

EFFICIENT AND ACCURATE SYSTEMS FOR QUERYING UNSTRUCTURED
DATA

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Daniel Kang
December 2022

© 2022 by Daniel Kang. All Rights Reserved.

Re-distributed by Stanford University under license with the author.



This work is licensed under a Creative Commons Attribution-3.0 United States License.

<http://creativecommons.org/licenses/by/3.0/us/>

This dissertation is online at: <https://purl.stanford.edu/fk030tb6783>

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Matei Zaharia, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Tatsunori Hashimoto, Co-Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Peter Bailis

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Christos Kozyrakis

Approved for the Stanford University Committee on Graduate Studies.

Stacey F. Bent, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format.

Abstract

Volumes of unstructured, non-tabular data (e.g., videos, audio, and text) have been increasing exponentially. This data is exciting to scientific researchers, business analysts, and data scientists for downstream analyses. For example, video can be used by urban planners to analyze traffic, ecologists to understand hummingbird-bacteria microcosms, and data scientists to analyze customer behavior in stores. However, this is impossible to do manually at scale: exabytes of data are generated per day, outstripping manual processing capacity.

In recent years, *automatic* analysis over this unstructured data has become possible via machine learning (ML). Analysts can use ML to extract structured information from these unstructured sources, such as object types and location from a video. The structured information can subsequently be used in downstream analysis, e.g., the urban planner can count the number of cars that passed by an intersection.

Unfortunately, using ML for these analyses is challenging. Deploying ML is prohibitively expensive for many organizations: naively analyzing a year of video from a small town can cost millions in cloud compute credits. ML methods are also unreliable, returning incorrect results, which can lead to downstream errors. Finally, deploying ML for analytics requires knowledge of deep learning, data systems, programming, and other technical skills.

In light of these challenges, we make two observations: many applications can tolerate approximations, *if there are guarantees on accuracy*, and methods for answering unstructured data queries range by up to 10 orders of magnitude in cost.

In this dissertation, we develop systems and algorithms for efficient and reliable unstructured data analytics, leveraging the two observations. Instead of returning exact answers, we return approximate answers generated by cheap approximations to expensive ML methods. Our systems can return statistically valid answers on a wide range of query types, including selection, aggregation, and limit queries. Furthermore, our systems can be up to *orders of magnitude* cheaper than standard methods of answering queries.

We further develop systems for monitoring and quality assurance over ML pipelines. In addition to being deployed for analytics, ML is increasingly being deployed in mission-critical settings, such as in autonomous vehicles. Despite being deployed in these settings, models are often unmonitored and the training data is often not vetted.

To address this, we propose abstractions for monitoring and quality assurance of ML deployments: model assertions and learned observation assertions. These assertions allow domain experts to specify errors, both at deployment time and over the data used to train these models. Assertions can find errors with both high recall (75%) and high precision (100%) in real-world autonomous vehicle, video analytics, and medical datasets.

The systems and abstractions in this dissertation have been deployed in a variety of real-world settings, including for autonomous vehicles and ecological analysis.

Acknowledgements

Many acknowledgements sections begin by saying there are too many to list, which I have been confused by. After attempting to write my acknowledgements, I realize there are too many people who have helped in my PhD and my memory is lacking. These acknowledgements are a small sample of those who have made my PhD possible.

I've had the great pleasure of being advised by three professors: Peter Bailis, Tatsunori Hashimoto, and Matei Zaharia. I began my PhD in the rotation program, unsure if I even wanted to do a PhD. My first rotation wasn't a good fit and I was considering leaving the program. Fortunately, several of my friends told me to stay for another rotation. That rotation was with Peter and Matei.

Peter and Matei both have incredible energy, albeit in different ways. When I first started, Peter and Matei stayed with me in lab late into the night working with me on my first paper draft. That draft eventually became my first full paper and one of my few papers that was accepted on first submission. Peter also worked with me on creating my first conference presentation over the course of two days at my first conference. I've learned so much by working with Peter and Matei in my six years at Stanford.

Their generosity also deserves its own mention. In the middle of my PhD, my mother became gravely ill and I had to take an extended period of time off. I was expecting to take a leave of absence. After all, which job lets you take off months at a time? But when I spoke to them about taking leave, they told me I could stay enrolled. It wasn't even a question to them.

I've also had the fortune of working with Tatsunori Hashimoto at my time at Stanford. I first met Tatsu at MIT, when I was doing undergraduate research. Tatsu took a bet on me and was the first person to actually teach me how to do research. After he started at Stanford, I began to work with him closely on the statistical aspects of my work. I didn't understand how useful complementary expertise could be until after I worked with Tatsu.

Aside from my advisors, I've had the opportunity to work with a number of other researchers. I've co-authored with Firas Abuzaid, Nikos Arechiga, Peter Bailis, Cody Coleman, Alex Derhacobian, John Emmons, Tadashi Fukami, Edward Gan, John Guibas, Tatsunori Hashimoto, Trevor Hebert, Animesh Koratana, Christos Kozyrakis, Peter Kraft, Ankit Mathur, Deepak Narayanan, Luigi Nardi, Kunle Olukotun, Shoumik Palkar, Sudeep Pillai, Deepti Raghavan, Francisco Romero, Chris Ré, Kaoru Tsuji, Teja Veeramacheneni, Matei Zaharia, Jian Zhang, and Tian Zhao. Time flies when you're working with such great colleagues and the six years have flown by.

I wouldn't have been as productive as I had without the support of my labmates. I've had the fortune to co-author with several of them, including one hail mary paper to travel to Japan (the paper got in, but our plans were scuttled due to COVID). But aside from that, my labmates have given me amazing feedback on my papers, talks, and research in general. My original office, Gates 433, with Edward Gan, Kexin Rong, and Sahaana Suri, was especially supportive. I also want to thank my wonderful labmates Firas Abuzaid, Lingjiao Chen, Cody Coleman, Trevor Gale, Fiodar Kazhamiaka, Omar Khattab, Peter Kraft, Shoumik Palkar, Deepti Raghavan, Keshav Santhanam, Kai Sheng Tai, Pratiksha Thaker, James Thomas, and Gina Yuan.

During my time at Stanford and beyond, I've had incredible support from my friends. If it weren't for them, I'm not sure that I would have even stayed to complete my PhD! There are simply too many to thank and I'm not sure if I can even list them all. My friends from all stages of my life have helped me immensely. Many of them I've known since high school, before I even started research: Andrew Cheong, the Gerr sisters, Manna, Siddharth, Stephen, Olivia, and many others. I've met so many people through the different stages of my life since then: Alex and Leon (apologies for the link spam), Cole and Hannah (I'll shamelessly take credit for introducing them), John, Alice, Anna, and Chris (thanks for putting up with my hot takes in book club), Max, Sumit, Judy, Tong, and many others.

And finally, I wouldn't be here with my family. My family has supported me long before my PhD even began. Even in the depth of her illness, my mother told me to focus on my research. While she won't see the publication of this thesis I know she would be proud.

Contents

Abstract	iv
Acknowledgements	vi
1 Introduction	1
1.1 Challenges to Unstructured Data Analytics	2
1.1.1 The High Cost of Unstructured Data Analytics	2
1.1.2 The Unreliability of Unstructured Data Analytics	3
1.1.3 The Programming Difficulty of Unstructured Data Analytics	4
1.2 Efficient and Reliable Unstructured Data Queries	5
1.2.1 Efficient ML-based Queries	5
1.2.2 Efficient and High Quality Proxy Score Generation	7
1.2.3 Monitoring and Quality Assurance	8
1.3 Organization	10
2 Background	11
2.1 Example Applications	11
2.1.1 Traffic Analysis	11
2.1.2 Ecological Analysis	12
2.1.3 News Analysis	12
2.2 Deploying ML for Analytics	13
2.3 Related Work	14
2.3.1 Approximate Query Processing	14
2.3.2 Retrieval	14

3	Proxy-based Algorithms and Systems	16
3.1	NOSCOPE	17
3.1.1	NOSCOPE Architecture and Techniques	19
3.1.2	Model Specialization	20
3.1.3	Evaluation	22
3.1.4	Discussion	23
3.2	Approximate Selection with Guarantees	24
3.2.1	Use Cases	27
3.2.2	Approximate Selection Queries	29
3.2.3	Algorithm Overview	32
3.2.4	Estimating proxy thresholds	34
3.2.5	Evaluation	42
3.2.6	Discussion	47
3.3	Approximate Aggregation with Predicates	48
3.3.1	Overview and Query Semantics	51
3.3.2	Query Formalism	52
3.3.3	Algorithm Description and Query Processing	53
3.3.4	Theoretical Analysis	56
3.3.5	Evaluation	61
3.3.6	Discussion	65
3.4	BLAZEIT	65
3.4.1	BLAZEIT System Overview	67
3.4.2	FRAMEQL: Expressing Complex Spatiotemporal Visual Queries	68
3.4.3	Query Optimizer Overview	71
3.4.4	Optimizing Aggregates	72
3.4.5	Optimizing Limit Queries	76
3.4.6	Evaluation	78
3.4.7	Discussion	84
4	Generating Proxy Scores Efficiently	85
4.1	TASTI: Semantic Indexes for Unstructured Data	85
4.1.1	Overview and Example	87
4.1.2	Index Construction	91

4.1.3	Query Processing with TASTI	94
4.1.4	Theoretical Analysis	97
4.1.5	Evaluation	99
4.1.6	Discussion	105
4.2	SMOL: Hardware Efficient Proxy Generation	106
4.2.1	Measurement Study of End-to-End DNN Inference	109
4.2.2	SMOL Overview	111
4.2.3	Cost Modeling for Visual Analytics	114
4.2.4	Input-aware Methods for Accuracy and Throughput Trade Offs	118
4.2.5	An Optimized Runtime Engine for End-to-End Visual Inference	120
4.2.6	Evaluation	124
4.2.7	Discussion	129
5	Specifying Errors in ML Deployments	130
5.1	Model Assertions	130
5.1.1	Model Assertions	133
5.1.2	Using Model Assertions for Active Learning with BAL	137
5.1.3	Consistency Assertions and Weak Supervision	139
5.1.4	Evaluation	142
5.1.5	Discussion	147
5.2	Learned Observation Assertions	147
5.2.1	Example and Background	150
5.2.2	System Overview	152
5.2.3	Learned Observation Assertions	154
5.2.4	Feature Distributions	156
5.2.5	Scoring Relative Plausibility	158
5.2.6	Applications	159
5.2.7	Evaluation	161
5.2.8	Discussion	166
6	Discussion	167
6.1	Future Directions	168

List of Tables

1.1	Costs of executing queries over unstructured data via self-hosted methods, an ML service, and using human annotators compared to the cost of executing a structured query over similar data.	3
3.1	Video streams and object labels queried in our evaluation.	22
3.2	Notation Summary	34
3.3	Summary of datasets, oracle models, proxy models, and true positive rates.	42
3.4	Summary of distributionally shifted datasets. These shifts are natural (weather related, different day of video) and synthetic.	44
3.5	Achieved accuracy of queries when using the empirical cutoff method and SUPG on data with distributional shift. The naive algorithm <i>deterministically fails</i> to achieve the targets, i.e., has a failure rate of 100%.	46
3.6	Summary of notation.	53
3.7	FRAMEQL’s data schema contains spatiotemporal and content information related to objects of interest, as well as metadata (class, identifiers).	69
3.8	Additional syntactic elements in FRAMEQL.	70
3.9	A comparison of object detection methods, filters, and speeds. More accurate object detection methods are more expensive.	71
3.10	Video streams and object labels queried in our evaluation.	79
3.11	Average error of 3 runs of query-rewriting using a specialized NN for counting. These videos stayed within $\epsilon = 0.1$	82
3.12	Estimated and true counts for specialized NNs run on two different days of video. In parentheses are the day of video.	82
3.13	Query details and number of instances for limit queries.	83

4.1	Throughput of ResNet-50 on the T4 with three different execution environments. Keras was used in [11]. The efficient use of hardware can result in over a 17× improvement in throughput.	110
4.2	Throughput of various models on the T4 GPU (classification models on the top and detection models on the bottom) [178]. As shown, all but the largest, state-of-the-art detection models are preprocessing bound.	112
4.3	Throughput and top-one accuracy for ResNets of different depths. As shown, there is a trade off between accuracy and throughput (i.e., computation).	115
4.4	Measurements of preprocessing, DNN execution, and pipelined end-to-end DNN inference for three configurations of DNNs and input formats: balanced, preprocessing-bound, and DNN-execution bound. As shown, SMOL matches or ties the most accurate estimate for all conditions.	115
4.5	A list of popular visual data formats and their low-fidelity features.	124
4.6	Summary of dataset statistics for the still image datasets we used in our evaluation. The datasets range in difficulty and number of classes. <code>bike-bird</code> is the easiest dataset to classify and <code>imagenet</code> is the hardest to classify.	125
4.7	Effect of training procedure and input format on accuracy for ResNet-50 and ResNet-34 on <code>imagenet</code> . SMOL can achieve an accuracy throughput trade-offs by changing the input format, achieving no accuracy loss for easier datasets.	128
5.1	A summary of tasks, models, and assertions used in our evaluation.	142
5.2	Number of lines of code (LOC) for each assertion. All assertions could be written in under 60 LOC including helper functions. The assertion main body could be written in under 25 LOC in all cases.	144
5.3	Precision of model assertions deployed on 50 randomly selected examples. Model assertions can be written with 88-100% precision across all domains.	144
5.4	Accuracy of pretrained and weakly supervised models. Weak supervision can improve accuracy with no human-generated labels.	147
5.5	Table of syntactic elements in FIXY’s DSL.	156
5.6	Description of features we used in this evaluation. Model only and count were manually specified features.	162
5.7	Precision at top 10, 5, and 1 of FIXY and ad-hoc MA baselines for finding tracks missed by humans. FIXY outperforms baselines by up to 2×.	163

