Human-efficient Discovery of Training Data for Visual Machine Learning

Ziqiang Feng
zf@cs.cmu.edu

July 2019

Thesis Proposal

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Mahadev Satyanarayanan (Satya) (Chair)
Martial Hebert
Roberta Klatzky
Padmanabhan Pillai (Intel Labs)

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

Contents

1 Introduction

Supervised deep learning has become the *de facto* standard for solving many computer vision problems, including image classification, object detection, and activity recognition. Deep learning avoids the need for domain-specific feature engineering. A domain expert can supervise the learning of *deep neural networks (DNNs)* by creating a large training set with labels (e.g., positive/negative for binary classification). The potential of capturing an expert's knowledge in a DNN is attractive in domains such as medical research, ecology, and military intelligence. The discriminative power of DNNs can embody domain-specific questions that are hard for a human non-expert to answer. Examples include: "Does this pathology image show pagetoid spread or nuclear atypia?"; "Is this a caterpillar of the pest moth *Cactoblastis cactorum* or of a benign moth?"; "Is this seismic image indicative of an underground nuclear explosion or a minor earthquake?".

Deep learning typically involves a three-step approach: (1) collecting and labeling a data set with sufficient (positive) examples; (2) designing and training a DNN using the labeled data; (3) deploying the DNN "in the field" for inference. Since AlexNet's [37] victory in the 2012 ImageNet competition, many advances have been made to accelerate and improve the training and inference of DNNs, including software frameworks such as TensorFlow [2] and PyTorch [45], DNN architectures such as ResNet [27] and MobileNet [28], as well as hardware accelerators such as Intel's Movidius VPU [15] and Google's TPU [34].

What remains a critical challenge is for domain experts to discover occurrences of rare phenomena, aka positive examples, from unlabeled data in order to construct a training set of adequate size. By definition, neither a detector nor an index exists for a novel target until a sufficient number of examples are discovered and used to train a DNN. Therefore the discovery of training data cannot be fully delegated to the computer. The prevalent solution to-date is to let humans examine a large quantity of examples exhaustively and assign labels to each of them. This process is time-consuming, attention-demanding, and often costly. For coarse-grained, mundane targets like cars, dogs, and chairs, human parallelism may be exploited through crowd-sourcing using tools such as Amazon Mechanical Turk (AMT) [53, 60]. Some well-known image data sets in academia including ImageNet [51] and COCO [38] were created in this manner.

Unfortunately, crowd-sourcing is not a viable solution for domain experts. First, crowds are not experts. By definition, only the expert possesses the depth of domain-specific knowledge needed to reliably distinguish true positives from false positives, and to avoid false negatives. Second, access to data sources is commonly restricted for reasons such as patient privacy, national security, or business policy. In the worst case, a single domain expert needs to create an entire training set alone. DNNs typically need to be trained on thousands of labeled examples before their accuracy reaches an acceptable level. How many hours will it take for a domain expert to assemble such a large training data set for a rare phenomenon? How much junk will she need to wade through? Human-efficiency is an import metric here. There are three aspects of human-efficiency in this context: (a) leveraging human heuristics and domain expertise effectively; (b) avoiding long stalls on the user side awaiting results; and (c) avoiding overloading the human user with a flood of irrelevant results. An expert's time is precious. Her time is well spent if most of it is consumed in examining results that prove to be true positives. It is poorly spent if most of it is used to wait for results, or to dismiss frivolous false positives.

Without an accurate detector to perform automated labeling, what tasks can an expert delegate to the computer? Prior work in OpenDiamond [30] explored the concept of *early discard* for interactive search of non-indexed images. In OpenDiamond, a user creates early discard filters using heuristics and examples, which try to drop a bulk of clearly irrelevant (aka obviously negative) data. Early discard filters are not perfect classifiers. Their purpose is to reduce the amount of data that a user must sift through to find positive examples. OpenDiamond used classic computer vision techniques such as RGB color histogram, scale-invariant feature transform (SIFT), and perceptual hashing to perform discard.

While prior research has demonstrated the potential of early discard, it is insufficient for solving the data discovery problem faced by domain experts. Phenomena that interest experts are often rare. When the target is rare, the early discard filters tend to have low accuracy. For example, our experience shows that using OpenDiamond to find fire hydrants in everyday Flickr photos has a precision under 0.1%. There are two ways in OpenDiamond to create more accurate filters: (a) adding more example "reference" patches to classic computer vision filters, such as RGB color histogram and SIFT; (b) inventing more sophisticated algorithms, which requires writing new computer code. Improvement gained from option (a) is usually slim, as classic computer vision techniques are inefficient at generalizing from more examples. On the other hand, option (b) poses a high barrier for domain experts, who may not have deep additional skills in programming and implementing computer vision algorithms. OpenDiamond was conceived long before the modern advent of DNNs. Many machine learning techniques have since been proposed that can learn better knowledge by simply ingesting more example data, without the need of writing new code. For example, *transfer learning* [44] allows "transferring" knowledge learned for a different task and improving it with more example data for the current task. Besides, methods such as *region proposal networks* [50] make it possible to filter different regions of an image, as opposed to OpenDiamond's whole-image filtering, allowing for more complex early discard conditions. These techniques have the potential to greatly improve domain experts' productivity, but they require novel programming abstractions, system design, and optimization.

This proposal lays out my plan to address these challenges through building *Eureka*, a system for human-efficient discovery of training data. Central to my thesis is the idea of combining early discard with two key concepts: just-in-time machine learning and an iterative data discovery paradigm. Just-in-time machine learning refers to the use of a spectrum of machine learning methods — from simple ones like linear regression and support vector machine (SVM), to sophisticated ones like transfer learning and deep learning — to improve the efficacy of early discard without writing new code as soon as newly-labeled data is discovered. My proposed data discovery paradigm repeats the just-in-time machine learning process in an interruptible, interactive, and human-in-the-loop manner, where the incrementally growing data set goes hand in hand with incrementally more sophisticated early discard filters. The expert utilizes her intuition and judgment through this process to embody her deep knowledge of the result data set. To achieve human-efficiency, machine-efficiency is important for rapidly discarding irrelevant data and reducing user wait time. To this end, I will exploit techniques such as edge computing and intelligent storage devices for efficient data processing.

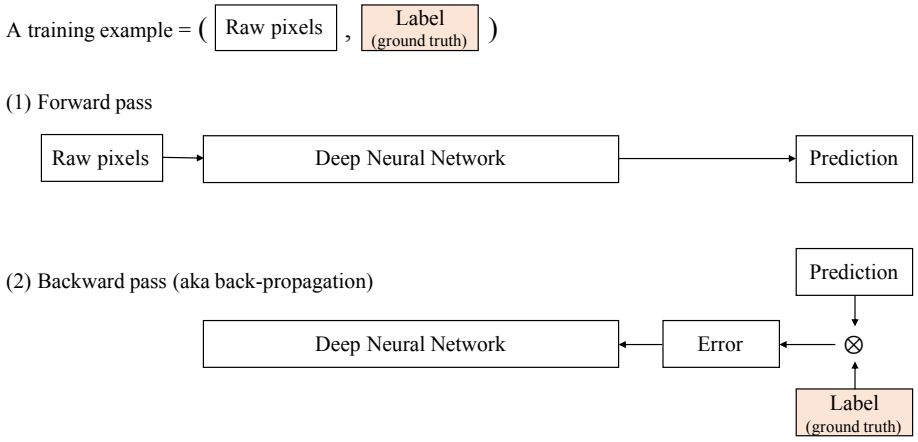


Figure 1: Training of a deep neural network (DNN) consists of two passes. The forward pass transforms input pixels into a best guess (prediction) using a model. The backward pass tries to adjust the model's parameters in the direction of reducing the measured error.

1.1 Thesis Statement

The manual effort of discovering a large training set for machine learning can be reduced by a system combining early discard, just-in-time machine learning, and the ability to create more accurate filters immediately without writing new code. This approach is efficient in different computing landscapes (e.g., edge computing and intelligent storage) and problem domains (e.g., object detection in images and activity recognition in videos).

1.2 Training Data for Supervised Deep Learning

Deep learning has emerged as a powerful tool for solving a wide range of computer vision problems, such as image classification [37], object detection [50], face recognition [5], and human action recognition [54]. Deep learning contrasts with classical methods such as support vector machine (SVM) that take as input explicit, engineered visual features. Instead, deep neural networks (DNNs) work end-to-end by taking raw RGB pixels as input, and outputting their prediction for the task, for example, classifying an image as dog versus cat.

