The ML developer may wish to assert that, within each video, each person should consistently be assigned the same gender, and should appear on the screen at similar positions on most nearby frames. Consistency assertions let developers specify such requirements by providing two functions:

- An *identification function* that returns an identifier for each model output. For example, in our TV application, this could be the person's name as identified by the model.
- An *attributes function* that returns a list of named attributes expected to be consistent for each identifier. In our example, this could return the gender attribute.

Given these two functions, LIBMA generates multiple Boolean assertions that check whether the various attributes of outputs with a common identifier match. In addition, it generates correction rules that can replace an inconsistent attribute with a guess at that attribute's value based on other instances of the identifier (we simply use the most common value). By running the model and these generated assertions over unlabeled data, LIBMA can thus *automatically* generate weak labels for data points that do not satisfy the consistency assertions. We show in §5 that these weak labels can automatically increase model quality.

4.1 API Details

The consistency assertions API supports ML applications that run over multiple inputs x_i and produce zero or more outputs $y_{i,j}$ for each input. For example, each output could be an object detected in a video frame. The user provides two functions over outputs $y_{i,j}$:

- $\mathrm{Id}(y_{i,j})$ returns an *identifier* for the output $y_{i,j}$, which is simply an opaque value.
- $\mathrm{Attrs}(y_{i,j})$ returns zero or more *attributes* for the output $y_{i,j}$, which are key-value pairs.

In addition to checking attributes, we found that many applications also expect their identifiers to appear in a "temporally consistent" fashion, where objects do not disappear and reappear too quickly. For example, one would expect cars identified in the video to stay on the screen for multiple frames instead of "flickering" in and out in most cases. To express this expectation, developers can provide a *temporal consistency threshold*, T, which specifies that each identifier should not appear or disappear for intervals less than T seconds. For example, we might set T to one second for TV footage that frequently cuts across frames, or 30 seconds for an activity classification algorithm that distinguishes between walking and biking. The full API for adding a consistency assertion is therefore $\mathrm{AddConsistencyAssertion}(\mathrm{Id}, \mathrm{Attrs}, T)$.

Examples. We briefly describe how one can use consistency assertions in several ML tasks motivated in §2.3:

Face identification in TV footage: This application uses multiple ML models to detect faces in images, match them to identities, classify their gender, and classifier their hair color. We can use the detected identity as our Id function and gender/hair color as attributes.

Video analytics for traffic cameras: This application aims to detect vehicles in video street traffic, and suffers from problems such as flickering or changing classifications for an object. The model's output is bounding boxes with classes on each frame. Because we lack a globally unique identifier (e.g., license plate number) for each object, we can assign an new identifier for each box that appears and assign the same identifier as it persists through the video. We can treat the class as an attribute and set T as well to detect flickering.

Heart rhythm classification from ECGs: In this application, domain experts informed us that atrial fibrillation heart rhythms need to persist for at least 30 seconds to be considered a problem. We used the detected class as our identifier and set T to 30 seconds.

4.2 Generating Assertions and Labels from the API

Given the Id, Attrs, and T values, LIBMA automatically generates Boolean assertions to check for matching attributes and to check that when an identifier appears in the data, it persists for at least T seconds. These assertions are treated the same as user-provided ones in the rest of the system.

LIBMA also automatically generates corrective rules that propose a new label for outputs that do not match their identifier's other outputs on an attribute. The default behavior is to propose the most common value of that attribute (e.g., the class detected for an object on most frames), but users can also provide a WeakLabel function to suggest an alternative based on all of that object's outputs.

For temporal consistency constraints via T, LIBMA will assert that at most one transitions can occur within a T-second window. For example, an identifier appearing is valid, but an identifier appearing, disappearing, then appearing is invalid. Users can override the default of at most one transition occurring. If a violation occurs, LIBMA will propose to remove or modify it. In the latter case, LIBMA needs to know how to generate an expected output on an input where the object was not identified (e.g., frames where the object flickered out in Figure 1). LIBMA requires the user to provide a WeakLabel function to cover this case, since it may require domain specific logic, e.g., averaging the locations of the object on nearby video frames.

5 EVALUATION

5.1 Experimental Setup

We evaluated LIBMA and model assertions on four diverse ML workloads based on real industrial and academic use-cases: analyzing TV news, video analytics, autonomous vehicles, and medical classification. For each domain, we describe the task, dataset, model, training procedure, and assertions. A summary is given in Table 1.

TV news. Our contacts analyzing TV news provided us 50 hour-long segments that were known to be problematic. They further provided pre-computed boxes of faces, identities, and hair colors; this data was computed from a range of models and sources, including hand-labeling, weak labels, and custom classifiers. We implemented the consistency assertions described in §4. We were unable to access the training code for this domain so were unable to perform retraining experiments for this domain.