List of Figures

1.1	A confident error from a state-of-the-art model for detecting birds. The confidence is in the top 99.9th percentile in the dataset.	4
1.2	An object detection model erroneously missing the prediction of the car in the second frame. Even widely used models (SSD) fail simple checks, such as temporal consistency.	8
1.3	A motorcycle missed by a human annotator in a self-driving car dataset. The dataset was generated by a leading label vendor, showing that even best-in-class services can have errors in “gold” labels.	9
3.1	NOSCOPE is a system for accelerating neural network analysis over videos via inference-optimized model search. Given an input video, target object, and reference neural network, NOSCOPE automatically searches for and trains a cascade of models that can reproduce the binarized outputs of the reference network with high accuracy—but up to three orders of magnitude faster. . . .	20
3.2	Accuracy vs. speedup achieved by NOSCOPE on each dataset. Accuracy is the percent of correctly labeled time windows, and speedup is over YOLOv2.	23
3.3	Box plot of achieved precisions of naive sampling from recent work [94, 124] and our improved algorithm. The naive algorithm returns precisions as low as 65% for over half the runs. In contrast, our algorithms (SUPG) achieve the precision target w.h.p.	25
3.4	Sample matching (a) and non-matching (b) frames for a selection query over a video stream used by our biologist collaborators. DNNs can serve as proxies to identify hummingbirds as shown in (a), but the confidence scores can be unreliable.	27
3.5	Syntax for specifying approximate selection queries.	29

3.6	SUPG algorithms use sampled oracle labels and proxy scores to identify a subset of records that satisfy a recall or precision target with high probability. Naive methods use limited oracle samples less efficiently and can fail to achieve the target recall or precision.	32
3.7	Precision box-plot of 100 trials of U-NoCI and SUPG’s importance sampling algorithm with a precision target of 90%. U-NoCI can fail up to 75% of the time. Furthermore, it can return precisions as low as 20%.	45
3.8	Recall of 100 trials of U-NoCI and SUPG’s sampling algorithm with a recall target of 90%. U-NoCI can fail up to 50% of the time and even catastrophically fail on ImageNet, returning a recall of as low as 20%.	45
3.9	Targeted precision vs achieved recall. Both importance sampling methods outperform U-CI in all cases. Two-stage importance sampling outperforms all methods and matches the one-stage importance sampling for ImageNet.	47
3.10	Targeted recall vs precision of the returned set. Up and to the right indicates higher performance. Importance sampling outperforms or matches U-CI in all cases. Our sqrt scaling outperforms proportional scaling for importance sampling in all cases, except for high recall settings.	48
3.11	Syntax for ABAE. Users provide a statistic to compute, an expensive predicate, an oracle limit, proxy scores, and a success probability. As is standard for aggregation queries, users may specify a group by key.	52
3.12	Sampling budget vs RMSE for uniform sampling and ABAE. ABAE outperforms on all budgets and datasets we evaluated on. ABAE can outperform by up to 1.5× on RMSE at a fixed budget and achieve the same error with up to 2× fewer samples.	63
3.13	Sampling budget vs normalized Q-error for uniform sampling and ABAE, with the standard deviation shaded. We see that ABAE outperforms on Q-error. The same trends hold for all other datasets.	64
3.14	Sampling budget vs CI width for uniform sampling and ABAE. ABAE can outperform by up to 1.5× on CI width at a fixed budget and achieve the same width with up to 2× fewer samples.	64
3.15	Schematic of the naive method of querying video and BLAZEIT. BLAZEIT does not require writing complex code and does not require pre-materializing all the tuples.	66

3.16	FRAMEQL syntax. As shown, FRAMEQL largely inherits SQL syntax.	70
3.17	End-to-end runtime of aggregate queries where BLAZEIT rewrote the query with a specialized network, measured in seconds (log scale). BLAZEIT outperforms all baselines. All queries targeted $\epsilon = 0.1$	81
3.18	Sample complexity of random sampling and BLAZEIT with control variates. Control variates via specialized NNs consistently outperforms standard random sampling. Note the y-axis is on a log scale.	82
3.19	End-to-end runtime of baselines and BLAZEIT on limit queries; BLAZEIT outperforms all baselines. The y-axis is log-scaled. All queries looked for 10 events.	84
4.1	TASTI system overview.	89
4.3	Number of target labeler invocations for baselines and TASTI for approximate aggregation queries (lower is better). TASTI outperforms baselines in all cases, including prior, per-query proxy state-of-the-art by up to $2\times$	104
4.4	False positive rate for recall-target SUPG queries (lower is better). We show the performance of baselines and TASTI. As shown, TASTI outperforms baselines in all cases.	105
4.5	Number of target labeler invocations for baselines and TASTI for limit queries (lower is better). TASTI outperforms baselines in all cases, including prior state-of-the-art by up to $34\times$	106
4.6	Breakdown of end-to-end inference of ResNet-50 and 18 for a batch size of 64 on the inference-optimized AWS <code>g4dn.xlarge</code> instance type. The execution of the DNN is $7.1\times$ and $22.9\times$ faster than preprocessing data for ResNet-50 and 18 respectively.	110
4.7	System diagram of SMOL. As input, SMOL takes a set of DNNs, visual input formats, and optional constraints. As output, SMOL returns an optimal set of plans or plan, depending on the constraints. SMOL will generate plans, estimate the resources for each plan, and select the Pareto optimal set of plans.	111
4.8	Examples of partial decoding for images. On the left, the ROI is the central crop of the image. For JPEG images, SMOL can decode only the macroblocks that intersect the ROI. For image formats that do not allow for independently decoding macroblocks, SMOL can partially decode based on raster order (right).	123

4.9	Throughput vs accuracy for the naive baseline, TAHOMA, and SMOL on the four image datasets (Pareto frontier only). SMOL can improve throughput by up to 5.9× with no loss in accuracy.	127
4.10	Query execution time vs requested error for BLAZEIT and SMOL on the four video datasets we evaluated. As shown, SMOL consistently outperforms BLAZEIT by using more accurate specialized NNs, which reduces sampling variance, and lower resolution data, which reduces preprocessing costs.	129
5.1	Top row: example of flickering in three consecutive frames of a video. The object detection method, SSD [120], 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.	132
5.2	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.	136
5.3	Percentile of confidence of the top-10 ranked errors by confidence found by OMG 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. Mdel assertions can find errors where the original model has high confidence (94th percentile).	145
5.4	Performance of random sampling, uncertainty sampling, uniform sampling from model assertions, and BAL for active learning. BAL improves accuracy on unseen data and can achieve an accuracy target (62% mAP) with 40% fewer labels compared to random and uncertainty sampling for night-street . BAL also outperforms both baselines for the NuScenes dataset.	146
5.5	Example of human labels (orange) and missing labels (red) in the Lyft Perception dataset. The black truck highlighted is within 25m of the AV. Such errors can cause downstream issues with perception and planning systems.	148
5.6	Example of the factor graph (top) and corresponding LIDAR point cloud data (bottom). The track is in black and other human-proposed labels are in orange for reference.	150

5.7	System diagram for FIXY. Users provide features over perception data (e.g., box volume) and associations between observations. Given these inputs, FIXY will learn feature distributions, generate graphical models, score new data, and output potential errors.	152
5.8	Example of a motorcycle (highlighted in red) missed by human proposals. We show both the LIDAR point cloud data (top) and the camera view (bottom).	157
5.9	Example of an unlikely track. Predictions are inconsistent within a track, suggesting that they are spurious.	160
5.10	Example of missing human label within a track that FIXY can find. The left panel only contains an ML model prediction while the right contains both a human label and an ML model prediction.	160
5.11	Example of a low probability bundle. The box of the person and truck highly overlap, but are strongly inconsistent in box volume.	160
5.12	Examples of labeling errors in the Lyft dataset. The missing objects in these examples can be within 20 meters the autonomous vehicle and several are in motion: vehicles in motion are the most important to detect.	163
5.13	Example of a model error (in black) in the Lyft dataset not found by ad-hoc model assertions. We show ground-truth boxes from human labels in orange for reference. The erroneous prediction overlaps across frames, but is not consistent. FIXY can find such errors as they produce unlikely values under learned feature distributions.	165

Chapter 1

Introduction

Unstructured (non-tabular) data in the form of videos, images, text, and audio, are now the largest forms of data produced and rapidly growing in size. As an example, a single fleet of vehicles (the Tesla fleet) can already produce *exabytes* of video data per day. Furthermore, video data already accounts for 88% of internet traffic [131]. Beyond data from vehicles, this data comes from street cameras, video conferencing meetings, personal cameras, forums, and social media.

As a parallel trend, programmatic capabilities of extracting *semantic* information from these unstructured data sources have dramatically improved. These programmatic capabilities have largely been in the form of powerful machine learning (ML) models. ML models can now extract object types and positions from images [150], actions from video [77], sentiment from text [186], and other information. In addition to ML models, there has been a proliferation of services providing high-quality human labels over unstructured data.

Given these two trends, analysts and scientists are increasingly interested in automatically answering queries over unstructured data to understand the real world. Urban planners can understand traffic patterns from camera feeds; social scientists can understand how trust in science changes after major world events from newspaper scans; journalists can understand dynamic situations from social media posts.

Unfortunately, answering unstructured queries is challenging for three reasons. First, extracting the semantic information necessary for queries is incredibly expensive. Extracting object types and positions from the video feeds of a small town using a state-of-the-art ML method may cost over \$1M in cloud compute credits. Second, ML methods can be unreliable, returning incorrect results, such as missing or hallucinated objects in a video.

Finally, deploying ML methods is challenging, requiring expertise in programming, statistics, and computer systems.

In this dissertation, I develop systems, algorithms, and techniques to address the issues in answering unstructured data queries. In order to allow non-experts to issue unstructured data queries, I have built systems that allow these application users to specify queries via standard SQL and extensions. By allowing application users to declaratively specify their queries in a familiar language, SQL, the systems I have built can then optimize these queries automatically.

The optimizations I have developed address the high cost of ML-based queries by using cheap approximations *while* providing guarantees on query accuracy. These optimizations can provide one to four orders of magnitude speedups compared to naive methods of answer queries. To address the reliability of ML methods, I have developed new programming abstractions to find errors in ML models and the data used to train them. These abstractions can help developers find errors in ML models with high precision (>80%) and can even be used to improve the quality of these ML models. My work has been deployed to aid scientists and autonomous vehicle companies, with real-world improvements of three orders of magnitude for analytics and $2\times$ cost improvements for finding errors.

In the remainder of this chapter, I discuss the challenges of unstructured analytics and provide an overview of my contributions towards answering unstructured analytics queries.

1.1 Challenges to Unstructured Data Analytics

1.1.1 The High Cost of Unstructured Data Analytics

The first challenge to answering unstructured queries is its high cost. To understand why the cost is prohibitive in answering unstructured queries, consider if it were free to execute ML models. In this scenario, we could simply fully materialize all the semantic information, e.g., by using Mask R-CNN to extract all object types and positions from every frame in a video, and use a standard structured database to answer queries.

Unfortunately, this is infeasible for many organizations. To understand the why, I analyzed the cost of answering queries in two scenarios: an urban planner analyzing a small town's worth of video (one month of video from 100 street corners) and a social scientist analyzing how language evolves through Wikipedia text (four billion tokens). For each scenario, we compute *only* the cost of materializing the semantic information using a self-hosted ML

	Urban planning	Wikipedia
Structured query	\$0.042	\$0.000026
Self-hosted ML (AWS)	\$380,000	\$59
ML service (GCP, OpenAI)	\$18,000,000	\$300,000
Human annotation (Scale AI)	\$630,000,000	\$320,000,000

Table 1.1: Costs of executing queries over unstructured data via self-hosted methods, an ML service, and using human annotators compared to the cost of executing a structured query over similar data.

model (AWS EC2), an ML model service (Google Cloud, Open AI), and human annotations (Scale AI). We also include the cost of a structured query over the same data for reference.

At the time of writing the costs were as follows. The AWS EC2 `g4dn.xlarge` costs \$0.526 per hour. The Google Cloud Vision API costs \$2.25 per 1,000 images. The OpenAI Davinci model costs \$0.08 per 1,000 tokens. Finally, Scale AI labeling costs \$0.08 per image and \$0.08 for 100 tokens.