DNNs need to be trained in order to be useful, thus the "learning" part in deep learning. Training of a DNN consists of two passes, as depicted in Figure 1. The forward pass takes as input the raw pixels of the image, and performs a series of non-linear transformations of them into a prediction — a best guess for the task. The transformations are represented by a model having many tunable parameters (aka weights). The backward pass (aka back-propagation) compares the DNN's prediction against the ground truth label and calculates the measured error. It then tries to adjust the parameters of the model in the direction of reducing the error using algorithms such as gradient descent. Labeled data is a key ingredient in this process. The forward pass and backward pass are typically repeated many times on different training examples to adjust the parameters until a satisfactory accuracy is reached.

DNN models usually have many layers and millions to billions of parameters, thus the "deep"

Data set	Size	Example labels
Object recognition/detection		
ImageNet [51]	1200 thousand images	car wheel, goldfish, Irish terrier, kite
COCO [38]	330 thousand images	person, bicycle, cat, dog, knife
PASCAL VOC [21]	12 thousand images	person, chair, dining table, dog
Activity recognition		
UCF101 [56]	13 thousand videos	basketball dunk, jump rope, play guitar
Charades [53]	10 thousand videos	eat sandwich, hold broom, open fridge
Unlabeled		
YFCC100M [58]	99.2 million images 0.8 million videos	

Table 1: Examples of large data sets that empower the advancement of deep learning.

in its name. Its capability to capture complicated knowledge entails the hunger for a large amount of *labeled* data. Indeed, the advancement of computer vision in recent years has been empowered by the availability of large labeled data sets. Table 1 shows the information of a few well-known data sets. In general, DNNs need to be trained on at least thousands of examples before their accuracy reaches acceptable levels. Creating these data sets is a huge undertaking, because it demands a lot of human attention and labor. Many of the publicly available image data sets exploited crowd-sourcing through platforms such as Amazon Mechanical Turk to parallelise the labeling tasks over many crowd workers.

1.3 Challenges Faced by Domain Experts

While deep learning is being widely used to detect everyday objects such as persons and cars for autonomous vehicles and traffic monitoring, its adoption by domain experts, such as ecologists and medical researchers, has been increasing more slowly. Based on prior experience interacting with experts from several domains, the major hurdle is the difficulty of constructing training sets. In order to solve domain-specific vision problems, domain experts need to create large labeled data sets of domain-specific targets. Unfortunately, crowd-sourcing has little to offer to this end. The task of distinguishing between delta smelt and threadfin shad *Dorosoma petenense* [23], for example, cannot be delegated to a less-skilled crowd worker. By definition, expertise in any domain is scarce, embodying large investment of time and money in education, training, and working experience. Crowd-sourcing to experts can be prohibitively expensive. Besides, data in many specialized domains (e.g., patients' chest X-rays) is under access restrictions for privacy, business policy, or national security reasons. In the extreme case, only one expert may have access to the data.

Interesting phenomena (objects or events) in many domains are usually rare, i.e., they have low *base rates* [39]. That is what makes them worth studying. The need to find rare phenomena from a large volume of unlabeled data poses a chicken-and-egg problem: without a large labeled data set, one cannot train an accurate DNN; without an accurate DNN, one cannot delegate automated labeling to the computer. Consider an expert trying to find a thousand *positive* examples

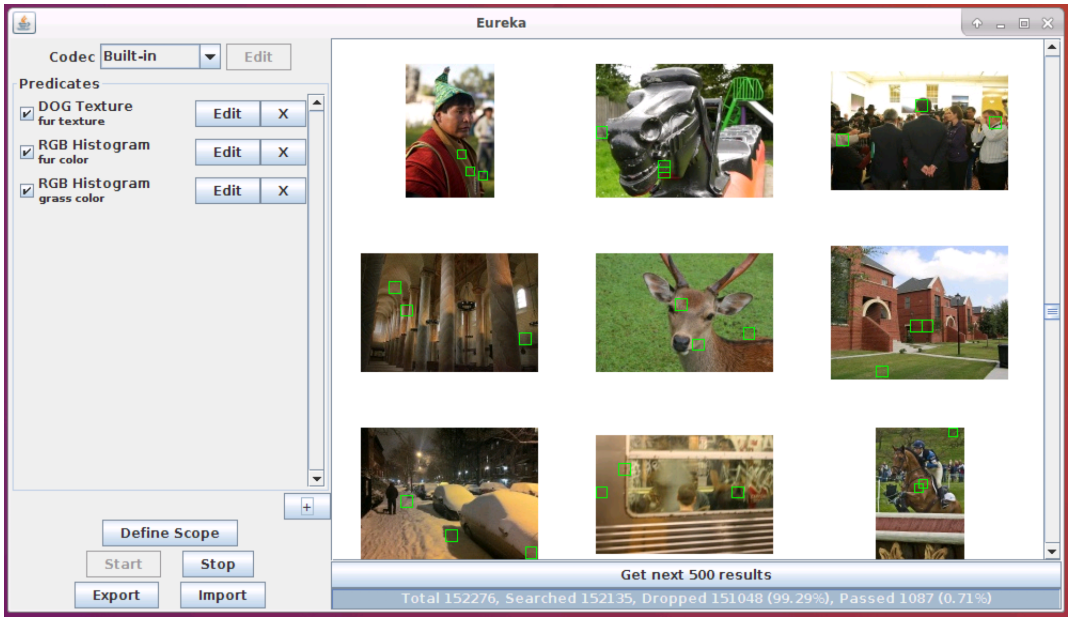


Figure 2: OpenDiamond [30] example: combining a Difference of Gaussians (DoG) texture filter and two RGB color histogram filters to find deer images. Out of nine candidates shown above, one is true positive.

of a phenomenon that is one-in-a-thousand rare. To do so, she is expected to go through one million images. Assuming it takes 20 seconds to label each image, it will take her seven sleepless months to accomplish the task! As a real world example, the MURA data set of 40,561 images for abnormality detection in radiology took 11 years to create [46].

Prior work in OpenDiamond [30] shed light on the problem with the concept of *early discard*. Figure 2 shows an example in OpenDiamond where the user combines a Difference of Gaussians (DoG) filter and two RGB color histogram filters to find images of deer from a Flickr photo data set. The process is intrinsically human-in-the-loop. The purpose of the filters is to discard clearly irrelevant images and thus trim the space of data that a human user must manually inspect and hand-label. In Figure 2, 99.29% of images in the data set are discarded by the filters (as shown in the status bar at the bottom: "Total 152276, Searched 152135, Dropped 151048 (99.29%), Passed 1087 (0.71%)"), whereas only one candidate shown in the screenshot is true positive.

Early discard alone, however, is not sufficient for solving the problem faced by domain experts. First, when the target is rare, the early discard filters' precision (i.e., the fraction of returned candidates that are truly targets) is usually low. Table 2 summarizes our experience of using OpenDiamond [30] to discover positive examples of three rare object classes from the Yahoo Flickr Creative Commons 100 Million data set [58]: deer, Taj Mahal, and fire hydrants. All three targets have estimated base rates under 0.1%. That is, they are rarer than one-in-a-thousand. The rarer a target is, the lower is the precision of the filters. In the case of fire hydrants, the human user needs to hand-label more than one million candidates to discover 1000 true positives. To address this issue, the user needs to be able to improve the efficacy of early discard during the human-in-the-loop process.

Target	Estimated base rate in YFCC100M	Filter precision in OpenDiamond	Hand-labeled images to discover 1000 true positives
Deer	0.07 %	0.27%	370 thousand
Taj Mahal	0.02 %	0.11%	900 thousand
Fire hydrant	0.005%	0.09%	1110 thousand

Table 2: OpenDiamond's [30] effectiveness in discovering rare objects in YFCC100M [58].

Figure 3: Searching for "person in red shirt" in common Flickr-style photos (left) and broad-view traffic cameras (right), using a person detector and a RGB color patch filter.

Second, OpenDiamond uses a simple whole-image filtering abstraction, making it ineffective on high-resolution, broad-view camera frames, which is a growing class of data sources. To illustrate, consider the task of finding examples of "person in red shirt". In OpenDiamond, one would use two filters: (1) a person detector that drops images with no person detected; (2) a RGB color patch filter that drops images with no red color patch. Figure 3 (left) shows how the filters will fire on a commonplace Flickr-style photo, where green bounding boxes mark detected persons and a yellow box marks detected red patch. The same filters, unfortunately, tend to fire many false positives on images like Figure 3 (right), where persons appear on most frames and there are a lot of unrelated red pixels.