Video analytics. Many modern video analytics systems use object detection as a core primitive (Kang et al., 2017; 2018; Hsieh et al., 2018; Jiang et al., 2018; Xu et al., 2019; Canel et al., 2019), in which the task is to localize and classify the objects in a frame of video. We focus on the object detection portion of these systems. We used ResNet-34 SSD (Liu et al., 2016) (henceforth SSD) model pretrained on MS-COCO (Lin et al., 2014). We deployed SSD for detecting vehicles in the `night-street` video that is commonly used (also referred to as `jackson`) (Kang et al., 2017; Xu et al., 2019; Canel et al., 2019; Hsieh et al., 2018). We used a separate day of video for training and testing.

We deployed three model assertions: `multibox`, `flicker`, and `appear`. The `multibox` assertion fires when three boxes highly overlap (Figure 7, Appendix). The `flicker` and `appear` assertions are implemented with our consistency API as described in §4.

Autonomous vehicles. We studied the problem of object detection for autonomous vehicles using the NuScenes dataset (Caesar et al., 2019), which contains labeled LIDAR point clouds and associated visual images. We split the data into separate train, unlabeled, and test splits. We detected vehicles only. We use the open-source Second model with PointPillars (Yan et al., 2018; Lang et al., 2019) for LIDAR detections and SSD for visual detections. We improve SSD via active learning and weak supervision in our experiments.

As NuScenes contains time-aligned point clouds and images, we deployed a custom assertion for 2D and 3D boxes agreeing, and the `multibox` assertion. We deployed a custom weak supervision rule that imputed boxes from the 3D predictions. While other assertions could have been deployed (e.g., `flicker`), we found that the dataset was not sampled frequently enough (at 2 Hz) for these assertions.

Medical classification. We studied the problem of classifying atrial fibrillation (AF) via ECG signals. We used a convolutional network that was shown to outperform cardiologists (Rajpurkar et al., 2019). Unfortunately, the full dataset used in (Rajpurkar et al., 2019) is not publicly available, so we used the CINC17 dataset (cin, 2017). CINC17 contains 8,528 data points that we split into train, validation, unlabeled, and test splits.

We consulted with medical researchers and deployed an assertion that asserts that the classification should not change between two classes in under a 30 second time period (i.e., the assertion fires when the classification changes from $A \rightarrow B \rightarrow A$ within 30 seconds), as described in §4.

5.2 Model Assertions can be Written with High Precision with Few LOC

We first asked whether model assertions could be written succinctly. To test this, we implemented the model assertions described above and counted the lines of code (LOC) necessary for each assertion. We count the LOC for the identity and attribute functions for the consistency asser-

Task	Model	Assertions
TV news	Custom	Consistency (§4, `news`)
Object detection (video)	SSD (Liu et al., 2016)	Three vehicles should not highly overlap (`multibox`), identity consistency assertions (`flicker` and `appear`)
Vehicle detection (AVs)	Second (Yan et al., 2018), SSD	Agreement of Point cloud and image detections (`agree`), `multibox`
AF classification	ResNet (Rajpurkar et al., 2019)	Consistency assertion within a 30s time window (`ECG`)

Table 1. A summary of tasks, models, and assertions used in our evaluation.

Assertion	LOC (no helpers)	LOC (inc. helpers)
`news`	7	39
`ECG`	23	50
`flicker`	18	60
`appear`	18	35
`multibox`	14	28
`agree`	11	28

Table 2. Number of lines of code (LOC) for each assertion. Consistency assertions are on the top and custom assertions are on the bottom. All assertions could be written in under 60 LOC including helper functions, when double counting between assertions. The assertion main body could be written in under 25 LOC in all cases. The helper functions included utilities such as computing the overlap between boxes.

Assertion	Precision (identifier and output)	Precision (model output only)
`news`	100%	100%
`ECG`	100%	100%
`flicker`	100%	96%
`appear`	100%	88%
`multibox`	N/A	100%
`agree`	N/A	98%

Table 3. Precision of our model assertions we deployed on 50 randomly selected examples. The top are consistency assertions and the bottom are custom assertions. We report both precision in the ML model outputs only and when counting errors in the identification function and ML model outputs for consistency assertions. As shown, model assertions can be written with 88-100% precision across all domains when only counting errors in the model outputs.

tions (see Table 1 for a summary of assertions). We counted the LOC with and without the shared helper functions (e.g., computing box overlap); we double counted the helper functions when used between assertions. As we show in Table 2, both consistency and domain-specific assertions can be written in under 25 LOC excluding shared helper functions and under 60 LOC when including helper functions. Thus, model assertions can be written with few LOC.

We then asked if model assertions could be written with high precision. To test this, we randomly sampled 50 data points that triggered each assertion and manually checked whether that data point had an incorrect output from the ML model. The consistency assertions return clusters of data points (e.g., `appear`) and we report the precision for errors in both the identifier and ML model outputs and only the ML model outputs. As we show in Table 3, model assertions can be written with at least 88% precision in all cases. Thus, we find that model assertions can be written with high precision.