As shown in Table 1.1, the cost of naively executing unstructured queries can be *orders of magnitude* more expensive than executing similar structured queries, *costing up to millions of dollars*. While the cost difference between the ML services may seem striking (up to $5,000\times$ more expensive), these ML services are widely used. One reason is that many organizations do not have teams of engineers that can train multi-billion parameter state-of-the-art models, such as GPT-3 [25]. As a result, they must rely on external services, which can cost millions of dollars to naively answer unstructured queries.

Thus, the cost of ML models is a major challenge in answering unstructured queries.

1.1.2 The Unreliability of Unstructured Data Analytics

The second challenge to answering unstructured queries is the unreliability of the ML models often used to answer these queries. An emerging body of work in the ML literature has shown that even state-of-the-art ML methods are brittle. These ML models can return highly miscalibrated answers [67], fail to generalize under distribution shift [19], and be fooled by imperceptible perturbations [63].

This unreliability in ML models can cause catastrophic errors in downstream applications, both for analytics applications and beyond. For example, errors in ML models have already caused fatal accidents in autonomous vehicles [171]. Furthermore, any errors in ML models will be reflected in unstructured data queries that depend on these models.



Figure 1.1: A confident error from a state-of-the-art model for detecting birds. The confidence is in the top 99.9th percentile in the dataset.

To understand how errors can affect unstructured data queries, consider the example of ecologists searching for hummingbirds in field videos. My collaborators in the Stanford biology department are interested in finding at least 50% of the hummingbirds in ~ 200 days of collected field videos [32, 126]. The 50% recall target is required for scientific validity when studying hummingbird feeding patterns. We deployed a state-of-the-art object detection model (Mask R-CNN [73]) to find birds in this video and labeled the top 1,000 clips of video (out of $\sim 8M$ total). Only 30 of these 1,000 clips contained hummingbirds, a precision of 3%. Furthermore, the minimum confidence of these 1,000 clips was 98.5%, showing that these models are highly miscalibrated on real-world data. Simply trusting the model’s confidence would result in inaccurate queries, as shown in Figure 1.1.

As these results show, ML can be unreliable. Directly deploying ML models would result in erroneous queries and other negative downstream consequences.

1.1.3 The Programming Difficulty of Unstructured Data Analytics

The third challenge in using ML to answer unstructured data queries is the difficulty in managing and deploying ML models. Training, managing, and deploying ML models in production requires knowledge of programming, deep learning, and data management. Most

domain experts in fields outside of computer science (e.g., ecologists, social scientists, business analysts) do not have this expertise. Furthermore, even organizations and experts that do have this expertise must use precious resources towards the management of ML models.

1.2 Efficient and Reliable Unstructured Data Queries

In this dissertation, I construct novel systems and algorithms for efficient and reliable unstructured data queries. To do so, I develop three key ideas. The first key idea is to allow application users to express unstructured data queries declaratively in standard SQL, similar to how structured databases allow users to declaratively express queries over structured data. Given a query specification, the second key idea is to combine cheap approximations to expensive ML methods with statistically principled algorithms, which can improve query costs by orders of magnitude. The third key idea is to design new programming abstractions to allow domain experts to specify when errors in ML models and the data used to train them might occur, and automatically mitigate them.

Thesis statement: High-level interfaces for querying unstructured data with ML, combined with automatic optimizations, can deliver high performance and statistical guarantees on result quality to make unstructured data analytics practical.

In the remainder of this section, I provide an overview of my contributions in the dissertation to support the thesis and a summary of the main results.

1.2.1 Efficient ML-based Queries

Unlike in standard queries over structured data, the primary cost in querying unstructured data is extracting the structured information via expensive ML methods or even human labelers. We show an example of costs in Table 1.1. Due to the costs of these methods, the extraction of the structured information must be done at query time. As a result, many standard query processing techniques cannot be applied and data management with ML must be rethought.

In the first line of work in this dissertation, I describe my work on generating and using cheap approximations, called *proxy models*, to accelerate ML-based queries. Proxy models are substantially cheaper than expensive ML models, but can be inaccurate, which is not acceptable in many applications. To rectify this, I have developed algorithms to accelerate

general classes of queries: selection, aggregation, and limit queries *with statistical guarantees on query results*. I further show how to efficiently generate these approximations.

Selection queries (classification). An important class of queries is selection queries, in which the user wishes to select records matching a predicate, e.g., frames of a video containing a hummingbird. I explored generating cheap approximations (which I refer to as *proxy scores*) in the NOSCOPE system [94]. NOSCOPE trains a proxy model to approximate whether or not a data record satisfies the target DNN-based predicate. The proxy model is used to generate a proxy score per data record, which is combined with the target DNN to answer queries. NOSCOPE can improve approximate selection by orders of magnitude compared to the solution of exhaustive labeling. NOSCOPE has inspired other recent techniques for ML-based data analytics [11, 30, 124]. Furthermore, I have shown that proxy models can accelerate general classes of traditional ML workloads [109], e.g., data-transformation bound workloads.

Selection queries with guarantees. While NOSCOPE and other systems [11, 30, 124] can accelerate approximate selection queries, they do not provide *statistical guarantees on the recall of the returned set*. These guarantees are critical for scientific rigor. For example, my collaborators in the Stanford biology department wish to find rare events of hummingbirds feeding in wildlife video. To ensure scientifically valid inferences, they require statistical guarantees on the recall of the discovered hummingbirds. I am actively working on deploying SUPG to this application.

To obtain statistical guarantees, I have developed SUPG, query semantics and sampling algorithms for approximate selection queries with guarantees [95]. My algorithms selectively sample the target DNN and optimize confidence intervals over the samples, which provides the statistical guarantees. Prior work uses uniform sampling, which I show results in poor quality results (i.e., returned sets with low precision). To improve the sampling efficiency, SUPG instead uses a novel set of weights for importance sampling. I show that my algorithms can improve query quality by up to $30\times$ for a fixed budget.

Aggregation and limit queries with guarantees. I have also developed algorithms to optimize aggregation (computing a statistic over the data records), aggregation with predicate (computing a statistic over a subset of the data records that satisfy a condition), and limit (finding a limited number of records that match a set of predicates) queries [93, 96]. Perhaps surprisingly, I show that these different queries require different algorithms.

BLAZEIT, which optimizes aggregation and limit queries, reduces variance in sampling for aggregation queries and ranks rare events for limit queries. In contrast, ABAE, which optimizes aggregation with predicate queries, uses stratified sampling based on proxy models to avoid sampling records not satisfying the predicates. We show that the convergence of ABAE requires novel analysis of stratified sampling with stochastic draws. Both systems can improve query execution times by orders of magnitude compared to baselines.

1.2.2 Efficient and High Quality Proxy Score Generation

While the algorithms I have described above can accelerate unstructured data queries, they require high-quality proxy scores. To generate proxy scores efficiently, I have developed methods for creating indexes for proxy-based algorithms and efficiently generating them.

Efficient indexes for proxy scores. While proxy scores can accelerate many query types, they can be inefficient to deploy. A common method of generating proxy scores is to train a new, cheap model *per query* to approximate the expensive target DNN [30, 124]. Unfortunately, this method does not share work across queries, requires ad-hoc training methods, and requires executing the target DNN many times.

To address these issues, I have developed TASTI, an indexing method for constructing proxy scores for unstructured data via an embedding index [97]. TASTI pre-computes embeddings that can be used to place records that are close under target DNN outputs together and annotates a small fraction of the records. To generate scores, TASTI assigns close records (by embedding distance) to the value of the nearest annotated record. Because these embeddings are pre-computed and are designed to work for any query over the target DNN output, they can be reused across queries and query types (including every query type I described above). I show that TASTI is simultaneously over 10× cheaper at index construction time and can return query results up to 24× better than ad-hoc proxy models.

Efficiently executing visual analytics [99]. Recent research, e.g., new accelerators, has greatly improved the throughput of DNNs by up to 150×. While this work has improved the throughput of *DNN execution*, it ignores other costs. I show that the *preprocessing* of visual data (e.g., image decoding) now bottlenecks end-to-end DNN inference for visual analytics systems by up to 23×, in the first measurement study of its kind [99]. I have built SMOL, a system that jointly optimizes preprocessing and DNN execution, to address this bottleneck. SMOL can improve throughput by up to 5.9× at a fixed accuracy.



Figure 1.2: An object detection model erroneously missing the prediction of the car in the second frame. Even widely used models (SSD) fail simple checks, such as temporal consistency.

1.2.3 Monitoring and Quality Assurance

Although my work and others have shown the promise to deploy ML in analytics and beyond, the widespread deployment of ML is hampered by the lack of reliability in ML models. For example, errors in ML models can cause fatal accidents in autonomous vehicles.

To begin to address the unreliability of ML methods, I have developed systems and abstractions to monitor ML methods, continuously improve ML models, and improve training data quality. My work has been deployed at an autonomous vehicle company.

As ML methods continue to improve on benchmark tasks, they are increasingly being deployed in mission-critical settings, such as autonomous vehicles. However, average-case measures of performance can hide potentially critical errors. Although software testing has developed tools for testing critical software, these tools are not directly applicable to ML.

My work has taken steps to bridge these two views. I’ve developed two abstractions, model assertions [101] and learned observation assertions [92]. Both abstractions are used to find potential errors in ML model predictions and human labels.

Model assertions allow users to specify specific forms of potential errors. For example, consider a state-of-the-art object detection DNN deployed over video to detect cars. Even state-of-the-art models can fail simple assertions, such as temporal consistency, e.g., that a car should not appear and disappear rapidly in a video (Figure 1.2). As an example of an assertion, I show the pseudocode for the flickering assertion, which can be written in a few lines of code:

```
# Boxes are indexed by time and id
def flicker(boxes: Box[][]):
    failures = 0
    for box1 in boxes[-1]:
```

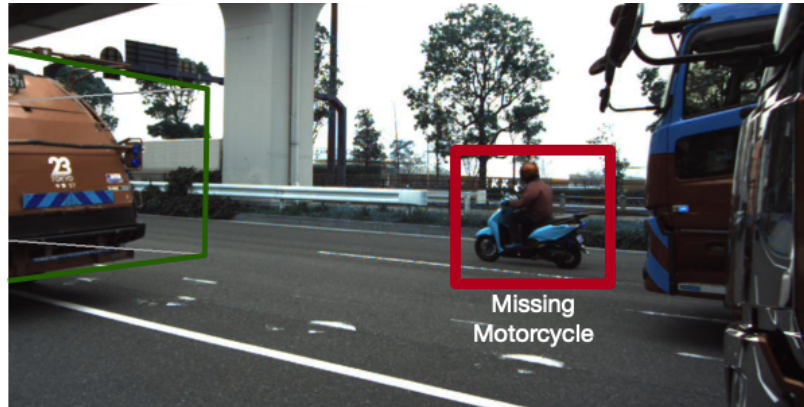


Figure 1.3: A motorcycle missed by a human annotator in a self-driving car dataset. The dataset was generated by a leading label vendor, showing that even best-in-class services can have errors in “gold” labels.

```

for box3 in boxes[-3]:
    if overlaps(box1, box3) and no_overlap(box1, boxes[-2]):
        failures += 1
return failures

```

Learned observation assertions (LOA) leverage existing human labels to learn when there may be discrepancies in ML model predictions or new, possibly erroneous, human labels (Figure 1.3). Model assertions and LOA can find errors with high true positive rate, at least 75% in all cases we studied.

Furthermore, I showed that assertions can be used in retraining ML models. Organizations continuously collect data to retrain ML models as they are deployed over new scenarios, e.g., autonomous vehicles seeing new streets. It is critical to select data that will improve the model as the majority of data is uninteresting. I showed that model assertions can be used to select “difficult” data (i.e., data that the model fails on), which improves ML model quality more than baselines. Assertions can reduce labeling costs by up to 40% at a fixed budget by finding such data.

My work allows application users to express unstructured data queries via methods they are already familiar with, SQL. Given queries specified declaratively with SQL, the systems and algorithms I have built automatically optimize unstructured data queries to be orders of magnitude more efficient. Furthermore, domain experts can improve the ML models used in these queries via abstractions until they are satisfied with model quality.

1.3 Organization

In this dissertation, I describe the systems, algorithms, and abstractions for efficient and reliable unstructured data queries. The remainder of this dissertation is organized as follows:

1. In Chapter 2, I present background on example applications, advances in machine learning, and related work.
2. In Chapter 3, I present NoSCOPE, SUPG, ABAE, and BLAZEIT, which are proxy-based algorithms and systems for accelerated ML-based queries. These systems collectively answer selection queries, aggregations, and aggregations with predicates with both best-effort and exact semantics. They can either improve query runtimes by orders of magnitude or improve query performance by up to $30\times$ depending on the setting.
3. In Chapter 4, I present TASTI and SMOL, which collectively execute ML-based queries efficiently. TASTI generates proxy scores efficiently by constructing an index that can be used across queries. SMOL executes proxy-based queries efficiently by generating query plans that consider preprocessing costs. These systems can deliver up to $24\times$ improved query runtimes.
4. In Chapter 5, I present model assertions and learned observation assertions, which are novel abstractions for finding errors in ML models and the data used to train them. Our prototype systems that implement these, OMG and FIXY, can find errors with high precision ($>80\%$) that prior work cannot find and improve model quality with up to $2\times$ fewer labels.
5. In Chapter 6, I conclude with a discussion of trends in ML-based applications and the results in this dissertation.