Intuitively, to address the above issue, one should execute the RGB color patch filter *inside* the bounding boxes on which the person detector fires. More generally, the user should be able to compose complex queries where filters depend on each other and operate on different granularity of data. OpenDiamond's "flat" data abstraction falls short of this complexity. On the other hand, computer vision is highly open-ended — there are unlimited ways to sub-divide each image. Our new system must be able to support flexible queries without requiring the domain experts to write task-specific code. These requirements pose challenges in terms of system design, architecture, and optimization.

Research Focus	Unique Challenges
Edge computing Completed. Section 4.	Low-bandwidth and long-latency WAN between edge nodes and client.
Cloud computing Completed. Section 5.	Long data access latency from compute node and contention for shared data bandwidth due to separation of compute layer and storage layer.
Smart Disks (emulated) Section 6.	Low compute capability of on-device processors and difficulty of programming.

Table 3: Research challenges of Eureka on different computing landscapes. Evaluation will be done with image data.

Research Focus	Unique Challenges
Object recognition in images Completed. Section 4.	Continuous improvement of filter accuracy without creating new code.
Activity recognition in videos Section 7	Programming abstraction and optimization for filtering the extra time dimension.
Queries on other multidimensional data (e.g., maps, pathological images) Section 8	Defining feasible filters and queries. Designing domain-specific query interface.

Table 4: Research challenges of Eureka for different problem domains. Evaluation will be done in the edge computing setting.

1.4 Research Plan

Eureka on Different Computing Landscapes

I plan to address the above challenges through building Eureka — a system for human-efficient discovery of training data for visual machine learning. As said, machine efficiency is crucial for achieving human efficiency, otherwise an expert's time will be wasted in waiting. In this regard, the crucial metric is *machine processed items per second*. Eureka needs to execute computation efficiently in various computing landscapes. For example, a wildlife ecologist may track animal activities using battery-backed wireless camera traps deployed in the forest, while a medical researcher searches through multi-year patient records concentrated at a few hospitals' servers. Under different compute/storage manifestations, the performance bottleneck in the system is likely to shift between hard disk, SSD, DRAM, CPU, network, etc. Specialized optimization may be required in different settings to alleviate hardware and software bottlenecks. I plan to focus on three prevailing or emerging computing landscapes shown in Table 3. I will evaluate my research on these different settings using image data. I outline their motivation, unique challenges and preliminary result in the corresponding sections.

Eureka for Different Problem Domains

To demonstrate that Eureka's methodology is effective for a variety of problem domains, I will design and evaluate Eureka using three different data/query types shown in Table 4, in the edge computing setting. The crucial metric here is *number of labeled true positives per minute*, aka *productivity*. This in turn depends on the human-centric time to examine each candidate, the rate of candidate arrival (thus machine efficiency), and the number of false positives (thus accuracy of the filters used). While results from different problem domains cannot compare with each other, we expect that in each task, an expert's productivity is improved over the iterative approach of Eureka. Each data/query type in Table 4 has unique characteristics (e.g., the time dimension in videos), demanding new query interface, system optimization and functionality. I will show how Eureka's design unifies the differences into a common framework to support expressive and efficient queries on different data.

2 Related Work

In order to solve a computer vision problem with deep learning, one typically follows a three-step workflow: (1) collecting and labeling a data set of sufficient size; (2) designing a DNN architecture and training it using the labeled data set; (3) deploying the DNN model "in the field" to perform inference on new data. Most of prior research effort focuses on (2) and (3), accelerating and improving the usability of DNN's training and inference (Section 2.1). The challenge of (1), creating labeled data, is relatively less explored (Section 2.2). This proposal addresses (1) with a unique focus on the combination of domain experts and visual data, without programming skills required of the user.

2.1 Training and Inference of Deep Neural Networks

In computer vision, an image is typically represented as a high-dimensional RGB pixel array. A deep neural network (DNN) is a sequence of non-linear transformations (aka layers) of the arrays into high-level semantic representations, such as the probability of an image having a dog versus a cat. Those transformations have millions to billions of parameters, or weights, that can be learned to capture complicated knowledge. The large number of parameters makes DNNs powerful, and yet expensive to execute.

Pioneered by AlexNet [37], many DNN architectures have been invented to achieve faster speed, higher accuracy, or both. To name a few, VGG [55], Inception [57], ResNet [27], MobileNet [28], CFNet [59], and Faster R-CNN [50]. Through better design of the layers and their interconnections, these new DNN architectures have much fewer parameters and run faster, while achieving superior accuracy.

Software frameworks such as TensorFlow [1], PyTorch [45] and MXNet [11] make it easy to design new DNN architectures, and run training and inference on CPUs and GPUs. By providing high-level abstractions like models, layers and operators, these frameworks allow the programmers to focus on application logic rather than low-level details, such as efficient linear algebra implementation or interfacing with the GPU. Other work like Petuum [62] and federated learning

	Crowd-sourcing [18, 53, 60]	Expert/crowd-sourcing [43]	Snorkel [47]	**This proposal**
Target visual data	○			○
Target domain experts		○	○	○
No coding barrier	○	○		○

Table 5: This proposal versus related work in the context of training set creation.

[40] focus on training efficiently in a distributed setting.

Deploying DNNs for practical use often requires them to run on resource-constrained devices with limited compute, memory, and battery. DNN's large number of parameters and operations makes it challenging to meet latency requirements, or even be able to execute on low-power hardware. To address this challenge, various techniques of model compression [12, 13, 26, 31] have been proposed to reduce the size of the DNN models at the expense of minimal accuracy loss. In terms of hardware, vendors produce specialized accelerators such as Intel Movidius [15] and Google TPU [34] dedicated for deep learning workloads.

Application characteristics can be exploited to drive system design and optimization. Among others, video analytics has emerged as an important class of applications. *Video analytics* is the task of detecting objects or events of interest from live video streams or archival video data. There is recently a large body of work for accelerating video analytics in mobile, edge, or cloud settings [10, 25, 29, 32, 33, 35, 61, 64]. For example, NoScope [35] and Focus [29] exploit object and view angle bias of a particular camera to train a cheap "student model" to mimic the output of an expensive "teacher model" in order to achieve low inference latency. Wang et al. [61] use similar methods to reduce data transmission from drones over a wireless network of limited bandwidth. Others focus on sharing scarce resources in multi-tenant and multi-application settings [10, 25, 32, 64]. Visual data storage systems like VDMS [49] and VStore [63] optimize data retrieval from the disk to maximize throughput of the application layer.

Essentially, all above work for video analytics starts with the premise that a highly accurate, probably expensive "gold standard" DNN exists to detect the phenomenon of interest, and adopts a batch processing or stream processing approach. This proposal addresses a fundamentally different problem. A "gold standard" DNN does not exist yet for a novel phenomenon, which is the whole point of constructing a training set. Discovering training data is intrinsically a human-in-the-loop process, for which the vast body of techniques for batch or stream processing has little to offer.

2.2 Creation of Training Sets

A training set is a set of examples each annotated with ground truth labels pertaining to a specific task. Training data is a key ingredient in deep learning. The quantity, quality, and diversity of a training set has crucial impact on the accuracy of the trained DNN.

Data augmentation is a well-known technique to create quasi-new training examples from existing ones. It generates distorted variants of existing examples and adds them to the training set. Classic data augmentation methods include flipping, rotating, randomly cropping, and adding random noise to images. More sophisticated methods are proposed with the help of machine

learning and computer graphics [20, 48]. While data augmentation can increase the robustness of the trained DNN, it is no substitute for truly diverse new examples from the real world.

The canonical way of creating large training sets is crowd-sourcing — parallelising the hand-labeling tasks to many crowd workers on platforms such as Amazon Mechanical Turk. Many data sets listed in Table 1 were created in this way. Crowd-sourcing may be improved with *active learning* to reduce the time and monetary cost [60].

For reasons discussed in Section 1.3, crowd-sourcing is not applicable to many domains. Nguyen et al. [43] explore the mixing of the crowd and experts for literature screening. They reach out to the experts only when the crowd is not "confident enough." Recently, Snorkel [47] advocates asking domain experts to write labeling functions, instead of annotating examples. It then uses a generative model and statistical inference to calculate probabilistic ground truth labels. This approach requires experts to write code, which is a significant barrier. Both [43] and [47] focus on labeling of text data. Visual data (pixels) appears to be much more unstructured. It is difficult to codify human heuristics based on raw pixel values in the way Snorkel does. Table 5 summarizes how this proposal is different from representative work in the context of training set creation. My thesis considers the unique combination of visual data and domain experts, while posing no coding barrier for users.