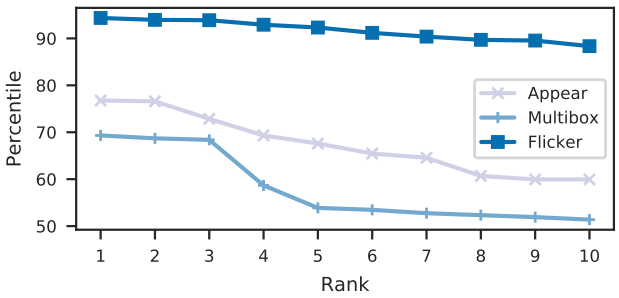


Figure 3. Percentile of confidence of the top-10 ranked errors by confidence found by LIBMA for video analytics. The x-axis is the rank of the errors caught by model assertions, ordered by rank. The y-axis is the percentile of confidence among all the boxes. As shown, model assertions can find errors where the original model has high confidence (94th percentile), allowing them to complement existing confidence-based methods for data selection.

5.3 Model Assertions can Identify Errors

Model assertions can identify high-confidence errors. We asked whether model assertions can identify high-confidence errors, or errors where the model returns the wrong output with high confidence. High-confidence errors are important to identify as confidence is used in downstream tasks, such as analytics queries and actuation decisions (Kang et al., 2017; 2018; Hsieh et al., 2018; Chinchali et al., 2019). Furthermore, sampling solutions that are based on confidence would be unable to identify these errors.

To determine if model assertions could identify high confidence errors, we collected the 10 data points with highest confidence error for each of the model assertions deployed for video analytics. We then plotted the percentile of the confidence among all the boxes for each error.

As shown in Figure 3, model assertions can identify errors within the top 94th percentile of boxes by confidence (the `flicker` confidences were from the average of the surrounding boxes). Importantly, uncertainty-based methods of monitoring would not catch these errors.

Model assertions can identify errors in human labels. We further asked whether model assertions could be used to identify errors in human-generated labels, i.e., a human is acting as a "ML model." While verification of human labels has been studied in the context of crowd-sourcing (Hirth et al., 2013; Tran-Thanh et al., 2013), several production

Description	Number
All labels	469
Errors	32
Errors caught	4

Table 4. Number of labels, errors, and errors caught from model assertions for Scale-annotated images for the video analytics task. As shown, model assertions caught 12.5% of the errors in this data.

labeling services (e.g., Scale (sca, 2019)) do not provide annotator identification which is necessary to perform this verification. We deployed a model assertion in which we tracked objects across frames of a video using an automated method and verified that the same object in different frames had the same label.

We obtained labels for 1,000 random frames from `night-street` from Scale AI (sca, 2019), which is used by several autonomous vehicle companies. Table 4 summarizes our results. Scale returned 469 boxes, which we manually verified for correctness. There were no localization errors, but there were 32 classification errors, of which the model assertion caught 12.5%. Thus, we see that model assertions can also be used to verify human labels.

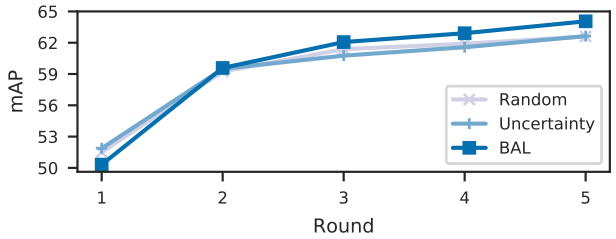
5.4 Model Assertions can Improve Model Quality via Active Learning

We evaluated LIBMA's active learning capabilities and BAL using the three domains which we had access to the training code (visual analytics, ECG, AVs).

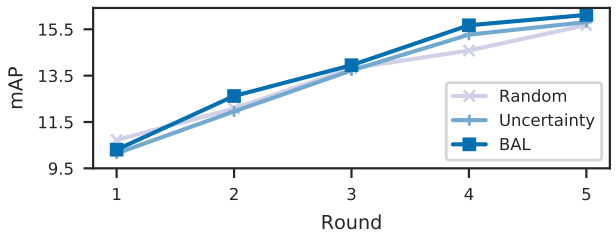
Multiple model assertions. We asked whether multiple model assertions could be used to improve model quality via continuous data collection. We deployed three assertions over `night-street` and two assertions for NuScenes. We used random sampling, uncertainty sampling with least confident (Settles, 2009), and BAL for the active learning strategies. We used the mAP metric for both datasets, which is widely used for object detection (Lin et al., 2014; He et al., 2017). We defer hyperparmeters to Appendix C.

As we show in Figure 4, BAL outperforms both random sampling and uncertainty sampling on both datasets after the first round, which is required for calibration. Prior work has shown that uncertainty sampling can be unsuited for deep networks (Sener & Savarese, 2017). For `night-street`, at a fixed accuracy threshold of 62%, BAL uses 33% fewer labels than random and uncertainty sampling. By the fifth round, BAL outperforms both random sampling and uncertainty sampling by 1.5% mAP. While the absolute change in mAP may seem small, doubling the model depth, which doubles the computational budget, on MS-COCO achieves a 1.7% improvement in mAP (ResNet-50 FPN vs. ResNet-101 FPN) (Girshick et al., 2018).