The work in this dissertation is adapted from a range of publications (VLDB '17 [94], VLDB '19 [93], VLDB '20 [95], MLSys '20 [101], VLDB '21 [96, 100], SIGMOD '22 [92, 97]). My research has inspired a number of extensions in the literature [15, 30, 38, 72, 124].

In addition to academic impact, my work has been deployed in a variety of real-world applications. My work on query processing has been deployed on ecological data to find hummingbirds in large-scale video in a cost-effective manner. Furthermore, my work on finding errors in ML deployments has been deployed at an autonomous vehicle company.

Chapter 2

Background

In this chapter, I provide background for unstructured data queries and discuss related work. I describe several examples of real-world unstructured data queries in Section 2.1, describe general trends for ML deployments in Section 2.2, and discuss related work in Section 2.3.

2.1 Example Applications

We begin by describing several example applications that require executing queries over unstructured data. We describe three examples: an urban planner conducting traffic analysis, ecologists finding hummingbirds in field videos for ecological analysis, and social scientists analyzing historical newspaper scans to understand how news affects public opinion.

2.1.1 Traffic Analysis

Suppose an urban planner is interested in conducting traffic analysis. To do so, the urban planner collects the videos from cameras placed on street corners around the city. After collecting this video, the urban planner may be interested in a range of queries.

To generally understand the data, the urban planner could issue an aggregation query that counts the average number of cars per frame of the video. Then, to further understand traffic patterns, the urban planner may execute a query that computes the average number of cars per frame of video when there is a red light.

After understanding high-level statistics about the video, the urban planner may be interested in seeing specific instances of events for further analysis. The urban planner may issue a query to find instances of five cars and a bus at a stop light to understand congestion.

Processing this video manually is infeasible even for small towns. Even analyzing a month of video over 100 intersections could cost up to millions of dollars, as shown in Table 1.1.

2.1.2 Ecological Analysis

Our collaborators in the Stanford biology department and Jasper Ridge nature preserve are studying bird-bacteria microcosms. To do so, they have collected microbial readings from flowers at a bush and field video of the flowers. The ecologists are interested in matching hummingbird feeding events (at the flowers) to the microbial readings. To find the feeding events, the ecologists could issue a query to find at least 90% of the hummingbird frames in the video. These frames can subsequently be used in their feeding analysis.

The ecologists have collected 200 camera-days of video, which is ~ 10 TB of data. Unfortunately, this data is too much for the team of three scientists to manually analyze. Furthermore, as they specialize in ecology, they are not well equipped to deploy complex ML methods over this large scale data.

2.1.3 News Analysis

Our social scientist collaborators at Harvard, Northwestern, and NYU are interested in understanding how news affects public opinion. To do so, they have collected newspaper scans from the 1900s to the present. After collecting these scans, the social scientists split the scans into semantically meaningful regions. For example, images on the page are separated from section headings and the main text. Finally, the regions are turned into text via optical character recognition.

After extracting the text, the social scientists are interested in a range of queries. To understand how major world events affect opinion on science, they may be interested in issuing a query to compute the average sentiment on science before and after the moon landing. They may also be interested in retrieving articles discussing the polio vaccine around its initial release.

These newspaper scans span decades. As a result, there are millions of scans and terabytes of data. Furthermore, the text generated by the scans is in the order of hundreds of gigabytes and growing as more text is added to the corpus.

2.2 Deploying ML for Analytics

As described in the examples above, the data volumes for these analyses are too large for manual analysis. As a result, analysts and scientists are increasingly turning to ML for automatic analyses.

The trend of using ML for analytics is driven in large part by the increasing capabilities of ML models. For example, ML models can match human-level performance on supervised object detection on the MS-COCO dataset [119]. Furthermore, on zero-shot learning tasks, large language models are now performing markedly better [25].

Unfortunately, as described in Section 1.1, there are three major problems in deploying ML: difficulty, cost, and reliability.

To understand these challenges, consider the example of finding hummingbirds in ecological field video. In order to find hummingbirds, the ecologists must:

1. Manage ~ 10 TB of video, which will not fit on a single laptop.
2. Decide which ML model to deploy among the many hundreds of options in the literature.
3. Write the code to deploy the ML model over large-scale video, including the serving, work distribution, and logging of results.
4. Collect the results, analyze the accuracy, and determine if the accuracy is high enough.

Doing these steps requires expertise in programming, computer vision, and large-scale data management. Furthermore, as described in Section 1.1, deploying the ML model can be costly and return inaccurate results.

Recent progress in ML is driven in large part by larger datasets and model sizes [25]. For example, language models display sharp improvements in downstream tasks when they are larger than a critical threshold [175]. Given these trends, we expect ML models to increase in cost as their capabilities improve.

Given these trends, many analysts and scientists are willing to tolerate *approximate answers* to queries. Answering approximate queries has a long history in the data community, known as approximate query processing (AQP). However, as we show, answering approximate queries over unstructured data requires new techniques, algorithms, and systems.

2.3 Related Work

We now discuss two areas of related work: approximate query processing for structured data and retrieval.

2.3.1 Approximate Query Processing

The data analytics community has developed a number of techniques for answering approximate queries over structured data. These approximate query processing (AQP) techniques broadly fall under two categories: online aggregation and offline synopsis generation [118]. Online aggregation “select[s] samples online and use these samples to answer OLAP queries” and offline synopsis generation “generate[s] synopses offline based on a-priori knowledge (e.g., data statistics or query workload) and use these synopses to answer OLAP queries” [118].

Offline synopsis generation uses pre-computed data structures, ranging from samples [5, 7], histograms [45, 142, 144], wavelets [66], to sketches [58, 59]. In all of these cases, with the exception of uniform samples, these methods require that the structured data is already available at ingest time to compute these synopses. Unfortunately, the structured data is not available ahead of time for unstructured data. As a result, these techniques cannot be directly applied to unstructured data queries.

Online aggregation methods aim to generally return progressively more accurate answers to aggregation queries. These methods typically uniformly sample from the records. Namely, online aggregation methods use random sampling as additional processing of structured records is similar to the cost of simply aggregating them, so uniform sampling is the most efficient. As we show in this dissertation, the cost of materializing structured data from unstructured data (i.e., running ML models) is so high, this necessitates new methods of answering approximate queries.

2.3.2 Retrieval

A closely related field is retrieval, in which the goal is to retrieve related records (typically text or image records) to a target record [65]. Retrieval is typically used in large applications, including for manual analysis [65], search [52], or question answering [105]. An emerging body of work in the retrieval community leverages modern ML methods [106]. These methods broadly use powerful deep neural networks to embed the data records and the target record. They then use similarity of embeddings to do retrieval.

In general, retrieval techniques do not directly apply to analytical queries. In particular retrieval techniques do not apply to aggregation queries, selection queries with guarantees on recall, and other analytical queries. However, several of the emerging techniques in retrieval are similar in spirit to the techniques described in this dissertation.

Chapter 3

Proxy-based Algorithms and Systems

In this chapter, I describe algorithms and systems that accelerate approximate unstructured data queries by leveraging *proxies*. Proxies are cheap approximations to expensive deep learning models.

As described in Section 2.3, answering approximate queries via AQP techniques has a long history in the structured data literature. However, unstructured data does not have the structured records materialized ahead of time. As a result, precomputation is not feasible for unstructured data. On the other hand, naive random sampling can answer queries but is not efficient.

To address these issues, I have developed methods of using proxies to “guide” sampling to answer unstructured data queries more efficiently. In this chapter, we describe a simple method of generating proxies, but defer a full discussion of efficient generation of proxies to Chapter 4. In the remainder of this chapter, I describe my algorithms and systems to leverage these proxies for efficient selection, aggregation, aggregation with predicate, and limit queries. The algorithms and systems I have developed can provide up to 40× improved query execution times or query quality (depending on the query type) compared to baselines.

In this chapter, we describe how to leverage these proxies to accelerate a range of queries including best-effort selection queries (NOSCOPE, Section 3.1), selection queries with guarantees on accuracy (SUPG, Section 3.2), aggregation queries with predicates (ABAE, Section 3.3), aggregation queries, and limit queries (BLAZEIT, Section 3.4).

3.1 NoSCOPE

The first system I describe is NoSCOPE, a system for querying videos that can reduce the cost of selection queries in video by orders of magnitude via *inference-optimized model search*. In particular, NoSCOPE supports queries in the form of the presence or absence of a particular object class. Given a query consisting of a target video, object to detect, and reference *pre-trained* neural network (e.g., webcam in Taipei, buses, YOLOv2 [149]), NoSCOPE automatically searches for and trains a sequence, or *cascade* [169], of models that preserves the accuracy of the reference network but are specialized to the target query and are therefore far less computationally expensive. That is, instead of simply running the reference NN over the target video, NoSCOPE searches for, learns, and executes a query-specific pipeline of cheaper models that approximates the reference model to a specified target accuracy. NoSCOPE’s query-specific pipelines forego the generality of the reference NN—that is, NoSCOPE’s cascades are *only* accurate in detecting the target object in the target video—but in turn execute up to three orders of magnitude faster (i.e., $265\text{--}15,500\times$ real-time) with 1-5% loss in accuracy for binary detection tasks over real-world fixed-angle webcam and surveillance video. To do so, NoSCOPE leverages both new types of models and a new optimizer for model search:

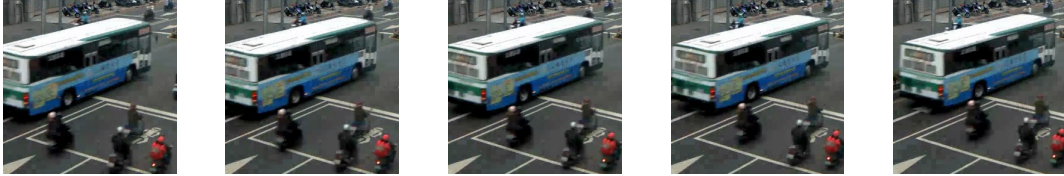
First, NoSCOPE uses proxy models that forego the full generality of the reference NN but faithfully mimic its behavior for the target query. In the context of our example query of detecting buses, consider the following buses that appeared in a public webcam in Taipei:



To generate these proxy models, NoSCOPE performs *model specialization*, using the full NN to generate labeled training data (i.e., examples) and subsequently training smaller NNs that are tailored to a given video stream and to a smaller class of objects. NoSCOPE then executes these proxy models, which are up to $340\times$ faster than the full NN, and consults the full NN only when the proxy models are uncertain (i.e., produce results with confidence below an automatically learned threshold).

Second, NoSCOPE’s *difference detectors* highlight temporal differences across frames.

Consider the following frames, which appeared sequentially in our Taipei webcam:



These frames are nearly identical, and all contain the same bus. Therefore, instead of running the full NN (or a proxy NN) on each frame, NOSCOPE learns a low-cost difference detector (based on differences of frame content) that determines whether the contents have changed across frames. NOSCOPE’s difference detectors are fast and accurate—up to 100k frames per second on the CPU.

A key challenge in combining the above insights and models is that the optimal choice of cascade is data-dependent. Individual model performance varies across videos, with distinct trade-offs between speed, selectivity, and accuracy. For example, a difference detector based on subtraction from the previous frame might work well on mostly static scenes but may add overhead in a video overseeing a busy highway. Likewise, the complexity (e.g., number of layers) of proxy NNs required to recognize different object classes varies widely based on both the target object and video. Even setting the thresholds in the cascade represents trade-off: should we make a difference detector’s threshold less aggressive to reduce its false negative rate, or should we make it more aggressive to eliminate more frames early in the pipeline and avoid calling a more expensive model?

To solve this problem, NOSCOPE performs inference-optimized model search using a cost-based optimizer that automatically finds a fast model cascade for a given query and accuracy target. The optimizer applies candidate models to training data, then computes the optimal thresholds for each combination of models using an efficient linear parameter sweep through the space of feasible thresholds. The entire search requires time comparable to labeling the sample data using the reference NN (an unavoidable step in obtaining such data).

We evaluate a NOSCOPE prototype on binary classification tasks on cameras that are in a fixed location and at a fixed angle; this includes pedestrian and automotive detection as found in monitoring and surveillance applications. NOSCOPE demonstrates up to three order of magnitude speedups over general-purpose state-of-the-art NNs while retaining high—and configurable—accuracy (within 1-5%) across a range of videos, indicating a promising new strategy for efficient inference and analysis of video data.

In the remainder of this section, we describe NoSCOPE’s architecture, describe its evaluation, and conclude with a discussion on NoSCOPE’s impacts.