3 Eureka Design

I have designed and implemented Eureka, a system for interactively searching and discovering training examples from large non-indexed data. The implementation of Eureka substantially extends OpenDiamond®, a platform for discard-based search [30]. Section 3.1 describes Eureka's system architecture, with an emphasis on discarding data as early as possible to alleviate bottlenecks. Section 3.2 proposes an iterative approach in which a domain expert interacts with the system to achieve increasing productivity. Sections 3.3 and 3.4 detail design and implementation that makes Eureka efficient and extensible.

3.1 System Architecture

Eureka considers unstructured visual data such as image and video stored or captured at a number of data sources. As discussed in Section 1.4, data sources may come in different forms (e.g., live camera feed from drones or archival patient records on hospital servers). A *cloudlet* is a compute infrastructure that can read from a data source at high bandwidth and low latency. In the setting of edge computing, the term "cloudlet" preserves its familiar interpretation. In others, a cloudlet may be re-interpreted as a machine in the data center, or an intelligent disk device. Figure 4 illustrates the system architecture of Eureka. A domain-specific front-end GUI runs on a client machine close to the expert. An early-discard back-end runs at each cloudlet. The back-ends execute on the cloudlets in parallel, and transmit thumbnails of undiscarded images to the front-end. Each thumbnail includes back pointers to its origin cloudlet and full-fidelity image on that cloudlet. The expert sees a merged stream of thumbnails from all back-ends. If a thumbnail merits closer examination, a mouse click on it will open a separate window to display the full-fidelity image and its associated meta-data. Thumbnails are queued by the front-end, awaiting

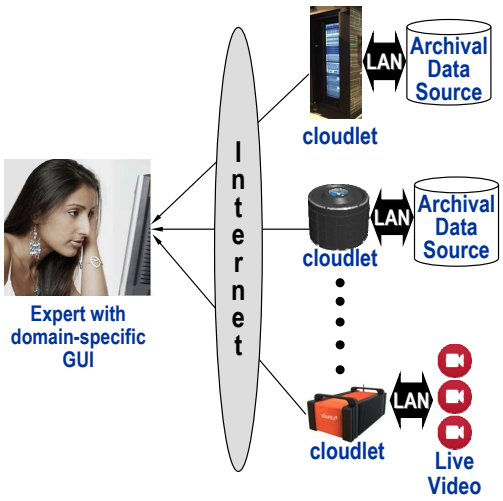


Figure 4: Eureka System Architecture

the expert's attention to inspect them. If demand greatly exceeds available attention, queue back pressure throttles cloudlet processing.

In all settings, high bandwidth and low latency access to data is used by early-discard code executing on a cloudlet. In other words, the rejection of clearly irrelevant data happens as early as possible in the processing pipeline that stretches from the data source to the user. With this architecture, the bandwidth demand between cloudlet and user is typically many orders of magnitude smaller than the bandwidth demand between data source and cloudlet. To avoid wasting the expert's time on waiting when each cloudlet yields candidates at a slow rate, Eureka harnesses high levels of parallelism by including more cloudlets in the process. Eureka can thus be viewed as an architecture that trades off computing resources (e.g., processing cycles, network bandwidth, and storage bandwidth) for effective use of expert attention. High-bandwidth access between cloudlet and data source is the key to making this trade-off effective.

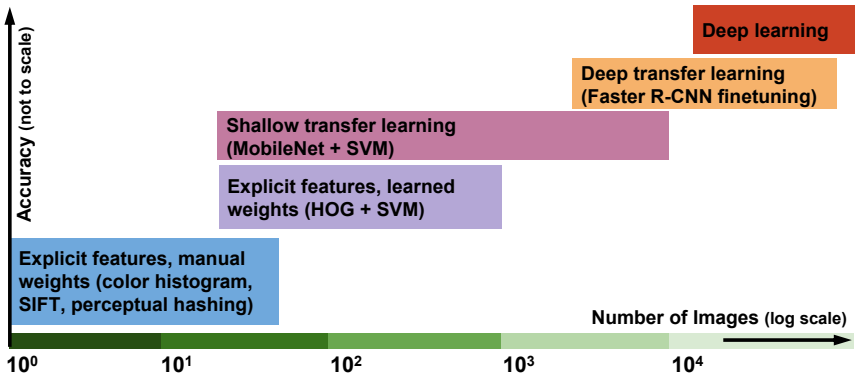
3.2 Iterative Early Discard and Just-in-time Machine Learning

Figure 2 shows a GUI that an expert uses to interact with Eureka. It allows the expert to construct a search query in the form of a pipeline of early-discard filters. When the user clicks "Start," Eureka deploys this pipeline of filters across many cloudlets to execute in parallel, and starts displaying returned candidates continuously. The expert examines the candidates shown in the window, and particularly picks out scarce true positives. At any time, the expert may click "Stop," effectively aborting the search, reconfigure the filter pipeline, and then start a new search. We define the expert's *productivity* as the number of true positives discovered per minute. Eureka's goal is to help an expert achieve high productivity.

Eureka advocates that an expert perform discovery of a rare phenomenon using an iterative workflow in order to progressively improve her productivity. To illustrate, consider the scenario described in Figure 5. Starting from just a few example images, how can the expert bootstrap her way to the thousands to tens of thousands of images needed for deep learning? We start with the premise that the *base rate* [39] is low — i.e., that the rodent is rarely seen, and hence there

An infectious disease expert has just learned that a shy rodent, long considered benign, may be the transmitter of a new disease. There is a need to create an accurate image classifier for this rodent so that it can be used in public health efforts to detect and eradicate the pest. The expert has only a few images of the rodent, but needs thousands to build an accurate DNN. There are likely to be some untagged occurrences of this rodent in the background of images that were captured for some other purpose in the epidemiological image collections of many countries. A Eureka search can reveal those occurrences. In the worst case, it may be necessary to deploy cloudlets with large arrays of associated cameras in the field to serve as live data sources for Eureka.

Figure 5: Example: Infectious Disease Control



Each type of model requires a different number of examples before its accuracy starts to improve steadily. Their accuracies saturate at different levels when sufficient data is given.

Figure 6: Training Data Set Size vs. Accuracy

are very few images in which it appears. If a good classifier already existed, it can be used to perform early discard on many cloudlets. The number of false positives would be low, and the rate of true positives would be reasonably high. The expert would neither waste her time rejecting obvious false positives, nor would she waste time waiting for the next image to appear. Most of her time would be spent examining results that prove to be true positives — in other words, high productivity. Alas, this state of affairs will only exist at the end of the Eureka workflow. No classifier exists at the beginning. What can she use at the start of the workflow?

Based on our experience, Figure 6 illustrates the trade-off that exists between classifier accuracy (higher is better, but not to scale) and the amount of labeled training data that is available. At the extreme left, the Eureka GUI allows simple features such as color and texture to be defined by example patches outlined on the few training examples that are available. This defines a very weak classifier that can be used as the basis of early discard. Because of the weakness of the classifier, there are likely to be many false positives. Unless the expert restricts her search to just a few data sources, she will be overwhelmed by the flood of false positives. Buried amidst the false positives are likely to be a few true positives. As the expert sees these in the result stream,

she labels and adds them to the training set. Over a modest amount of time (tens of minutes to a few hours, depending on the base rate and number of cloudlets), the training set is likely to grow to a few tens of images. At this point, there is a sufficient amount of training data to create a classifier based on more sophisticated features such as Histogram of Oriented Gradients (HOG), and learned weights from a simple machine learning algorithm such as support vector machine (SVM). This SVM is trained using examples that are newly-discovered, and thus cannot be pre-trained beforehand. The resulting classifier is still far from the desired accuracy, but it is significantly improved. Since the improved accuracy reduces the number of false positives, the number of data sources explored in parallel can be increased by recruiting more cloudlets. Once the training set size reaches a few hundreds, shallow transfer learning can be used. This yields an improved classifier that further reduces false positives, and allows further expansion in the number of data sources, thus speeding up the search. Once the training set size reaches a few thousands, deep transfer learning can be used and beyond that, deep learning. This iterative workflow can be terminated at any point if a classifier of sufficient accuracy has been obtained.