Single model assertion. Due to the limited data quantities for the ECG dataset, we were unable to deploy more than one assertion. Nonetheless, we further asked whether a single model assertion could be used to improve model quality. We ran five rounds of data labeling with 100 examples each round for ECG datasets. We ran the experiment 8 times and report averages. We show results in Figure 5. As shown, data collection with a single model assertion generally matches or outperforms both uncertainty and random sampling.



(a) Active learning for `night-street`.



(b) Active learning for NuScenes.

Figure 4. Performance of random sampling, uncertainty sampling, and BAL for active learning. The round is the round of data collection (see §3). As shown, BAL improves accuracy on unseen data and can achieve the same accuracy (62% mAP) as random sampling with 33% fewer labels for `night-street`. BAL also outperforms both baselines for the NuScenes dataset.

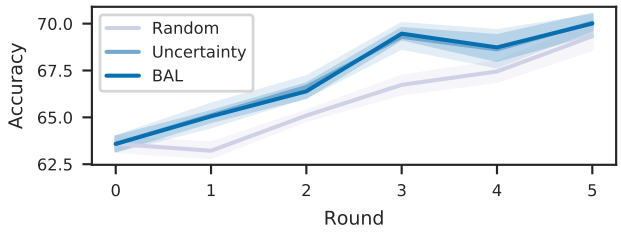


Figure 5. Active learning results with a single assertion for the ECG dataset. As shown, with just a single assertion, model-assertion based active learning can match uncertainty sampling and outperform random sampling.

5.5 Model Assertions can Improve Model Quality via Weak Supervision

We used our consistency assertions to evaluate the impact of weak supervision using assertions for the domains we had weak labels for (video analytics, AVs, and ECG). We describe hyperparameters in Appendix C.

Table 5 shows that model assertion-based weak supervision

Domain	Pretrained	Weakly supervised
Video analytics (mAP)	34.4	49.9
AVs (mAP)	10.6	14.1
ECG (% accuracy)	70.7	72.1

Table 5. Accuracy of the pretrained and weakly supervised models for video analytics, AV and ECG domains. Weak supervision can improve accuracy with no human-generated labels.

can improve relative performance by 46.4% for video analytics and 33% for AVs. Similarly, the ECG classification can also improve with no human-generated labels. These results show that model assertions can be useful as a primitive for improving model quality with no additional data labeling.

6 Related Work

ML QA. A range of existing ML QA systems focus on validating inputs via schemas (Polyzotis et al., 2019; Baylor et al., 2017) or tracking performance over time (Baylor et al., 2017). However, these systems apply to situations with meaningful schemas (e.g., tabular data) and ground-truth labels at test time (e.g., predicting click-through rate). While model assertions could also apply to these cases, we target situations that do not contains meaningful schemas or labels at test time.

A variety of other ML QA systems focus on training pipelines (Renggli et al., 2019) or validating numerical errors (Odena & Goodfellow, 2018). These approaches are important at finding pre-deployment bugs, but do not apply to test-time scenarios. As a result, they are complementary to model assertions.

White-box testing systems, e.g., DeepXplore (Pei et al., 2017), test ML models by taking inputs and perturbing them. However, as we have discussed, a validation set cannot cover all possibilities in the deployment set. Furthermore, these systems do not give guarantees under model drift.

Verified ML. Verification has been applied to ML models in simple cases. For example, Reluplex (Katz et al., 2017) can verify that extremely small networks will make correct control decisions given a fixed set of inputs and other work has shown that similarly small networks can be verified against minimal perturbations of a fixed set of input images (Raghunathan et al., 2018). However, verification requires a specification, which may not be feasible to implement, e.g., even humans may disagree on certain predictions (Kirillov et al., 2018). Furthermore, the largest verified networks we are aware of (Katz et al., 2017; Raghunathan et al., 2018; Wang et al., 2018; Sun et al., 2019) are orders of magnitude smaller than the networks we consider.

Software Debugging. Writing correct software and verifying the correctness of software has a long history, with many proposals from the research community. We hope that many such practices are adopted in deploying machine learning models; we focus on assertions in this work (Goldstine et al., 1947; Turing, 1949). Assertions have been shown to reduce the prevalence of bugs, when deployed correctly (Kudrjavets et al., 2006; Mahmood et al., 1984). There are many other such methods, such as formal verification (Klein et al., 2009; Leroy, 2009; Keller, 1976), methods of conducting large-scale testing (e.g., fuzzing) (Takanen et al., 2008; Godefroid et al., 2012), and symbolic execution to trigger assertions (King, 1976; Cadar et al., 2008). Probabilistic assertions have been used to verify simple distributional properties of programs, such as differentially private programs should return an expected mean (Sampson et al., 2014). However, ML developers may not be able to specify distributions and data may shift in deployment.