3.1.1 NoSCOPE Architecture and Techniques

NoSCOPE Queries and Goal. NoSCOPE targets binary classification queries—i.e., presence or absence of a given class of object in a video over time. In NoSCOPE, users input *queries* by selecting a target object class (e.g., one of the 9000 classes recognized by YOLO9000, such as humans, cars, and buses [149]) as well as a target video. Subsequently, NoSCOPE outputs the time intervals in the video when an object of the specified class was visible according to a given *reference model*, or full-scale NN trained to recognize objects in images. NoSCOPE allows users to specify a target accuracy in the form of false positive and false negative rates and aims to maximize throughput subject to staying within these rates.¹ In summary, given these inputs, NoSCOPE’s goal is to *produce the same classification output as applying the target model on all frames of the video, at a substantially lower computational cost and while staying within the specified accuracy target.*

System Components. NoSCOPE is comprised of three components, as shown in Figure 3.1: *a)* proxy models, *b)* difference detectors, and *c)* an inference-optimized cost-based optimizer. When first provided a new video, NoSCOPE applies the reference model to a subset of the video, generating labeled examples. Using these examples, NoSCOPE searches for and learns a cascade of cheaper models to accelerate the query on the specific video. NoSCOPE subsequently runs the cascade over the remainder of the video, stopping computation at the cheapest layer of the cascade as soon as it is confident.

NoSCOPE uses two types of models. First, NoSCOPE trains *proxy models* (Section 3.1.2) that perform classification tasks. For example, while detecting humans with perfect accuracy in all frames may require running the full reference model, we show that a much smaller NN can output a confidence value c that lets us safely label a frame as “no human” if it is below some threshold c_{low} , label it as “human” if $c > c_{\text{high}}$, and pass the frame to the full NN if it is unsure (i.e., $c_{\text{low}} < c < c_{\text{high}}$). Second, NoSCOPE uses *difference detectors* to check whether the current frame is similar to a recent frame whose label is known (e.g., for a camera looking at a hallway, this could be an image where the hallway is empty).

¹A false positive is a case where NoSCOPE reports an object but running the reference model would have reported no object. A false negative is a case where NoSCOPE reports no object but the reference model would have reported one.

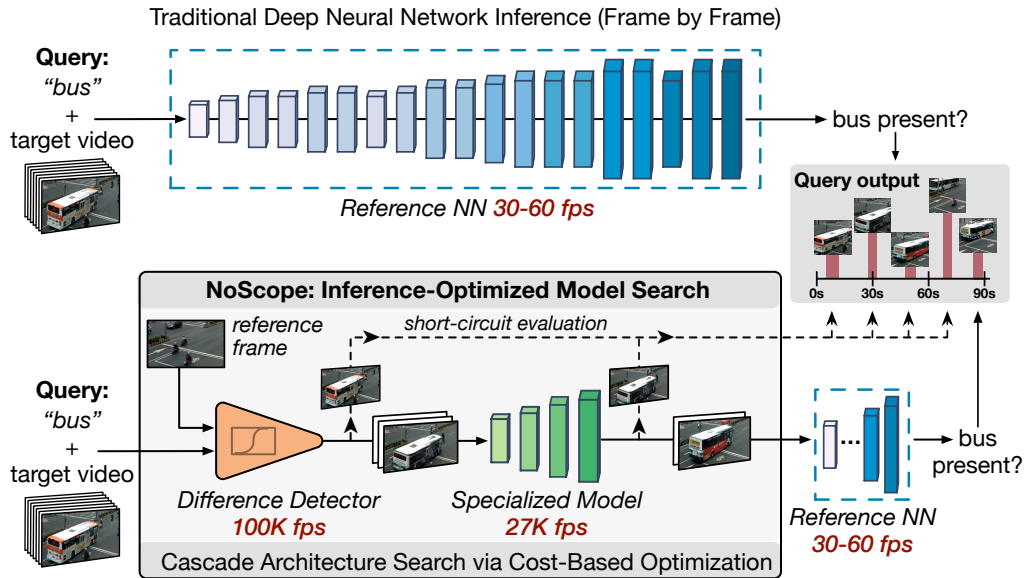


Figure 3.1: NoSCOPE is a system for accelerating neural network analysis over videos via inference-optimized model search. Given an input video, target object, and reference neural network, NoSCOPE automatically searches for and trains a cascade of models that can reproduce the binarized outputs of the reference network with high accuracy—but up to three orders of magnitude faster.

Finally, to automatically search for and configure these models NoSCOPE includes a *cost-based optimizer* that learns an efficient configuration of filters for each query to achieve the target accuracy level (i.e., false positive and false negative rates). We have found (and empirically demonstrate) that customizing cascades for each video is critical for performance.

We provide an overview of NoSCOPE’s proxy generation technique, but defer the description of the difference detectors, cost-based optimization, and model search to Kang et al. [94].

3.1.2 Model Specialization

NoSCOPE uses proxy models to accelerate queries. These are smaller models that faithfully mimic the behavior of a reference model on a *particular* task. Generic NNs can classify or detect thousands of classes, and the generality of these methods naturally leads to costly inference. Proxy models forego the full generality of a generic, reference model but mimic its behavior on a subset of tasks the generic model can perform. In query systems such

as NOSCOPE, we are generally only interested in identifying a small number of objects—as opposed to the thousands of classes a generic NN can classify—and, in video inference, such objects may only appear from a small number of angles or configurations.

NOSCOPE performs model specialization by applying a larger, reference model to a target video and using the output of the larger model to train a smaller, proxy model. Given sufficient training data from the reference model for a specific video, the proxy model can be trained to mimic the reference model on the video while requiring fewer computational resources (e.g., NN layers) compared to the reference model. However, unlike the reference model, the proxy model learns from examples from the target video and is unlikely to *generalize* to other videos or queries. Thus, by sacrificing generalization and performing both training and inference on a restricted task and input data distribution, we can substantially reduce inference cost.

Critically, in contrast with related approaches to model compression [70, 79], the goal of model specialization is *not* to provide a model that is indistinguishable from the reference model on *all tasks*; rather, the goal of model specialization is to provide a model that is indistinguishable (to a given accuracy target) for a *restricted* set of tasks. This strategy allows efficient inference at the expense of generality.

NOSCOPE uses shallow NNs as its specialized models. Shallow NNs are efficient at inference time and naturally output a confidence in their classification. NOSCOPE uses this confidence to defer to the reference model when the specialized model is not confident (e.g., when no loss in accuracy can be tolerated). NOSCOPE implements proxy models based on the AlexNet architecture [110] (filter doubling, dropout), using ReLU units for all the hidden layers and a softmax unit at the end to return a confidence for the class we are querying. However, to reduce inference time, NOSCOPE’s networks are significantly shallower than AlexNet.

To train proxy models, NOSCOPE uses standard NN training practices. NOSCOPE uses a continuous section of video for training and cross-validation and learns NNs using RM-Sprop [78] for 1-5 epochs, with early stopping if the training loss increases. In addition, during model search, NOSCOPE uses a separate evaluation set that is not part of the training and cross-validation sets for each model.

Table 3.1: Video streams and object labels queried in our evaluation.

Video Name	Object	Resolution	FPS	# Eval frames	Length (hrs)
taipei	bus	1000x570	30	1296k	12.0
coral	person	1280x720	30	1188k	11.0
amsterdam	car	680x420	30	1296k	12.0
night-street	car	1000x530	30	918k	8.5
elevator	person	640x480	30	592k	5.5
roundabout	car	1280x720	25	731k	8.1

3.1.3 Evaluation

We evaluate NOSCOPE on binary classification for real-world webcam and surveillance videos. In this dissertation, we present end-to-end results and defer a detailed evaluation to Kang et al. [94]. We show that NOSCOPE can achieve $40\times$ improved throughput compared to exhaustive execution.

Evaluation Queries and Metrics. We evaluated NOSCOPE on a fixed set of queries shown in Table 3.1. We use YOLOv2 [149], a state-of-the-art multi-scale NN, as our reference model. YOLOv2 operates on 416x416 pixel images (resizing larger or smaller images). YOLOv2 achieves 80 fps on the Tesla P100 GPU installed on our measurement machine. We obtain videos from seven webcams—five from YouTube Live, and one that we manually obtained. We split each video into two parts: training and evaluation. Five videos have two days worth of video, with 8-12 hours of footage per day due to lighting conditions; for these videos, we use the first day of video for training and the second day for evaluation. For two videos, we use the first 2.3 hours for training and separate an evaluation set (5-8 hours) by a minimum of 30 minutes.

We measure throughput by timing the complete end-to-end system excluding the time taken to decode video frames.

Hardware Environment. We perform our experiments on an NVIDIA DGX-1 server, using at most one Tesla P100 GPU and 32 Intel Xeon E5-2698 v4 cores during each experiment. The complete system had 80 cores and multiple GPUs, but we limited our testing to a subset of these so our results would be representative of a less costly server. The system also had a total of 528 GB of RAM.

End-to-end evaluation. Figure 3.2 illustrates the overall range of performance that NOSCOPE achieves on our target queries. For each dataset, we obtained the points in the plot

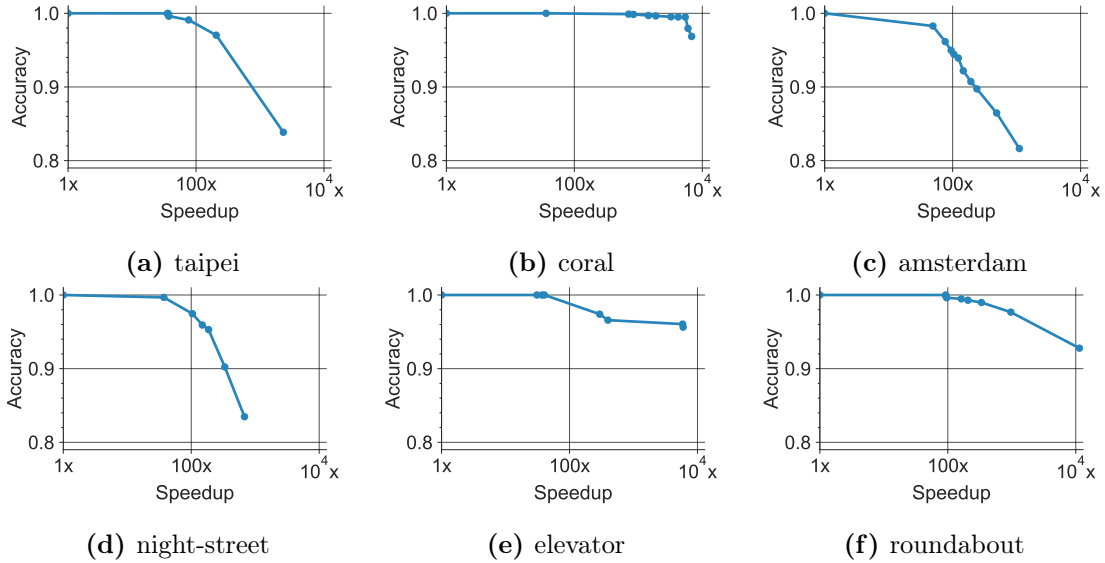


Figure 3.2: Accuracy vs. speedup achieved by NoSCOPE on each dataset. Accuracy is the percent of correctly labeled time windows, and speedup is over YOLOv2.

by running NoSCOPE’s CBO with increasing false positive and false negative thresholds (FP^* and FN^* , with $FP^* = FN^*$) and measuring the resulting speedup. NoSCOPE demonstrates several orders of magnitude speedup across all the datasets, with magnitude depending on the desired accuracy levels. In all cases, NoSCOPE achieves a $30\times$ speedup with at least 98% accuracy. In many cases, this level of accuracy is retained even at a $100\times$ speedup, and NoSCOPE can obtain $1000\times$ to $10,000\times$ speedups at 90+% accuracy. The video with the lowest peak speedup at the 90% accuracy mark is `taipei`, which shows a busy intersection—thus, the difference detectors cannot eliminate many frames. However, even in this video, NoSCOPE can offer an $30\times$ speedup over YOLOv2 with no loss in accuracy.

3.1.4 Discussion

As we have shown, NoSCOPE can accelerate inference over video at scale via inference-optimized model search. To do so, NoSCOPE uses proxy models trained via model specialization and difference detection. By combining these with cost-based optimization and model search, NoSCOPE can achieve up to orders of magnitude improved throughput compared to exhaustive evaluation.

As early work published in 2017, NoSCOPE has inspired further work on and using proxy models, including from groups at MIT, CMU, University of Washington, and MSR

[16, 30, 72, 124]. Proxy models are now a standard tool of ML-based analytics.

While NOSCOPE accelerates selection queries, it unfortunately does not provide guarantees on query results. Furthermore, it does not accelerate a range of other common queries, such as aggregation queries. In the following sections, we describe how to leverage proxies to answer a range of query types with guarantees on accuracy, including selection, aggregation, and limit queries.