Throughout this iterative workflow, the most precious resource is the attention of the human expert. Eureka helps to optimize the use of this scarce resource in two ways. First, it enables immediate creation of more accurate filters without writing new code. It does so by leveraging machine learning techniques (e.g., SVM, deep learning) and trains new machine learning models *just-in-time*. Second, as improved filters become available, Eureka allows the search to be easily expanded to more cloudlets and data sources, thus harnessing more parallelism to increase the rate at which results are delivered and avoid stalls on the expert's side.

3.3 Programming Abstractions

Item

Eureka views data sources as unstructured collections of *items*. In the simplest case, an item refers to a single image in JPEG, PNG or other well-known format stored on the disk. An item may also be a short video segment from a continuously streaming camera, a tile from whole-slide images in digital pathology [24], a region from OpenStreetMap, and other forms of domain-specific multi-dimensional data. An item is the unit of data to be discarded, and to be displayed in the front-end GUI. The appropriate granularity of an item is specific to a task. For example, an object detection task may use individual frames as items, while an activity recognition task may use 10-second video segments as items. The front-end allows the user to configure what constitutes an item before starting a search.

Filter

A *filter* is an abstraction for executing any computer vision code in Eureka. A filter's main function is to inspect items, declare which ones are clearly "irrelevant," and then discard them. Eureka separates its runtime and the filters through a narrow API, so that third-party developers can easily create and "plug-and-play" new filters. A filter uses these APIs to get user-supplied parameters (e.g., example texture patches) for a query. A filter is required to implement a scoring function, `score(item)`, where it examines a given item and outputs a numeric score. The

runtime applies the filter's score function to each item, and if the returned score exceeds a user-provided threshold, the item is deemed to *pass*; otherwise the item is discarded. An early-discard query pipeline consists of multiple filters, with corresponding parameters and passing thresholds. The current system requires an item to pass all of the filters before transmitting and presenting it to the user. This effectively implements the Boolean operator AND across filters. Eureka could easily be extended to support the full range of Boolean operators and expressions. Eureka performs short-circuit evaluation: once an item fails a filter, it is discarded without further evaluation by later filters in a cascade, thus achieving "early" in early discard.

Attribute

A filter can also attach *attributes* to an item as a by-product of scoring. Attributes are key-value pairs that can represent arbitrary data, and are accessed using the `get-attribute(item, key)` function, and can be written using the `set-attribute(item, key, val)` interface. The primary purpose of the attribute abstraction is to facilitate communication between filters, where one filter gets attributes set by another. Attributes are analogous to columns in relational databases but with significant differences. In Eureka, attributes are rarely complete for all items (rows) in the data, both due to early-discard of items in the pipeline and due to fast-aborted searches. Additionally, unlike most databases, where the schema tends to be stable, new attributes may be created rapidly in each new query as the user applies new filters (e.g., a retrained DNN). Finally, the user can designate a set of interesting attributes to be retrieved along with the items. Unwanted attributes are stripped off before Eureka transmits results back to the user to reduce bandwidth demand over the WAN. Returned attributes can be used for aggregations and joins using other tools such as relational databases.

Examples

Table 6 shows several example filters. While some filters output a pass/fail Boolean result (e.g., JPEG decoder), others output a numeric score (e.g., SVM confidence) that can be compared against a threshold. The SVM filter illustrates how attributes enable communication between filters. It uses the `mobilenet_featvec` attribute created by the MobileNet [28] filter, which in turn uses the `rgb` attribute created by the JPEG decoder, forming a chain of *dependency*. This attribute mechanism allows decomposition of complex tasks into independent, manageable, and reusable components, while still adhering to the filter chain abstraction.

3.4 Implementation

Most of the Eureka system runs on cloudlets, reads raw data, and executes queries. Figure 7 depicts the logical execution model, highlighting the major system components, and how the processed data flows through them.

Filter Containers

Eureka encapsulates each filter in its own Docker container to achieve *software generality*. This facilitates use of different software frameworks and libraries concurrently within a single query.

Filter	Synopsis
JPEG decoder	jpeg_decode() → bool Decodes a JPEG image. Set-attributes: `rgb` Returns true if successful, false otherwise.
SIFT matching	sift_match(distance_ratio: float, example: Image) → int Finds matched SIFT keypoints between example and test image. Get-attributes: `rgb` Returns number of matched keypoints.
MobileNet classification	mobilenet_classify(target_class: string, top_k: int) → bool Classifies image into ImageNet classes and test if target_class is in top_k predictions. Get-attributes: `rgb` Set-attributes: `mobilenet_featvec` Returns true if target_class is in top_k predictions of the test image, false otherwise.
SVM	svm(training_data: List⟨Image⟩) → float Train an SVM with the given training set, using MobileNet's 1024-dimensional feature as SVM input. Then use the SVM to classify the test image. Get-attributes: `mobilenet_featvec` Returns probability of the test image being positive.

Each filter has algorithm-specific parameters, get-/set-attributes and return scores. The user can specify a threshold on each filter's return score to drop objects below the threshold.

Table 6: Examples of Eureka Filters

For example in Table 6, the JPEG decoder may be a proprietary library, while SIFT is written in OpenCV, MobileNet in TensorFlow, and SVM in Scikit-learn. Some filters may depend on conflicting versions of software (e.g., of TensorFlow). The use of containers ensures reproducibility and simplifies filter deployment, as there is no need to pre-install all software dependencies of a filter on a back-end machine.

Itemizer

The itemizer reads raw data from its data source, transforms it into an item stream, and injects this stream into the execution pipeline. The simplest case just involves loading individual files from disk. More generally, the itemizer can pre-process the data from its native format and transform it into the items needed for the query. For example, it can take continuous data (e.g., streaming or stored video), and emit multiple separate items at a granularity appropriate for the query (e.g., individual frames for object detection, or overlapping short video clips for activity recognition). In addition to selecting granularity, users can also set the *scope* of the itemizer. This can limit the search to only a subset of underlying data based on meta-data attributes such as geographical location and recorded date or time.

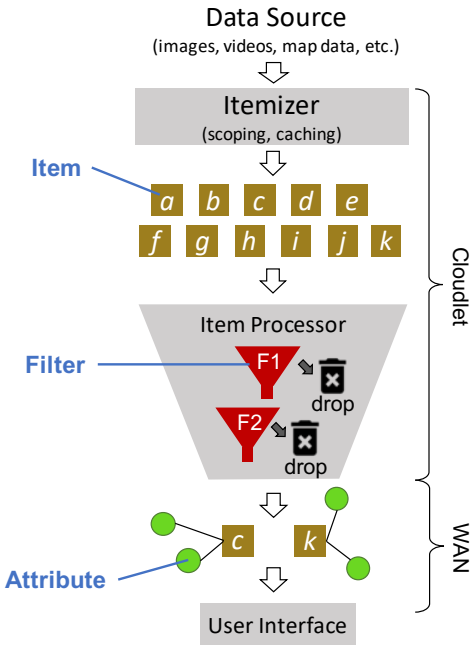


Figure 7: Logical Execution Model

Item Processor

The item processor is responsible for much of the query execution in Eureka. It evaluates the early-discard filters on each item independently, exploiting available data parallelism and multiple cores when available. It uses filter scores and supplied thresholds to decide whether an item should be discarded. As mentioned in Section 3.3, short-circuit evaluation of the filter chain is used to implement early discard. The item processor communicates to the filter containers using a narrow API, of which the three main functions (`score`, `get-attribute` and `set-attribute`) were described earlier. These attributes are retained in memory for access by downstream elements of the filter chain. The narrow-waist API and the use of Docker containers simplify the conversion of off-the-shelf computer vision code into a Eureka filter. Only the items passing all filters (usually only a tiny subset of the entire item stream) along with their attributes are sent to the user.

4 Eureka for Object Detection in Images on the Edge

Cameras are becoming inexpensive and pervasive. A 640x480 VGA CMOS sensor costs as little as $0.65 today [35]. A 2013 survey in the UK estimated one surveillance camera for every 11 people [7]. These cameras capture large volumes of visual data that can be valuable for retrospective or live analysis. A lot of data, however, is stored on the edge dispersed across the Internet for two reasons. First, wide-area network (WAN) bandwidth is scarce and expensive, making it infeasible to transfer all data into the cloud data center. The US national average broadband connectivity is 18.7 Mbps in 2017 [4], barely sufficient to stream a 4K video according to

Figure 8: Examples of Search Targets Used in Case Studies with the YFCC100M data set

Netflix's guidance [42]. Second, privacy concerns and business policies may prevent data being transmitted to the cloud [16]. To address these issues, *edge computing* [52] has been proposed by placing edge nodes, or *cloudlets*, at a location with high-bandwidth and low-latency access to data sources (aka network proximity). Cloudlets may vary in their form factors. Many smart cameras today (e.g., Amazon DeepLens) are effectively cloudlets that are capable of performing sophisticated processing of the captured frames. Others including those known as the fog [8], micro data centers [41], or mobile edge cloud [9] ensure network proximity between compute infrastructure and data source.