Structured Prediction, Inductive Bias. Several ML methods encode structure or inductive biases into the training procedure or models themselves (BakIr et al., 2007; Haussler, 1988). For example, structured prediction attempts to predict output with additional constraints (e.g., object detection) (BakIr et al., 2007). While such methods are promising, designing algorithms and models with specific inductive biases can be challenging for non-experts. Additionally, these methods generally do not contain runtime checks for aberrant behavior.

Weak Supervision, Semi-supervised Learning. The goal of weak supervision is to leverage higher-level and/or noisier input from human experts to improve the quality of models (Varma et al., 2016; Ratner et al., 2017; Jin et al., 2018). In semi-supervised learning, structural assumptions over the data are used to leverage unlabeled data (Zhu, 2011). However, these methods generally do not contain runtime checks and, to the best of our knowledge, have not been used as a form of active learning.

7 Conclusion

In this work, we introduce model assertions, a model-agnostic technique that allows domain experts to indicate errors in ML models. We show that model assertions can be used at runtime to detect high-confidence errors, which prior methods would not detect. We also propose methods to use model assertions for active learning and weak supervision to improve model quality. We implement model assertions in LIBMA to demonstrate that they can apply to a wide range of real-world ML tasks.

References

Af classification from a short single lead ecg recording: the physionet/computing in cardiology challenge

2017, 2017. URL https://physionet.org/challenge/2017/.

Scale api: The api for training data, 2019. URL https://scale.ai/.

Athalye, A., Engstrom, L., Ilyas, A., and Kwok, K. Synthesizing robust adversarial examples. *ICML*, 2018.

Auer, P., Cesa-Bianchi, N., and Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

BakIr, G., Hofmann, T., Schölkopf, B., Smola, A. J., Taskar, B., and Vishwanathan, S. *Predicting structured data*. MIT press, 2007.

Baylor, D., Breck, E., Cheng, H.-T., Fiedel, N., Foo, C. Y., Haque, Z., Haykal, S., Ispir, M., Jain, V., Koc, L., et al. Tfx: A tensorflow-based production-scale machine learning platform. In *SIGKDD*. ACM, 2017.

Berry, D. A. and Fristedt, B. Bandit problems: sequential allocation of experiments (monographs on statistics and applied probability). *London: Chapman and Hall*, 5: 71–87, 1985.

Bubeck, S., Munos, R., and Stoltz, G. Pure exploration in multi-armed bandits problems. In *International conference on Algorithmic learning theory*, pp. 23–37. Springer, 2009.

Cadar, C., Dunbar, D., Engler, D. R., et al. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *OSDI*, volume 8, pp. 209–224, 2008.

Caesar, H., Bankiti, V., Lang, A. H., Vora, S., Liong, V. E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., and Beijbom, O. nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019.

Canel, C., Kim, T., Zhou, G., Li, C., Lim, H., Andersen, D., Kaminsky, M., and Dulloor, S. Scaling video analytics on constrained edge nodes. *SysML*, 2019.

Chen, L., Xu, J., and Lu, Z. Contextual combinatorial multi-armed bandits with volatile arms and submodular reward. In *Advances in Neural Information Processing Systems*, pp. 3247–3256, 2018.

Chinchali, S., Sharma, A., Harrison, J., Elhafsi, A., Kang, D., Pergament, E., Cidon, E., Katti, S., and Pavone, M. Network offloading policies for cloud robotics: a learning-based approach. *arXiv preprint arXiv:1902.05703*, 2019.

Coldewey, D. Uber in fatal crash detected pedestrian but had emergency braking disabled, 2018. URL https://techcrunch.com/2018/05/24/uber-in-fatal-crash-detected-pedestrian-but-had-emergency-braking-disabled/.

Davies, A. How do self-driving cars see? (and how do they see me?), 2018. URL https://www.wired.com/story/the-know-it-alls-how-do-self-driving-cars-see/.

EHRA. Guidelines for the management of atrial fibrillation: the task force for the management of atrial fibrillation of the european society of cardiology (esc). *European heart journal*, 31(19):2369–2429, 2010.

Evans, L. C. Graduate studies in mathematics. In *Partial differential equations*. Am. Math. Soc., 1998.

Girshick, R., Radosavovic, I., Gkioxari, G., Dollár, P., and He, K. Detectron. https://github.com/facebookresearch/detectron, 2018.

Godefroid, P., Levin, M. Y., and Molnar, D. Sage: whitebox fuzzing for security testing. *Queue*, 10(1):20, 2012.

Goldstine, H. H., Von Neumann, J., and Von Neumann, J. Planning and coding of problems for an electronic computing instrument. 1947.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *ICLR*, 2015.