3.2 Approximate Selection with Guarantees

Given the rise of the ability to collect large datasets, practitioners regularly aim to find all instances of rare events in these datasets. For example, biologists in a lab at Stanford have collected months of video of a flower field and wish to identify timestamps when hummingbirds are feeding so they can match hummingbird feeding patterns with microbial readings from the flowers. Furthermore, our contacts at an autonomous vehicle company are interested in auditing when their labeled data may be wrong, e.g., missing pedestrians [50], so they can correct them. Importantly, these events are *rare* (e.g., at most 0.1-1% of frames contain hummingbirds) and users are interested in the *set of matching records* as opposed to aggregate measures (e.g., counts).

Unfortunately, executing *oracle predicates* (e.g., human labelers or deep neural networks) to find such events can be prohibitively expensive, so many applications have a *budget* on executing oracle predicates. For example, biologists can watch only so many hours of video and companies have fixed labeling budgets.

NOSCOPE can accelerate selection queries by leveraging proxy models. These proxy models are typically small ML models that provide a confidence score for the label and selection predicate. If the proxy model’s confidence scores are reliable and consistent with the oracle, they can be used to filter out the vast majority of data unlikely to match.

There are two major challenges in using these proxy models to reduce the labeling cost subject to a budget: reliability of proxy models and oracle labeling efficiency.

First, given the budget, using an unreliable proxy model can result in false negatives or positives, making it difficult to guarantee the accuracy of query results. Existing systems do not provide guarantees on the accuracy. In fact they can fail unpredictably and catastrophically, providing results with low accuracy a significant fraction of the time [11, 30, 83, 93, 94, 124]. For example, when users request a precision of at least

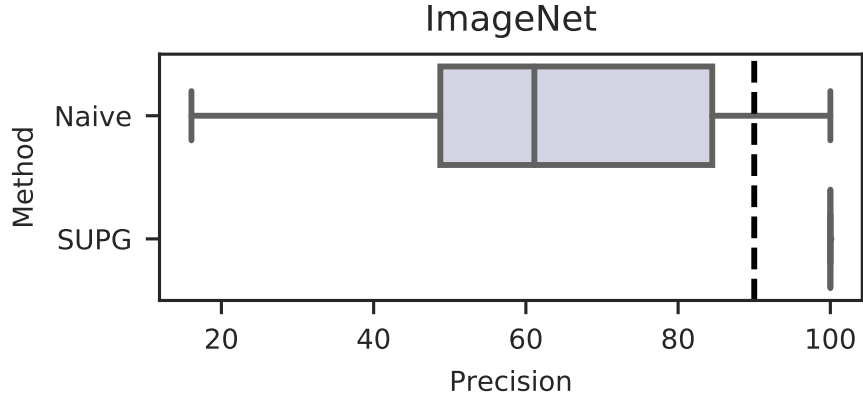


Figure 3.3: Box plot of achieved precisions of naive sampling from recent work [94, 124] and our improved algorithm. The naive algorithm returns precisions as low as 65% for over half the runs. In contrast, our algorithms (SUPG) achieve the precision target w.h.p.

90%, over repeated runs, existing systems return results with less than 65% precision *over half the time*, with some runs low as 20% (Figure 3.3). These failures can be even worse in the face of shifting data distributions, i.e., model drift (Section 3.2.5). Such failures are unacceptable in production deployment and for scientific inference.

Second, existing systems do not make efficient use of limited oracle labels to maximize the quality of query results. To avoid vacuous results (e.g., achieving a perfect recall by returning the whole dataset will have poor precision), NOSCOPE, probabilistic predicates, and other work uniformly sample records to label with the oracle in order to decide on the final set of records to return. We show that this is wasteful. In the common case where records matching the predicate are rare, the vast majority of uniformly sampled records will be negatives. Thus, naively extending existing techniques yields results with accuracy guarantees can fail to maintain high result quality given these uninformative labels.

In response we develop novel algorithms that provide *both* statistical guarantees and efficient use of oracle labels for approximate selection. We further develop query semantics for the two settings we consider: the recall target and precision target settings.

Accuracy guarantees. To address the challenge of guarantees on failure probability, we first define probabilistic guarantees for two classes of approximate selection queries. We have found that users are interested in queries targeting a minimum recall (RT queries) or targeting a minimum precision (PT queries), subject to an oracle label budget and a failure probability on the query returning an invalid result (Section 3.2.2). For instances,

the biologist are interested in 90% recall and a failure probability of at most 5%.

We develop novel algorithms (SUPG algorithms) that provide these guarantees by using the oracle budget to sample records to label, and estimating a proxy confidence threshold τ for which it is safe to return all records with proxy score above τ . Naive use of uniform sampling will not account for the probability that there is a deviation between observed labels and proxy scores, and will further introduce multiple hypothesis testing issues. This will result in a high probability of failure. In response, we make careful use of confidence intervals and multiple hypotheses corrections to ensure that the failure probability is controlled.

Oracle sample efficiency. A key challenge is deciding which data points to label with the oracle given the limited budget: as we show, uniform sampling is inefficient. Instead, we develop novel, optimal importance sampling estimators that use of the correlation between the proxy and the oracle, while taking into account mismatches between the binary oracle and continuous proxy. Intuitively, importance sampling upweights the probability of sampling data points with high proxy scores, which are more likely to contain events of interest.

However, naive use of importance sampling results in poor performance when sampling according to proxy scores. Using a variance decomposition, we find that a standard approach for obtaining importance weights (i.e., using weights proportional to the proxy) is suboptimal and, excluding edge cases, performs no better than uniform random sampling.

Instead, we show that sampling proportional to the *square root* of the proxy scores allows for more efficient estimates of the proxy threshold when the proxy scores are confident and reliable (Section 3.2.4). For precision target queries, we additionally extend importance sampling to use a two-stage sampling procedure. In the first stage, our algorithm estimates a safe interval to further sample. In the second stage, our algorithm samples directly from this range, which we show greatly improves sample efficiency.

Careless use of importance sampling can hurt result quality when used with poor proxy models. If proxy scores are uncorrelated with the true labels, importance sampling will in fact increase the variance of sampling. To address these issues, we defensively incorporate uniform samples to guard against situations where the proxy may be adversarial [137]. This procedure still maintains the probabilistic accuracy guarantees.

We implement and evaluate these algorithms on real and synthetic datasets. Our algorithms achieve desired accuracy guarantees, even in the presence of model drift. We further show that our algorithms outperform alternative methods in providing higher result quality,



Figure 3.4: Sample matching (a) and non-matching (b) frames for a selection query over a video stream used by our biologist collaborators. DNNs can serve as proxies to identify hummingbirds as shown in (a), but the confidence scores can be unreliable.

by as much as $30\times$ higher recall/precision under precision/recall constraints respectively.

In the remainder of this section, we describe use cases for SUPG, its algorithmic procedure in detail, and our evaluation of SUPG.

3.2.1 Use Cases

To provide additional context and motivation for approximate selection queries, we describe scenarios where statistically efficient queries with guarantees are essential. Each scenario is informed by discussions with academic and industry collaborations.

Biological Discovery

Scenario. We are actively collaborating with the Fukami lab at Stanford University, who study bacterial colonies in flowers [154]. The Fukami lab is interested in hummingbirds that move bacteria between flowers as they feed, as this bacterial movement can affect both the micro-ecology of the flowers and later hummingbird feeding patterns. To study such phenomena, they have collected videos of bushes with tagged flowers at the Jasper Ridge biological preserve. They have recorded six views of the scene with a total of approximately 9 months of video. At 60 fps, this is approximately 1.4B frames of video. To perform downstream analyses, our collaborators want to select all frames in the video that contain hummingbirds. Due to the rarity of hummingbird appearances ($< 0.1\%$) and the length of the video, they have been unable to manually review the video in its entirety. We illustrate the challenge in Figure 3.4.

Proxy model. Prior to our collaboration, the Fukami lab used motion detectors as a proxy for identifying frames with birds. However, the motion detectors have severe limitations: their precision is extremely low (approximately 2%) and they do not cover the full field of view of the bush. As an alternative, we are actively using DNN object detector models to identify hummingbirds directly from frames of the video [18, 73]. These DNN models are more precise than motion detectors, and can provide a confidence score in addition to a Boolean predicate result.

During discussion with the Fukami lab, we have found that the scientists require high probability guarantees on recall, as finding the majority of hummingbirds is critical for downstream analysis. Furthermore, they are interested in improving precision relative to the motion detectors. The scientists have specified that they need a recall of at least 90% and a precision that is as high as possible, ideally above 20%.

Autonomous Vehicle Training

Scenario. An autonomous vehicle company may collect data in a new area. To train the DNNs used in the vehicle, the company may extract point cloud or visual data and use a labeling service to label pedestrians. Unfortunately, labeling services are known to be noisy and may not label pedestrians even when they are visible [50].

To ensure that all pedestrians are labeled, an analyst may wish to select all frames where pedestrians are present but are not annotated in the labeled data. However, as autonomous vehicle fleets collect enormous amounts of data (petabytes per day), the analyst is not able to manually inspect all the data.

Proxy model. As the proxy model, the analyst can use an object detection method and remove boxes that are in the labeled dataset. The analyst can then use the confidences from the remaining boxes from the object detection as the proxy scores.

As this is a mission-critical setting, the analyst is interested in guarantees on recall. Missing pedestrians in the labeled dataset can transfer to missing pedestrians at deployment time, which can cause fatal accidents. The analyst may also be interested in using other proxies, such as 3D detections from LIDAR data. In this work, we only study the use of a single proxy model, but we see extending our algorithms to multiple proxy models as an exciting area of future work.

This scenario is not limited to autonomous vehicles but can apply to other scenarios

```

SELECT * FROM table_name
WHERE filter_predicate
ORACLE LIMIT o
USING proxy_estimates
[RECALL | PRECISION] TARGET t
WITH PROBABILITY p

```

Figure 3.5: Syntax for specifying approximate selection queries.

where curating high quality machine learning datasets is of paramount concern.

3.2.2 Approximate Selection Queries

We introduce our definitions for our approximate selection queries (SUPG queries), describe the probabilistic guarantees they respect, and define metrics for comparing result quality.

Query Semantics

A SUPG query is a selection query for set of records matching a predicate, with syntax given in Figure 3.5. Unlike much of the existing work in approximate query processing, SUPG queries return a set of matching records rather than a scalar or vector aggregate [7, 75]. We defined these semantics to formalize a common class of queries our collaborators and industrial contacts are interested in executing.

The query specifies a filter predicate given by a “ground truth” oracle, as well as a limited budget of total calls to the oracle over the course of query execution. We use the term oracle to refer to any expensive predicate the user wishes to approximate. In some cases, the oracle may be an expensive DNN (e.g., the highly accurate Mask R-CNN [73]) that may not exactly match the ground truth labels that a human labeler would provide. However, the use of proxies to approximate powerful deep learning models is common in the literature [11, 83, 93, 94, 124], so we study how to provide guarantees in applications that use a larger DNN as an oracle.

Since oracle usage is limited, queries also specify proxy confidence scores for whether a record matches the predicate. The proxy scores must be correlated with the probability that a record matches the filter predicate to be useful. Nonetheless, our novel algorithms will return valid results even if proxy scores are not correlated.

The accuracy of the set of results can be measured using either *recall* (the fraction of true matches returned) or *precision* (the fraction of returned results that are true matches).

Based on the application, a user can specify either a minimum recall or precision target as well as a desired probability of achieving this target. We refer to these two options as *precision target* (PT) and *recall target* (RT) queries. As an example, consider the following RT query:

```
SELECT * FROM hummingbird_video
WHERE HUMMINGBIRD_PRESENT(frame) = True
ORACLE LIMIT 10,000
USING DNN_CLASSIFIER(frame) = "hummingbird"
RECALL TARGET 95%
WITH PROBABILITY 95%
```

where both `HUMMINGBIRD_PRESENT` and `DNN_CLASSIFIER` are user-defined functions (UDFs). This query selects the frames of the video that contains a hummingbird with recall at least 95%, using at most 10,000 oracle evaluations, and a failure probability of at most 5%, using confidence probabilities from a DNN classifier as a proxy. The oracle could be a human labeler or expensive DNN.

Finally, we note that some queries may require both a recall and precision target. Unfortunately, jointly achieving both targets may require an unbounded number of oracle queries. Since all use cases we consider have limited budgets, we defer our discussion of these queries to an extended version of this chapter [95].