4.1 Setup and Methodology

We evaluate Eureka in an edge computing setting using the Yahoo Flickr Creative Commons 100 Million data set (YFCC100M) [58]. As shown in Table 1, YFCC100M has more than 99.2 million unlabeled Flickr photos and is the largest multimedia data set to date. Compared to other curated data sets, YFCC100M preserves the long-tail distribution of uncommon objects in the real world. We evenly partition the data set on SSDs across eight cloudlets, ensuring high bandwidth and low latency. The front-end GUI connects to the cloudlets via the Internet.

We compare Eureka against two alternatives: brute-force and OpenDiamond. With brute-force, the user manually inspects and labels every image. For targets with a low base rate, this requires the user to plow through many images before collecting even a small number of true positives. OpenDiamond [30] improves the situation for the user by using a simple filter chain to perform early discard. These filters are based on the initial set of sample images of the target, and are essentially identical to the queries used in the first iteration of the Eureka experiments. However, there is no use of machine learning or of iterative refinement.

Target	Estimated base rate	Images inspected by user	Positive examples discovered
Deer	0.07 %	7,447	111
Taj Mahal	0.02 %	4,791	105
Fire hydrant	0.005%	15,379	74

Table 7: Summary Results for Case Studies

4.2 Savings of Expert Time

We present results for three experimental case studies. In each, we attempt to build a labeled training data set for a novel target. The three targets chosen are (in descending order of base rate): (1) deer, (2) Taj Mahal, and (3) fire hydrant. Figure 8 gives examples of each target. Although no specialized expertise is needed to identify these targets, they are fairly uncommon in Flickr photos and still serve as an initial proof of the Eureka concept.

Table 7 summarizes the overall results of using Eureka to build a training set of approximately 100 examples for each target. Although the specific numbers are primarily determined by data set characteristics and filter quality, they give an intuition about the targets' scarcity and the human effort spent. We estimate the base rates in YFCC100M based on the meta-data of Flickr tags, titles, and descriptions. Although this estimation is subject to inclusion error (tag without actual target) and exclusion error (target without tag), it provides at least a crude estimate of the targets' prevalence and their relative rarity to each other. The meta-data is only used in analysis of the results and *not* used in the Eureka search process.

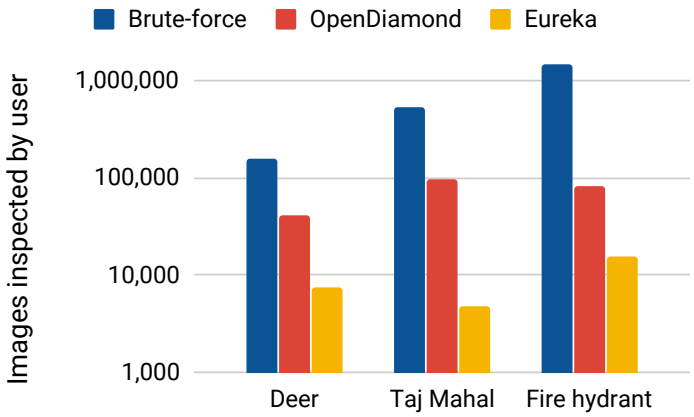
Figure 9 (Y-axis in log scale) compares Eureka to alternatives in the number of images the user has to inspect when building a training set of given size. For brute-force, this number is extrapolated using the estimated base rate. For OpenDiamond, this is based on the precision of the filters used in the first Eureka iteration. We see that OpenDiamond reduces demand for human attention by an order of magnitude over brute-force. Eureka reduces demand by a further order of magnitude. The detailed results of individual case studies, including breakdowns of the iterations following the Eureka approach, have been reported in [22].

4.3 Importance of Proximity to Data

Eureka includes more cloudlets in the search when there is extra expert attention, i.e., the expert is waiting. When the data sources are dispersed across the Internet, edge computing is key to achieving high efficiency without stressing the WAN. The proximity of cloudlets to data sources is crucial — typically providing LAN connectivity (1 Gbps or higher) to an archival data source.

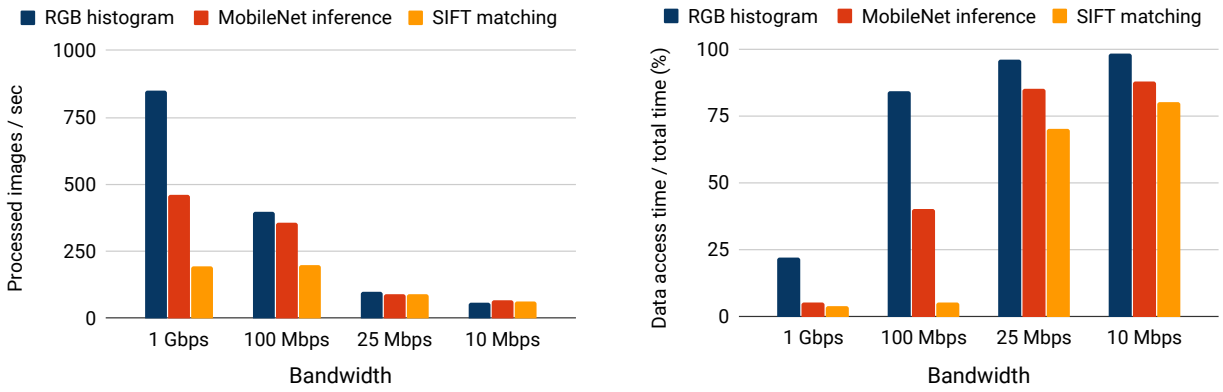
To study the importance of proximity, we throttled the bandwidth between cloudlets and their data sources. We ran experiments at 1 Gbps, 100 Mbps, 25 Mbps and 10 Mbps. The US national average broadband value of 18.7 Mbps in 2017 [4] lies towards the lower end of this range.

We selected three filters to benchmark (ordered by increasing cost in terms of computation time): RGB histogram, MobileNet inference, and SIFT matching. Figure 10(a) reports the processing throughput on the cloudlets (processed images per second, higher is better) as we



This graph shows the estimated amount of human attention needed using three different approaches to acquire a training set of fixed size (111 images of deer, 105 images of Taj Mahal, and 74 images of Fire hydrant). Note that Y-axis is in log scale.

Figure 9: Number of Images Presented to User



(a) Image Processing Rate

(b) Fraction of Time Spent on Data Access

Figure 10: Effect of Bandwidth between Cloudlet and Data Source

decrease the bandwidth from 1 Gbps to 10 Mbps. At 1 Gbps, the RGB histogram filter achieves significantly higher throughput than MobileNet and SIFT because it is the least computationally expensive. As the bandwidth decreases, its throughput decreases drastically. SIFT, the most expensive filter, is still bound by computation at 100 Mbps, but also suffers from low bandwidth starting at 25 Mbps. Under 25 Mbps, there is only marginal difference in throughput among the three filters, implying data access has become the bottleneck for all of the filters. This can be confirmed in Figure 10(b), where we measure the percentage of total run time spent on retrieving data as opposed to computation (lower is better). At the lowest bandwidth of 10 Mbps, even the most computationally expensive SIFT filter spends 80% of its run time retrieving data, and the RGB filter spends 98%! This confirms the importance of network proximity between cloudlet and data source.

Edge	12-core Intel Xeon E5-1650 @ 3.6GHz (total 43 GHz)	32 GB RAM
Cloud	16-core Intel Xeon Platinum 8175M @ 2.50GHz (total 42 GHz)	64 GB RAM

Table 8: Hardware configurations used in edge and cloud (AWS) experiments.

5 Eureka on the Cloud

While the previous section addresses the emerging edge computing paradigm, many other data sets have historically been concentrated into the cloud. Besides, some organizations may afford to gather their visual data in a private cloud. Hence, it will be valuable for Eureka to be able to search over data in the cloud efficiently.

A common design in cloud computing is the separation of storage layer and compute layer. This is exemplified by the Elastic MapReduce (EMR) in Amazon Web Services (AWS). EMR treats the AWS Simple Storage Service (S3) as first-class data source. When the user launches a job, an EMR cluster backed by AWS Elastic Compute Cloud (EC2) machines is instantiated just-in-time. It then reads input from S3, executes the job, writes results back to S3, and tears down the EC2 machines when finished. The existence of the EC2 machines along with their virtual disks (aka Elastic Block Store or EBS) is transient. This separation provides efficient scaling, consistency, and data preservation. Nonetheless, the separate storage layer may exhibit different performance traits than local disks in the edge setting.