Haussler, D. Quantifying inductive bias: Ai learning algorithms and valiant's learning framework. *Artificial intelligence*, 36(2):177–221, 1988.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pp. 2980–2988. IEEE, 2017.

Hirth, M., Hoßfeld, T., and Tran-Gia, P. Analyzing costs and accuracy of validation mechanisms for crowdsourcing platforms. *Mathematical and Computer Modelling*, 57 (11-12):2918–2932, 2013.

Hsieh, K., Ananthanarayanan, G., Bodik, P., Venkataraman, S., Bahl, P., Philipose, M., Gibbons, P. B., and Mutlu, O. Focus: Querying large video datasets with low latency and low cost. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pp. 269–286, 2018.

Jiang, J., Ananthanarayanan, G., Bodik, P., Sen, S., and Stoica, I. Chameleon: scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pp. 253–266. ACM, 2018.

Jin, S., RoyChowdhury, A., Jiang, H., Singh, A., Prasad, A., Chakraborty, D., and Learned-Miller, E. Unsupervised hard example mining from videos for improved object detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 307–324, 2018.

Kang, D., Emmons, J., Abuzaid, F., Bailis, P., and Zaharia, M. Noscope: optimizing neural network queries over video at scale. *Proceedings of the VLDB Endowment*, 10 (11):1586–1597, 2017.

Kang, D., Bailis, P., and Zaharia, M. Blazeit: Fast exploratory video queries using neural networks. *arXiv preprint arXiv:1805.01046*, 2018.

Katz, G., Barrett, C., Dill, D. L., Julian, K., and Kochenderfer, M. J. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pp. 97–117. Springer, 2017.

Keller, R. M. Formal verification of parallel programs. *Communications of the ACM*, 19(7):371–384, 1976.

King, J. C. Symbolic execution and program testing. *Communications of the ACM*, 19(7):385–394, 1976.

Kirillov, A., He, K., Girshick, R., Rother, C., and Dollár, P. Panoptic segmentation. *arXiv preprint arXiv:1801.00868*, 2018.

Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., et al. sel4: Formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pp. 207–220. ACM, 2009.

Kudrjavets, G., Nagappan, N., and Ball, T. Assessing the relationship between software assertions and faults: An empirical investigation. In *Software Reliability Engineering, 2006. ISSRE'06. 17th International Symposium on*, pp. 204–212. IEEE, 2006.

Lang, A. H., Vora, S., Caesar, H., Zhou, L., Yang, J., and Beijbom, O. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 12697–12705, 2019.

Lee, T. Tesla says autopilot was active during fatal crash in mountain view. https://arstechnica.com/cars/2018/03/tesla-says-autopilot-was-active-during-fatal-crash-in-mountain-view/, 2018.

Leroy, X. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft coco: Common objects in context. In *European conference on computer vision*, pp. 740–755. Springer, 2014.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. Ssd: Single shot multibox detector. In *European conference on computer vision*, pp. 21–37. Springer, 2016.

Lu, T., Pál, D., and Pál, M. Contextual multi-armed bandits. In *Proceedings of the Thirteenth international conference on Artificial Intelligence and Statistics*, pp. 485–492, 2010.

Mahmood, A., Andrews, D. M., and McCluskey, E. J. Executable assertions and flight software. 1984.

Mintz, M., Bills, S., Snow, R., and Jurafsky, D. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pp. 1003–1011. Association for Computational Linguistics, 2009.

Odena, A. and Goodfellow, I. Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. *arXiv preprint arXiv:1807.10875*, 2018.

Pei, K., Cao, Y., Yang, J., and Jana, S. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pp. 1–18. ACM, 2017.

Polyzotis, N., Zinkevich, M., Roy, S., Breck, E., and Whang, S. Data validation for machine learning. *SysML*, 2019.

Radlinski, F., Kleinberg, R., and Joachims, T. Learning diverse rankings with multi-armed bandits. In *Proceedings of the 25th international conference on Machine learning*, pp. 784–791. ACM, 2008.

Raghunathan, A., Steinhardt, J., and Liang, P. Certified defenses against adversarial examples. *arXiv preprint arXiv:1801.09344*, 2018.

Rajpurkar, P., Hannun, A. Y., Haghpanahi, M., Bourn, C., and Ng, A. Y. Cardiologist-level arrhythmia detection with convolutional neural networks. *Nature Medicine*, 2019.

Ratner, A., Bach, S., Varma, P., and Ré, C. Weak supervision: The new programming paradigm for machine learning, 2017. URL https://dawn.cs.stanford.edu/2017/07/16/weak-supervision/.

Renggli, C., Karla, B., Ding, B., Liu, F., Schawinski, K., Wu, W., and Zhang, C. Continuous integration of machine learning models with ease.ml/ci: Towards a rigorous yet practical treatment. *SysML*, 2019.

Sampson, A., Panchekha, P., Mytkowicz, T., McKinley, K. S., Grossman, D., and Ceze, L. Expressing and verifying probabilistic assertions. *ACM SIGPLAN Notices*, 49 (6):112–122, 2014.