Probabilistic Guarantees

More formally, a SUPG query \mathcal{Q} is defined by an oracle predicate $O(x) \in \{0, 1\}$ over a set of records x from a dataset \mathcal{D} . The ideal result for the query would be the matching records $O^+ := \{x \in \mathcal{D} : O(x) = 1\}$. However, since the oracle is assumed to be expensive, the query specifies a budget of s calls to the oracle $O(x)$, as well as a proxy model $A(x) \in [0, 1]$ whose use is unrestricted. The query specifies a minimum recall or precision target γ . Then, a *valid* query result would be a set of records \mathcal{R} such that $\text{Precision}(\mathcal{R}) > \gamma_p$ or $\text{Recall}(\mathcal{R}) > \gamma_r$ depending on the query type. Recall that

$$\text{Precision}(\mathcal{R}) := \frac{|\mathcal{R} \cap O^+|}{|\mathcal{R}|} \quad \text{Recall}(\mathcal{R}) := \frac{|\mathcal{R} \cap O^+|}{|O^+|}.$$

A SUPG query further specifies a failure probability δ . Many precision or recall targets γ may be impossible to achieve deterministically given a limited budget of s calls to the oracle, as they require exhaustive search. Thus, it is common in approximate query processing and

statistical inference to use randomized procedures with a bounded failure probability [44]. A randomized algorithm satisfies the guarantees in \mathcal{Q} if it produces valid results \mathcal{R} with high probability. That is, for PT queries:

$$\Pr[\text{Precision}(\mathcal{R}) \geq \gamma_p] \geq 1 - \delta \quad (3.1)$$

and for RT queries:

$$\Pr[\text{Recall}(\mathcal{R}) \geq \gamma_r] \geq 1 - \delta. \quad (3.2)$$







These high probability guarantees are much stronger than merely achieving an average recall or precision as many existing systems do [11, 83, 94, 124]. For example, in Figure 3.8 we illustrate the true recall provided for queries targeting 90% recall to NOSCOPE system, and compare them with the recall provided by SUPG which satisfies the stronger guarantee in Equation 3.2. NOSCOPE only achieves the target recall approximately half of the time, with many runs failing to achieve the recall target by a significant margin. Such results that fail to achieve the recall target would have a significant negative impact on downstream statistical analyses.

Result Quality

Since SUPG queries only specify a target for either precision or recall (the *target metric*), there are many valid results for a given query which may be more or less useful. For instance, if a user targets 99% recall the entire dataset is always a valid result, even though this may not be useful to the user. In this case, it would be more useful to return a smaller set of records to minimize false positives. Similarly, if a user targets high precision the empty set is always a valid result, and is equally useless. Thus, we define selection query quality in this chapter as follows:

Definition 1. For RT/PT queries, a higher quality result is one with higher precision/recall, respectively.

There is an inherent trade-off between returning valid results and maximizing result quality, analogous to the trade-off between maximizing precision and maximizing recall in binary classification [28, 64], but efficient use of oracle labels will allow us to develop more efficient importance sampling based query techniques.

Data Records	Proxy Scores	Oracle (naive)	Oracle (SUPG)
	0.94	?	✓
	0.92	✓	✓
...			
	0.75	x	?
	0.71	?	✓
...			
	0.30	x	x
	0.12	x	?



 Selected by Naive, 66% recall
 Selected by SUPG, >95% recall

Figure 3.6: SUPG algorithms use sampled oracle labels and proxy scores to identify a subset of records that satisfy a recall or precision target with high probability. Naive methods use limited oracle samples less efficiently and can fail to achieve the target recall or precision.

3.2.3 Algorithm Overview

In this section, we describe the system setting that our SUPG algorithms operate in, and outline the major stages in the algorithm: sampling oracle labels, choosing a proxy threshold, and returning a set of data record results.

Operational Architecture

Our algorithms are designed for batch query systems that perform selection on datasets of existing records. Users can issue queries over the data with specified predicates and parameters as described earlier. Note that the oracle and proxy models used to evaluate the filter predicate are provided by the user as UDFs (callback functions) and are not inferred by the system. Thus, a user must provide either a ground truth DNN or interface to obtain human input as an oracle, as well as pre-trained inexpensive proxy models. In practice, one can provide user interfaces for interactively requesting human labels [4] as well as scripts for automatically constructing smaller proxy models from an existing oracle [94, 124], though those are outside the scope of this chapter.

We illustrate how SUPG uses oracle and proxy models in Figure 3.6. For all query types, SUPG first executes the proxy model over the complete set of records \mathcal{D} as we assume the proxy model is cheap relative to the oracle model. Then, SUPG samples a set \mathcal{S} of s records

to label using the oracle model. The choice of which records to label using the oracle is done adaptively, that is, the choice of samples may depend on the results of previous oracle calls for a given query.

Algorithm 1 SUPG query processing

```

function SUPGQUERY( $\mathcal{D}$ ,  $A$ ,  $O$ )
   $\mathcal{S} \leftarrow \text{SampleOracle}(\mathcal{D})$ 
   $\tau \leftarrow \text{EstimateTau}(\mathcal{S})$ 
   $\mathcal{R}_1 \leftarrow \{x : x \in \mathcal{S} \wedge O(x) = 1\}$ 
   $\mathcal{R}_2 \leftarrow \{x : x \in \mathcal{D} \wedge A(x) \geq \tau\}$ 
  return  $\mathcal{R}_1 \cup \mathcal{R}_2$ 

```

We summarize this sequence of operations SUPG uses to return query results in Algorithm 1. After calling the oracle to obtain predicate labels over a sample \mathcal{S} , SUPG sets a proxy score threshold τ and then returns results \mathcal{R} that consist of both labeled records in \mathcal{S} matching the oracle predicate as well as records with proxy scores above the threshold τ . τ is tuned so that the final results \mathcal{R} satisfy minimum recall or precision targets with high probability, and we describe the process for setting τ below.

Choosing a Proxy Threshold

Since the proxy scores are the only source of information on the query predicate besides the oracle model, SUPG naturally returns records corresponding to all records with scores above a threshold τ . This strategy is known to be optimal in the context of retrieval and ranking as long as proxy scores grow monotonically with an underlying probability that the record matches a predicate [125]. We have observed in practice that this is approximately true for proxy models by computing empirical match rates for bucketed ranges of the proxy scores, so we use this as the default strategy in SUPG. For proxy models that are completely uncorrelated or have non-monotonic relationships with the oracle, all algorithms using proxies will have increasingly poor quality, but SUPG will still provide accuracy guarantees.

Thus, the key task is selecting the threshold τ to maintain result validity while maximizing result quality. This threshold must be set at query time since the relation between proxy scores and the predicate is unknown, especially when production model drift is an issue. Existing systems have often relied on pre-set thresholds determined ahead of time, which we show in Section 3.2.5 can lead to severe violations of result validity.

One naive strategy for selecting τ at query time is to uniform randomly sample records

Table 3.2: Notation Summary

Symbol	Description
$O(x)$	Oracle predicate value
$A(x)$	Proxy confidence score
δ	Failure probability
γ	Target Recall / Precision
τ	Proxy score threshold for selection
\mathcal{S}	Records sampled for oracle evaluation

to label with the oracle until the budget is exhausted, and then select τ that achieves a target accuracy over the sample. However, this strategy on its own does not provide strong accuracy guarantees or make efficient use of the sample budget. Thus, in Section 3.2.4 we introduce more sophisticated methods for sampling records and estimating the threshold: that is, implementations of `SampleOracle` and `EstimateTau`.

3.2.4 Estimating proxy thresholds

Recall that SUPG selects all records with proxy scores above a threshold τ . Denote this set of records

$$\mathcal{D}(\tau) := \{x : A(x) \geq \tau\}.$$

SUPG query accuracy thus critically depends on the choice of τ . In this section we describe our algorithms for estimating a threshold that can guarantee valid results with high probability, while maximizing result quality. While precision target (PT) and recall target (RT) queries require slightly different threshold estimation routines, in both cases SUPG samples records to label with the oracle. Using this sample, SUPG will select a threshold τ that achieves the target metric on the dataset \mathcal{D} with high probability.

In order to explain our algorithms and compare them with existing work, we will also describe a number of baseline techniques which do not provide statistical guarantees, and do not make efficient use of oracle labels to improve result quality.

We now describe baselines without guarantees, how to correct these baselines for statistical guarantees on failure probability, and finally our novel importance sampling algorithms.

Baselines Without Guarantees

The simplest strategy for estimating a valid threshold would be to take a uniform i.i.d. random sample of records \mathcal{S} , label the records with the oracle, and then use \mathcal{S} as an exact

representative of the dataset \mathcal{D} when choosing a threshold. This is the approach used by probabilistic predicates and NoSCOPE [94, 124], and we call this approach U-NoCI because it uses a uniform sample and does not account for failure probabilities using confidence intervals (CI). Let $\text{Recall}_{\mathcal{S}}(\tau)$ and $\text{Precision}_{\mathcal{S}}(\tau)$ denote the empirical recall and precision for the sampled data \mathcal{S} , and 1_c denote the indicator function that condition c holds:

$$\text{Recall}_{\mathcal{S}}(\tau) := \frac{\sum_{x \in \mathcal{S}} 1_{A(x) \geq \tau} O(x)}{\sum_{x \in \mathcal{S}} O(x)} \quad (3.3)$$

$$\text{Precision}_{\mathcal{S}}(\tau) := \frac{\sum_{x \in \mathcal{S}} 1_{A(x) \geq \tau} O(x)}{|\mathcal{S}|}. \quad (3.4)$$

The U-NoCI-P approach maximizes result quality subject to constraints on these empirical recall and precision estimates. For PT queries this results in finding the minimal τ (minimizing false negatives) that achieves the target metric on \mathcal{S} , and for RT queries this results in finding the maximum τ (minimizing false positives). Formally this is defined as,

$$\tau_{\text{U-NoCI-P}}(\mathcal{S}) = \min\{\tau : \text{Precision}_{\mathcal{S}}(\tau) \geq \gamma\} \quad (3.5)$$

$$\tau_{\text{U-NoCI-R}}(\mathcal{S}) = \max\{\tau : \text{Recall}_{\mathcal{S}}(\tau) \geq \gamma\}. \quad (3.6)$$

However, we have no guarantee that the thresholds selected in this way will provide valid results on the complete dataset, due to the random variance in choosing a threshold based on the limited sample. We empirically show that such algorithms fail to achieve targets up to 80% of the time in Section 3.2.5.

Guarantees through Confidence Intervals

In order to provide probabilistic guarantees, we form confidence intervals over τ and take the appropriate upper or lower bound.

Normal approximation. In Lemma 1 we describe an asymptotic bound relating sample averages to population averages, allowing us to bound the discrepancy between recall and precision achieved on \mathcal{S} vs \mathcal{D} . This approximation is commonly used in the approximate query processing literature [6, 68, 75].

For ease of notation we will refer to the upper and lower bounds provided by Lemma 1

using helper functions

$$\text{UB}(\mu, \sigma, s, \delta) := \mu + \frac{\sigma}{\sqrt{s}} \sqrt{2 \log \frac{1}{\delta}} \quad (3.7)$$

$$\text{LB}(\mu, \sigma, s, \delta) := \mu - \frac{\sigma}{\sqrt{s}} \sqrt{2 \log \frac{1}{\delta}}. \quad (3.8)$$

Lemma 1. Let \mathcal{S} be a set of s i.i.d. random variables $x \sim \mathcal{X}$ with mean μ and finite variance σ^2 and sample mean $\hat{\mu}$. Then,

$$\lim_{s \rightarrow \infty} \Pr [\hat{\mu} \geq \text{UB}(\mu, \sigma, s, \delta)] \leq \delta$$

and

$$\lim_{s \rightarrow \infty} \Pr [\hat{\mu} \leq \text{LB}(\mu, \sigma, s, \delta)] \leq \delta.$$

Lemma 1 defines the expected variation in recall and precision estimates as s grows large, and follows from the Central Limit Theorem [174]. Using this, we can select conservative thresholds that with high probability still provide valid results on the underlying dataset \mathcal{D} . Though this bound is an asymptotic result for large s , quantitative convergence rates for such statistics are known to be fast [20] and we found that this approach provides the appropriate probabilistic guarantees at sample sizes $s > 100$.

We will now describe baseline uniform sampling based methods for estimating τ in both RT and PT queries.

Recall Target. For recall target queries, we want to estimate a threshold τ such that $\text{Recall}_{\mathcal{D}}(\tau) \geq \gamma$ with probability at least $1 - \delta$. To maximize result quality we would further like to make τ as large as possible. We present the pseudocode for a threshold selection routine U-CI-R that provides guarantees on recall in Algorithm 2.

Note that Algorithm 2 finds a cutoff τ that achieves a conservative recall of γ' on \mathcal{S} instead of the target recall γ . This inflated recall target accounts for the potential random variation from forming the threshold on \mathcal{S} rather than \mathcal{D} .

Validity justification. Let τ_o be the largest threshold providing valid recall on \mathcal{D} :

$$\tau_o := \max\{\tau : \text{Recall}_{\mathcal{D}}(\tau) \geq \gamma\}$$

If $\text{Recall}_{\mathcal{S}}(\tau_o) \leq \gamma'$ then Algorithm 1 will select a threshold τ' where $\tau' \leq \tau_o$ since recall varies inversely with the threshold. Then, $\text{Recall}_{\mathcal{D}}(\tau') \geq \text{Recall}_{\mathcal{D}}(\tau) \geq \gamma$ and the results derived from τ' would be valid.