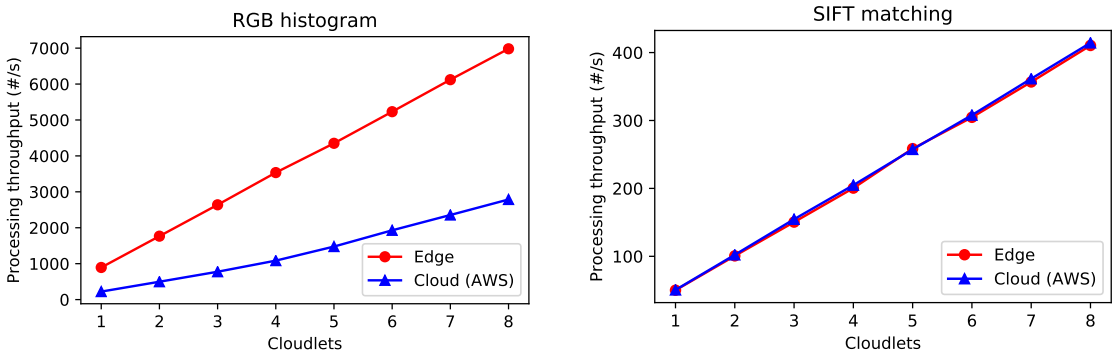
5.1 Setup and Methodology

We use eight EC2 instances on AWS as cloudlets in the cloud setting. Since there is no exact match on AWS with our edge machines, we choose the closest type we can get (m5.4xlarge). Table 8 gives the hardware configurations used in our edge and cloud experiments. Both settings have roughly the same total CPU clock rates.

We follow Section 4 to use the YFCC100M data set. However, in this experiment, the data is stored on a public S3 bucket. In other words, it is not stored on the individual EBS volumes associated with the EC2 machines, which makes it easy to scale the number of cloudlets. At run time, each cloudlet processes a disjoint subset of the data set according to a meta data file stored on their local volumes.

5.2 Results

Figure 11 reports the processing throughput in the edge and in the cloud settings when we change the number of cloudlets. We measure a cheap RGB histogram filter and an expensive SIFT matching filter. Both settings scale nicely up to at least eight cloudlets. The gap in Figure 11(a) is due to the latency and bandwidth limit between a single EC2 instance and S3. It manifests that AWS S3 is, as advertised, optimized for scalability rather than latency. The SIFT matching filter in Figure 11(b) is CPU-bound. Since there are sufficient images to parallelise over the many cores, its throughput is limited by the total CPU clock rate, for which both the edge and the cloud have approximately the same number.



(a) RGB histogram (cheap)

(b) SIFT matching (expensive)

Figure 11: Processing Throughput When Scaling the Number of Cloudlets.

6 Eureka on Smart Disks (Emulated)

The concept of "early" in "early discard" refers to the processing pipeline from data storage (disk or SSD) through various stages of server hardware, operating systems, and application layers, to transmission across the Internet, further processing at the client operating system and applications layers, and eventually the human expert. Discarding data as early as possible throughout this pipeline improves the efficiency of the system. Arguably, the disk or SSD is at the extreme end in this pipeline. I will explore the use of *active disk* [3], also referred to as *intelligent storage* [36] or *near-data processing* [6]. The concept refers to executing data processing code on the controller within the storage device. It has the advantage of lower-latency access to data, lower energy cost per operation, reduced data movement across the system bus, and freeing up the main processor's cycles for other operations. With the well-known stagnation of single-core clock speed, and the growth of data outpacing various aspects of computing infrastructure (e.g., the ability of putting more cores on a chip, DRAM speed, network bandwidth, etc.), it carries great potential to utilize active storage devices to meet the demand for scalability. Low compute capability, heterogeneous hardware, and difficulty of programming have been the major obstacles to the adoption of these devices. Prior work has explored exploiting intelligent SSDs for classic big data workloads, such as key-value store [17], relational database [19], and MapReduce [14]. It remains unexplored how to utilize active storage for analyzing visual data. I plan to address this issue by developing image filters to run on (emulated) intelligent disk devices.

7 Eureka for Activity Recognition in Videos

Compared to images, videos provide an extra temporal dimension of information, enabling recognition of activities. For example, only by looking at an ordered sequence of frames can one distinguish a person doing jumping jacks from waving hands. So far, research in video understanding has focused on human activity recognition and is still at the nascent stage. Learning temporal knowledge from videos is uniquely difficult. Besides, at a common rate of 30 FPS, merely a few seconds' video clip contains a large number of frames, making the DNNs extremely

Figure 12: Left: example frames from the UCF-101 data set (source: [56]). Right: example frame from a deployed surveillance camera.

computationally expensive.

I plan to evaluate Eureka using surveillance videos stored on the edge. This represents the valuable use case of analyzing surveillance cameras dispersed geographically for law enforcement or business analytics. Particularly, I will focus on high-resolution and broad-view cameras, a rapidly growing class of data sources of this kind. These characteristics pose additional challenges. First, the data looks very different from prior publicly available human action data sets, hindering the generalization of pre-trained DNN models. Figure 12 compares example frames from a well-known data set (UCF-101) with a surveillance camera deployed in the real world. Surveillance data has high resolutions, tiny subjects, and a lot of noise and irrelevant background, making it both computationally and algorithmically challenging. Second, real-world queries often contain complex search conditions rather than a single action label, such as "search for a man wearing red shirt running after a child on the street." The system thus needs to execute and combine filters on video-, frame- and patch-levels. This may call for new programming abstractions and optimization in Eureka in addition to those described in Section 3.

8 Eureka for Other Multi-dimensional Data and Queries

The effectiveness of Eureka generalizes beyond commonplace visual data. To validate this, I plan to evaluate Eureka using multi-dimensional data stored on the edge. Among others, two data types are particularly interesting: (1) whole slide images used in digital pathology [24]. These images are extremely huge, with one image sizing several Gigabytes. They thus typically provide a multi-resolution pyramid view into the data. How to incorporate these exorbitantly large images and the pyramid view into Eureka will be a challenge. (2) high definition map data, especially that popularized by self-driving cars. This data is intrinsically geometric and tightly intra-connected, posing challenges in terms of data model, filtering, visualization, and so on. Many questions remain open: What is the "killer app" of analyzing HD maps with machine learning?; How to apply deep learning techniques developed for RGB images on maps?; What is the appropriate interface for user to select, query, and visualize map data? I will explore these questions as part of my proposed work.

9 Timeline

Time		Plan
2019 Summer	May – Aug	Implementation of Eureka on emulated smart disks
2019 Fall	Sept	FAST submission
	Oct – Nov	Implementation of Eureka for video
	Dec	MobiSys submission
2020 Spring	Jan – Apr	Implementation of Eureka for other multi-dimensional data
	May	SEC submission
2020 Summer	May – Aug	Thesis writing
2020 Fall	Sept – Nov	Finish thesis dissertation
	Dec	Thesis defense

Bibliography

[1] TensorFlow. https://www.tensorflow.org/, 2019. 2.1

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org. 1

[3] A. Acharya, M. Uysal, and J. Saltz. Active Disks: Programming Model, Algorithms and Evaluation. In *Proceedings of Architectural Support for Programming Languages and Operating Systems*, 1998. 6

[4] Akamai. Q1 2017 state of the Internet / connectivity report. 2017. 4, 4.3

[5] Brandon Amos, Bartosz Ludwicznik, and Mahadev Satyanarayanan. OpenFace: A general-purpose face recognition library with mobile applications. Technical Report CMU-CS-16-118, School of Computer Science, Carnegie Mellon University, June 2016. 1.2

[6] Rajeev Balasubramonian, Jichuan Chang, Troy Manning, Jaime H Moreno, Richard Murphy, Ravi Nair, and Steven Swanson. Near-data processing: Insights from a micro-46 workshop. *IEEE Micro*, 2014. 6

[7] David Barrett. One surveillance camera for every 11 people in britain, says cctv survey. Dayli Telegraph, July 2013. 4

[8] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012. 4

[9] Gabriel Brown. Converging telecom & IT in the LTE RAN. 2013. 4

[10] Christopher Canel, Thomas Kim, Giulio Zhou, Conglong Li, Hyeontaek Lim, David G. Andersen, Michael Kaminsky, and Subramanya R. Dulloor. Scaling video analytics on constrained edge nodes. In *SysML*, 2019. 2.1