Sener, O. and Savarese, S. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*, 2017.

Settles, B. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

Sun, C., Shrivastava, A., Singh, S., and Gupta, A. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pp. 843–852, 2017.

Sun, X., Khedr, H., and Shoukry, Y. Formal verification of neural network controlled autonomous systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pp. 147–156. ACM, 2019.

Takanen, A., Demott, J. D., and Miller, C. *Fuzzing for software security testing and quality assurance*. Artech House, 2008.

Taylor, L. and Nitschke, G. Improving deep learning using generic data augmentation. *arXiv preprint arXiv:1708.06020*, 2017.

Tran-Thanh, L., Venanzi, M., Rogers, A., and Jennings, N. R. Efficient budget allocation with accuracy guarantees for crowdsourcing classification tasks. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pp. 901–908. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

Turing, A. Checking a large routine. In *Report on a Conference on High Speed Automatic Calculating machines*, pp. 67–69. Cambridge University Mathematics Lab, 1949.

Varma, P., He, B., Iter, D., Xu, P., Yu, R., De Sa, C., and Ré, C. Socratic learning: Augmenting generative models to incorporate latent subsets in training data. *arXiv preprint arXiv:1610.08123*, 2016.

Wang, J. and Perez, L. The effectiveness of data augmentation in image classification using deep learning. *Convolutional Neural Networks Vis. Recognit*, 2017.

Wang, S., Pei, K., Whitehouse, J., Yang, J., and Jana, S. Formal security analysis of neural networks using symbolic intervals. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pp. 1599–1614, 2018.

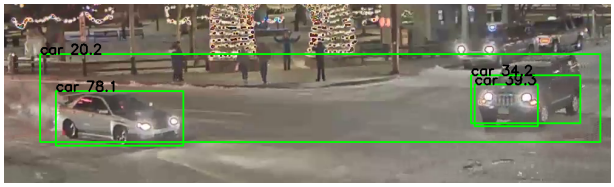
Xu, T., Botelho, L. M., and Lin, F. X. Vstore: A data store for analytics on large videos. In *Proceedings of the Fourteenth EuroSys Conference 2019*, pp. 16. ACM, 2019.

Yan, Y., Mao, Y., and Li, B. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10):3337, 2018.

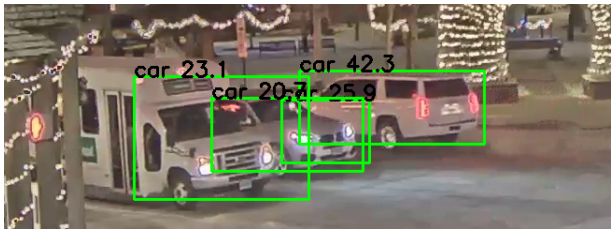
Zhu, X. Semi-supervised learning. In *Encyclopedia of machine learning*, pp. 892–897. Springer, 2011.

(a) Frame 1 (b) Frame 2

Figure 6. Two example frames from the same scene with an inconsistent attribute (the identity) from the TV news use case.

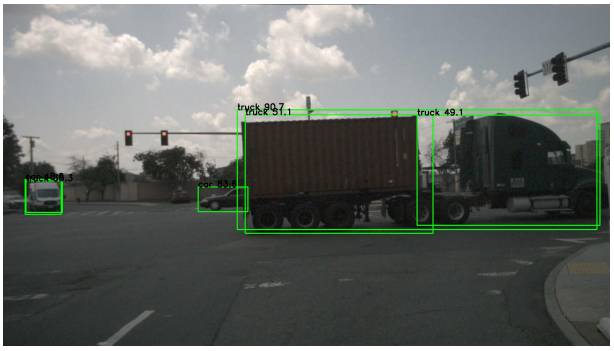


(a) Example error 1.



(b) Example error 2.

Figure 7. Examples errors when three boxes highly overlap (see `multibox` in Section 5). Best viewed in color.



(a) Example error flagged by `multibox`. SSD predicts three trucks when only one should be detected.

(b) Example error flagged by `agree`. SSD misses the car on the right and the LIDAR model predicts the truck on the left to be too large.

Figure 8. Examples of errors that the `multibox` and `agree` assertions catch for the NuScenes dataset. LIDAR model boxes are in pink and SSD boxes are in green. Best viewed in color.

A Examples of Errors Caught by Model Assertions

In this section, we illustrate several errors caught by the model assertions used in our evaluation.

First, we show an example error in the TV news use case in Figure 6. Recall that these assertions were generated with our consistency API (§4). In this example, the identifier is the box's `sceneid` and the attribute is the `identity`.

Second, we show an example error for the visual analytics use case in Figure 7 for the `multibox` assertion. Here, SSD erroneously detects multiple cars when there should be one.

Third, we show two example errors for the AV use case in Figure 8 from the `multibox` and `agree` assertions.

Assertion class	Assertion sub-class	Description	Examples
Consistency	Multi-source	Model outputs from multiple sources should agree	• Verifying human labels (e.g., number of labelers that disagree) • Multiple models (e.g., number of models that disagree)
	Multi-modal	Model outputs from multiple modes of data should agree	• Multiple sensors (e.g., number of disagreements from LIDAR and camera models) • Multiple data sources (e.g., text and images)
	Multi-view	Model outputs from multiple views of the same data should agree	• Video analytics (e.g., results from overlapping views of different cameras should agree) • Medical imaging (e.g., different angles should agree)
Domain knowledge	Physical	Physical constraints on model outputs	• Video analytics (e.g., cars should not flicker) • Earthquake detection (e.g., earthquakes should appear across sensors in physically consistent ways) • Protein-protein interaction (e.g., number of overlapping atoms)
	Unlikely scenario	Scenarios that are unlikely to occur	• Video analytics (e.g., maximum confidence of 3 vehicles that highly overlap), • Text generation (e.g., two of the same word should not appear sequentially)
Perturbation	Insertion	Inserting certain types of data should not modify model outputs	• Visual analytics (e.g., synthetically adding a car to a frame of video should be detected as a car), • LIDAR detection (e.g., similar to visual analytics)
	Similar	Replacing parts of the input with similar data should not modify model outputs	• Sentiment analysis (e.g., classification should not change with synonyms) • Object detection (e.g., painting objects different colors should not change the detection)
	Noise	Adding noise should not modify model outputs	• Image classification (e.g., small Gaussian noise should not affect classification) • Time series (e.g., small Gaussian noise should not affect time series classification)
Input validation	Schema validation	Inputs should conform to a schema	• Boolean features should not have inputs that are not 0 or 1 • All features should be present

Table 6. Example of model assertions. We describe several assertion classes, sub-classes, and concrete instantiations of each class. In parentheses, we describe a potential severity score or an application.

B Classes of Model Assertions

We present a non-exhaustive list of common classes of model assertions in Table 6 and below. Namely, we describe how one might look for assertions in other domains.

Our taxonomization is not exact and several examples will contain features from several classes of model assertions. Prior work on schema validation (Polyzotis et al., 2019; Baylor et al., 2017) and data augmentation (Wang & Perez, 2017; Taylor & Nitschke, 2017) can be cast in the model assertion framework. As these have been studied, we do not focus on these classes of assertions in this work.

Consistency assertions. An important class of model assertions checks the consistency across multiple models or sources of data. The multiple sources of data could be the output of multiple ML models on the same data, multiple sensors, or multiple views of the same data. The output from the various sources should agree and consistency model assertions specify this constraint. These assertions can be generated via our API as described in §4.

Domain knowledge assertions. In many physical domains, domain experts can express physical constraints or unlikely scenarios. As an example of a physical constraint, when predicting how proteins will interact, atoms should not physically overlap. As an example of an unlikely scenario, boxes of the visible part of cars should not highly overlap (Figure 7). In particular, model assertions of unlikely scenarios may not be 100% precise, i.e., will be soft assertions.

Perturbation assertions. Many domains contain input and output pairs that can be perturbed (perhaps jointly) such that the output does not change. These perturbations have been widely studied through the lens of data augmentation (Wang & Perez, 2017; Taylor & Nitschke, 2017) and adversarial examples (Goodfellow et al., 2015; Athalye et al., 2018).

Input validation assertions. Domains that contain schemas for the input data can have model assertions that validate the input data based on the schema (Polyzotis et al., 2019; Baylor et al., 2017). For example, boolean inputs that are encoded with integral values (i.e., 0 or 1) should never be negative. This class of assertions is an instance of preconditions for ML models.

C Hyperparameters

Hyperparameters for active learning experiments. For `night-street`, we used 300,000 frames of one day of video for the training and unlabeled data. We sampled 100 frames per round for five rounds and used 25,000 frames of a different day of video for the test set. Due to the cost of obtaining labels, we ran each trial twice.

For the NuScenes dataset, we used 350 scenes to bootstrap the LIDAR model, 175 scenes for unlabeled/training data for SSD, and 75 scenes for validation (out of the original 850 labeled scenes). We trained for one epoch at a learning rate of 5×10^{-5}. We ran 8 trials.

For the ECG dataset, we train for 5 rounds of active learning with 100 samples per round. We use a learning rate of 0.001 until the loss plateaus, which the original training code did.

Hyperparameters for weak supervision experiments. For `night-street`, we used 1,000 additional frames with 750 frames that triggered `flicker` and 250 random frames with a learning rate of 5×10^{-6} for a total of 6 epochs.

For the NuScenes dataset, we used the same 350 scenes to bootstrap the LIDAR model as in the active learning experiments. We trained with 175 scenes of weakly supervised data for one epoch with a learning rate of 5×10^{-5}.

For the ECG dataset, we use 1,000 weak labels and the same training procedure as in active learning.