[11] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015. 2.1

[12] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. Glimpse: Continuous, Real-Time Object Recognition on Mobile Devices. In *Proceedings of ACM SenSys*, 2015. 2.1

[13] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, pages 2285–2294, 2015. 2.1

[14] Benjamin Y Cho, Won Seob Jeong, Doohwan Oh, and Won Woo Ro. XSD: Accelerating

mapreduce by harnessing the gpu inside an ssd. In *Proceedings of the 1st Workshop on Near-Data Processing*, 2013. 6

[15] Intel Corp. `https://www.movidius.com/myriadx`, 2019. 1, 2.1

[16] Nigel Davies, Nina Taft, Mahadev Satyanarayanan, Sarah Clinch, and Brandon Amos. Privacy mediators: Helping iot cross the chasm. In *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications*. ACM, 2016. 4

[17] Arup De, Maya Gokhale, Rajesh Gupta, and Steven Swanson. Minerva: Accelerating data analysis in next-generation ssds. In *2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2013. 6

[18] Jia Deng, Jonathan Krause, and Li Fei-Fei. Fine-grained crowdsourcing for fine-grained recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2013. 2.2

[19] Jaeyoung Do, Yang-Suk Kee, Jignesh M Patel, Chanik Park, Kwanghyun Park, and David J DeWitt. Query processing on smart ssds: opportunities and challenges. In *Proceedings of the 2013 ACM SIGMOD*. ACM, 2013. 6

[20] Debidatta Dwibedi, Ishan Misra, and Martial Hebert. Cut, paste and learn: Surprisingly easy synthesis for instance detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1301–1310, 2017. 2.2

[21] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2): 303–338, June 2010. 1.2

[22] Ziqiang Feng, Shilpa George, Jan Harkes, Padmanabhan Pillai, Roberta Klatzky, and Mahadev Satyanarayanan. Edge-based discovery of training data for machine learning. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018. 4.2

[23] Frederick Feyrer, Donald Portz, Darren Odum, Ken B Newman, Ted Sommer, Dave Contreras, Randall Baxter, Steven B Slater, Deanna Sereno, and Erwin Van Nieuwenhuyse. Smeltcam: Underwater video codend for trawled nets with an application to the distribution of the imperiled delta smelt. *PloS one*, 8(7):e67829, 2013. 1.3

[24] Adam Goode, Benjamin Gilbert, Jan Harkes, Drazen Jukic, and Mahadev Satyanarayanan. Openslide: A vendor-neutral software foundation for digital pathology. *Journal of Pathology Informatics*, September 2013. 3.3, 8

[25] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. MCDNN: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 123–136. ACM, 2016. 2.1

[26] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015. 2.1

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for

image recognition. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, pages 770–778, 2016. 1, 2.1

[28] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 1, 2.1, 3.3

[29] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B Gibbons, and Onur Mutlu. Focus: Querying large video datasets with low latency and low cost. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 269–286, 2018. 2.1

[30] Larry Huston, Rahul Sukthankar, Rajiv Wickremesinghe, Mahadev Satyanarayanan, Gregory R Ganger, Erik Riedel, and Anastassia Ailamaki. Diamond: A storage architecture for early discard in interactive search. In *Proceedins of USENIX Conference on File and Storage Technologies*, 2004. 1, 2, 1.3, 2, 3, 4.1

[31] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016. 2.1

[32] Angela H Jiang, Daniel L-K Wong, Christopher Canel, Lilia Tang, Ishan Misra, Michael Kaminsky, Michael A Kozuch, Padmanabhan Pillai, David G Andersen, and Gregory R Ganger. Mainstream: Dynamic stem-sharing for multi-tenant video processing. In *2018 USENIX Annual Technical Conference USENIX ATC 18)*, pages 29–42, 2018. 2.1

[33] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. Chameleon: scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 253–266. ACM, 2018. 2.1

[34] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017. 1, 2.1

[35] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. Noscope: optimizing neural network queries over video at scale. *Proceedings of the Very Large Data Bases*, 2017. 2.1, 4

[36] K. Keeton, D. Patterson, and J. Hellerstein. A Case for Intelligent Disks (IDISKs). *ACM SIG on Management of Data Record*, 1998. 6

[37] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012. 1, 1.2, 2.1

[38] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision*. Springer, 2014. 1, 1.2

[39] Spencer K Lynn and Lisa Feldman Barrett. 'Utilizing' signal detection theory. *Psychological science*, 2014. 1.3, 3.2

[40] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016. 2.1

[41] Rich Miller. AOL gets small with outdoor micro data centers. `https://www.datacenterknowledge.com/archives/2012/07/06/aol-micro-data-centers`, Jul 2012. 4

[42] Netflix. Internet connection speed recommendations. `https://help.netflix.com/en/node/306`, 2017. 4

[43] An Thanh Nguyen, Byron C Wallace, and Matthew Lease. Combining crowd and expert labels using decision theoretic active learning. In *Third AAAI conference on human computation and crowdsourcing*, 2015. 2.2, 2.2

[44] Sinno Jialin Pan, Qiang Yang, et al. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 2010. 1

[45] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS-W*, 2017. 1, 2.1

[46] Pranav Rajpurkar, Jeremy Irvin, Aarti Bagul, Daisy Ding, Tony Duan, Hershel Mehta, Brandon Yang, Kaylie Zhu, Dillon Laird, Robyn L Ball, et al. MURA: Large dataset for abnormality detection in musculoskeletal radiographs. *arXiv preprint arXiv:1712.06957*, 2017. 1.3

[47] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment*, 11(3):269–282, 2017. 2.2, 2.2

[48] Alexander J Ratner, Henry Ehrenberg, Zeshan Hussain, Jared Dunnmon, and Christopher Ré. Learning to compose domain-specific transformations for data augmentation. In *Advances in neural information processing systems*, pages 3236–3246, 2017. 2.2

[49] Luis Remis, Vishakha Gupta-Cledat, Christina Strong, and Ragaad Altarawneh. Vdms: An efficient big-visual-data access for machine learning workloads. In *Workshop on Systems for Machine Learning and Open Source Software at NeurIPS*, 2018. 2.1

[50] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, 2015. 1, 1.2, 2.1

[51] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 2015. 1, 1.2

[52] Mahadev Satyanarayanan. The Emergence of Edge Computing. *IEEE Computer*, 50(1), January 2017. 4

[53] Gunnar A. Sigurdsson, Gül Varol, Xiaolong Wang, Ali Farhadi, Ivan Laptev, and Abhinav Gupta. Hollywood in homes: Crowdsourcing data collection for activity understanding. In *European Conference on Computer Vision*, 2016. 1, 1.2, 2.2

[54] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems*, pages 568–576, 2014. 1.2

[55] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 2.1

[56] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012. 1.2, 12

[57] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, pages 1–9, 2015. 2.1

[58] Bart Thomee, David A Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. YFCC100M: the new data in multimedia research. *Communications of the ACM*, 2016. 1.2, 1.3, 2, 4.1

[59] Jack Valmadre, Luca Bertinetto, João Henriques, Andrea Vedaldi, and Philip HS Torr. End-to-end representation learning for correlation filter based tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2805–2813, 2017. 2.1

[60] Sudheendra Vijayanarasimhan and Kristen Grauman. Large-scale live active learning: Training object detectors with crawled data and crowds. *International Journal of Computer Vision*, 2014. 1, 2.2, 2.2

[61] Junjue Wang, Ziqiang Feng, Zhuo Chen, Shilpa George, Mihir Bala, Padmanabhan Pillai, Shao-Wen Yang, and Mahadev Satyanarayanan. Bandwidth-efficient live video analytics for drones via edge computing. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 159–173. IEEE, 2018. 2.1

[62] Eric P Xing, Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. Petuum: A new platform for distributed machine learning on big data. *IEEE Transactions on Big Data*, 1(2):49–67, 2015. 2.1

[63] Tiantu Xu, Luis Materon Botelho, and Felix Xiaozhu Lin. VStore: A data store for analytics on large videos. In *Proceedings of the Fourteenth EuroSys Conference 2019*, EuroSys '19, pages 16:1–16:17, 2019. doi: 10.1145/3302424.3303971. URL `http://doi.acm.org/10.1145/3302424.3303971`. 2.1

[64] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. Live video analytics at scale with approximation and delay-tolerance. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 377–392, 2017. 2.1