It remains to show that with probability $1 - \delta$, γ' satisfies:

$$\text{Recall}_{\mathcal{S}}(\tau_o) \leq \gamma'. \quad (3.9)$$

Let $Z_1(\tau), Z_2(\tau)$ be sample indicator random variables for matching records above and below τ_o , corresponding to the samples in \mathcal{S} .

$$\begin{aligned} Z_1(\tau) &:= \{1_{A(x) \geq \tau} O(x) : x \in \mathcal{S}\} \\ Z_2(\tau) &:= \{1_{A(x) < \tau} O(x) : x \in \mathcal{S}\}. \end{aligned}$$

Note that $\frac{\hat{\mu}_{Z_1(\tau)}}{\hat{\mu}_{Z_1(\tau)} + \hat{\mu}_{Z_2(\tau)}} = \text{Recall}_{\mathcal{S}}(\tau)$, which increases with $\hat{\mu}_{Z_1(\tau)}$ and decreases with $\hat{\mu}_{Z_2(\tau)}$. Thus, if we let

$$\gamma^* = \frac{\text{UB}(\mu_{Z_1(\tau_o)}, \sigma_{Z_1(\tau_o)}, s, \frac{\delta}{2})}{\text{UB}(\mu_{Z_1(\tau_o)}, \sigma_{Z_1(\tau_o)}, s, \frac{\delta}{2}) + \text{LB}(\mu_{Z_2(\tau_o)}, \sigma_{Z_2(\tau_o)}, s, \frac{\delta}{2})}$$

then asymptotically as $s \rightarrow \infty$ Lemma 1 ensures $\text{Recall}_{\mathcal{S}}(\tau_o) = \frac{\hat{\mu}_{Z_1(\tau_o)}}{\hat{\mu}_{Z_1(\tau_o)} + \hat{\mu}_{Z_2(\tau_o)}} \leq \gamma^*$ with probability $1 - \delta$. γ^* is not computable from our sample so we use plug-in estimates for τ_o , μ , and σ to estimate a $\gamma' \rightarrow \gamma^*$ as $s \rightarrow \infty$.

Precision Target. For precision target queries, we want to estimate a threshold τ such that $\text{Precision}_{\mathcal{D}}(\tau) \geq \gamma$ with high probability. To maximize result quality (i.e., maximize recall), we would further like to make τ as small as possible.

Unlike for recall target queries, there is no monotonic relationship between $\text{Precision}_{\mathcal{D}}(\tau)$ and τ : $\text{Precision}_{\mathcal{D}}(\tau_1)$ may be greater than $\text{Precision}_{\mathcal{D}}(\tau_2)$ even if $\tau_1 < \tau_2$. Thus, for PT queries we calculate lower bounds on the precision provided by a large set of candidate thresholds τ , and return the smallest candidate threshold that provides results with precision above the target.

Algorithm 2 Uniform threshold estimation (RT)

```

function  $\tau_{\text{U-CI-R}}(\mathcal{D})$ 
   $\mathcal{S} \leftarrow \text{UniformSample}(\mathcal{D}, s)$ 
   $\hat{\tau}_o \leftarrow \max\{\tau : \text{Recall}_{\mathcal{S}}(\tau) \geq \gamma\}$ 
   $Z_1 \leftarrow \{1_{A(x) \geq \hat{\tau}_o} O(x) : x \in \mathcal{S}\}$ 
   $Z_2 \leftarrow \{1_{A(x) < \hat{\tau}_o} O(x) : x \in \mathcal{S}\}$ 
   $\gamma' \leftarrow \frac{\text{UB}(\hat{\mu}_{z_1}, \hat{\sigma}_{z_1}, s, \delta/2)}{\text{UB}(\hat{\mu}_{z_1}, \hat{\sigma}_{z_1}, s, \delta/2) + \text{LB}(\hat{\mu}_{z_2}, \hat{\sigma}_{z_2}, s, \delta/2)}$ 
   $\tau' \leftarrow \max\{\tau : \text{Recall}_{\mathcal{S}}(\tau) \geq \gamma'\}$ 
return  $\tau'$ 

```

Algorithm 3 Uniform threshold estimation (PT)

```

 $m \leftarrow 100$  ▷ Minimum step size
function  $\tau_{\text{U-CI-P}}(\mathcal{D})$ 
   $\mathcal{S} \leftarrow \text{UniformSample}(\mathcal{D}, s)$ 
   $A_{\mathcal{S}} \leftarrow \text{Sort}(\{A(x) : x \in \mathcal{S}\})$ 
   $M \leftarrow \lceil s/m \rceil$ 
  Candidates  $\leftarrow \{\}$ 
  for  $i \leftarrow m, 2m, \dots, s$  do
     $\tau \leftarrow A_{\mathcal{S}}[i]$ 
     $Z \leftarrow \{O(x) : x \in \mathcal{S} \wedge A(x) \geq \tau\}$ 
     $p_l \leftarrow \text{LB}(\hat{\mu}_Z, \hat{\sigma}_Z, |Z|, \delta/M)$  ▷ Precision Bound
    if  $p_l > \gamma$  then
      Candidates  $\leftarrow \text{Candidates} \cup \{\tau\}$ 
return  $\min_{\tau} \text{Candidates}$ 

```

We provide pseudocode for U-CI-R which uses confidence intervals over a uniform sample (Algorithm 3). Since the procedure uses Lemma 1 M times by union bound we need each usage to hold with probability $1 - \delta/M$ for the final returned threshold to be valid with probability $1 - \delta$.

Validity justification. Let

$$Z(\tau) = \{O(x) : x \in \mathcal{S} \wedge A(x) \geq \tau\},$$

then $\hat{\mu}_{Z(\tau)} = \text{Precision}_{\mathcal{S}}(\tau)$ and $\mu_{Z(\tau)} = \text{Precision}_{\mathcal{D}}(\tau)$. Asymptotically by Lemma 1, with probability $1 - \delta/M$

$$\text{LB}(\hat{\mu}_{Z(\tau)}, \sigma_{Z(\tau)}, |Z(\tau)|, \delta/M) \leq \mu_{Z(\tau)}.$$

By the union bound, as long as each τ in the Candidate set has $\text{LB}(\hat{\mu}_{Z(\tau)}, \sigma_{Z(\tau)}, |Z(\tau)|, \delta/M) >$

γ , the precision for each of the candidates over the dataset also exceeds γ . Since we do not know σ , in Algorithm 3 we use sample plug-in estimates for $\sigma_{Z(\tau)}$. Alternatively one could use a t-test (both are asymptotically valid).

Importance Sampling

The U-CI routines for estimating τ in Algorithms 2 and 3 provide valid results with probability $1 - \delta$. However if the random sample chosen for oracle labeling \mathcal{S} is uninformative, the confidence bounds we use will be wide and the threshold estimation routines will return results that have lower quality in order to provide valid results. Thus, we explain how SUPG uses importance sampling to select a set of points that improve upon uniform sampling. We refer to these more efficient routines as IS-CI estimators.

Importance sampling chooses records x with replacement from the dataset \mathcal{D} with weighted probabilities $w(x)$ as opposed to uniformly with base probability $u(x)$. One can compute the expected value of a quantity $f(x)$ with reduced variance by then sampling according to w rather than u and using the reweighting identity:

$$\mathbb{E}_{x \sim u} [f(x)] = \mathbb{E}_{x \sim w} \left[f(x) \frac{u(x)}{w(x)} \right]. \quad (3.10)$$

Abbreviating the reweighting factor as $m(x) := u(x)/w(x)$, we can then define reweighted estimates for recall and precision on a weighted sample \mathcal{S}_w :

$$\text{Recall}_{\mathcal{S}_w}(\tau) := \frac{\sum_{x \in \mathcal{S}} 1_{A(x) \geq \tau} O(x) m(x)}{\sum_{x \in \mathcal{S}_w} O(x) m(x)} \quad (3.11)$$

$$\text{Precision}_{\mathcal{S}_w}(\tau) := \frac{\sum_{x \in \mathcal{S}} 1_{A(x) \geq \tau} O(x) m(x)}{\sum_{x \in \mathcal{S}_w} m(x)} \quad (3.12)$$

If we can reduce the variance of these estimates, we can use the tighter bounds to improve the quality of the results at a given recall or precision target.

The optimal choice of $w(x)$ for the standard importance sampling setting is $w(x) \propto f(x)u(x)$ [174]. However, this assumes $f(x)$ is a known function. In our setting, we want $f(x) = 1_{A(x) \geq \tau} O(x)$ which is both stochastic and a priori unknown. This prevents us from directly applying traditional importance sampling weights based on $f(x)$. Instead, we can use the proxy $A(x)$ to define sampling weights.

Our approach solves for the optimal sample weights for proxies that are highly correlated

Algorithm 4 Importance threshold estimation (RT)

```

function  $\tau_{\text{IS-CI-R}}(\mathcal{D})$ 
   $\vec{w} \leftarrow \{\sqrt{A(x)} : x \in \mathcal{D}\}$ 
   $\vec{w} \leftarrow .9 \cdot \vec{w} / \|\vec{w}\|_1 + .1 \cdot \vec{1} / |\mathcal{D}|$  ▷ Defensive Mixing
   $\mathcal{S} \leftarrow \text{WeightedSample}(\mathcal{D}, \vec{w}, s)$ 
   $m(x) \leftarrow \frac{1/|\mathcal{D}|}{w(x)}$ 
   $\tau_o \leftarrow \max\{\tau : \text{Recall}_{\mathcal{S}_w}(\tau) \geq \gamma\}$ 
   $\hat{z}_1 \leftarrow \{1_{A(x) \geq \tau_o} O(x)m(x) : x \in \mathcal{S}\}$ 
   $\hat{z}_2 \leftarrow \{1_{A(x) < \tau_o} O(x)m(x) : x \in \mathcal{S}\}$ 
   $\gamma' \leftarrow \frac{\text{UB}(\hat{\mu}_{z_1}, \hat{\sigma}_{z_1}, s, \delta/2)}{\text{UB}(\hat{\mu}_{z_1}, \hat{\sigma}_{z_1}, s, \delta/2) + \text{LB}(\hat{\mu}_{z_2}, \hat{\sigma}_{z_2}, s, \delta/2)}$ 
   $\tau' \leftarrow \max\{\tau : \text{Recall}_{\mathcal{S}_w}(\tau) \geq \gamma'\}$ 
return  $\tau'$ 

```

with the oracle, i.e. *calibrated* with $A(x) = \Pr_{x \sim u}[O(x) = 1 | A(x)]$. In practice this will not hold exactly, but as long as the proxy scores are approximately proportional to the probability we can use them to derive useful sample weights. We show in Section 3.2.4 that the optimal weights which minimize the variance are proportional to $\sqrt{A(x)1_{A(x) \geq \tau} u(x)}$. To guard against situations where the proxy could be inaccurate, we defensively mix a uniform distribution with these optimal weights in our algorithms [137].

Note that the validity of our results does not depend on the proxy being calibrated, but this importance sampling scheme allows us to obtain lower variance threshold estimates and thus more efficient query results when the proxy is close to calibrated.

Recall target. For recall target queries, we extend Algorithm 2 to use weighted samples according to Theorem 1. We use the weights to optimize the variance of $E[O(x)]$ as a proxy for reducing the variance of $E[1_{A(x) \geq \tau_o} O(x)]$ and $E[1_{A(x) < \tau_o} O(x)]$. We present this weighted method, IS-CI-R, in Algorithm 4. The justification for high probability validity is the same as before.

Precision target. For PT queries we can combine Theorem 1 with an additional observation: if we know there are at most n_{match} positive matching records in \mathcal{D} , then there is no need to consider thresholds lower than the n_{match}/γ -th highest proxy score in \mathcal{D} , since any lower thresholds cannot achieve a precision of γ . SUPG thus devotes half of the oracle sample budget to estimating the upper bound n_{match} and the remaining half for running a weighted version of Algorithm 3 on candidate thresholds. We present this two-stage weighted sampling algorithm, IS-CI-P, in Algorithm 5.

Algorithm 5 Importance threshold estimation (PT)

```

 $m \leftarrow 100$  ▷ Minimum step size
function  $\tau_{\text{IS-CI-P}}(\mathcal{D})$ 
   $\vec{w} \leftarrow \{\sqrt{A(x)} : x \in \mathcal{D}\}$ 
   $\vec{w} \leftarrow .9 \cdot \vec{w} / \|\vec{w}\|_1 + .1 \cdot \vec{1} / |\mathcal{D}|$  ▷ Defensive Mixing
   $\mathcal{S}_0 \leftarrow \text{WeightedSample}(\mathcal{D}, w, s/2)$  ▷ Stage 1
   $m(x) \leftarrow \frac{1/|\mathcal{D}|}{w(x)}$ 
   $Z \leftarrow \{O(x)m(x) : x \in \mathcal{S}_0\}$ 
   $n_{\text{match}} \leftarrow |\mathcal{D}| \cdot \text{UB}(\hat{\mu}_Z, \hat{\sigma}_Z, s/2, \delta/2)$ 
   $A \leftarrow \text{SortDescending}(\{A(x) : x \in \mathcal{D}\})$ 
   $\mathcal{D}' \leftarrow \{x : A(x) \geq A[n_{\text{match}}/\gamma]\}$ 
   $\mathcal{S}_1 \leftarrow \text{WeightedSample}(\mathcal{D}', w, s/2)$  ▷ Stage 2
   $A_{\mathcal{S}_1} = A \cap \mathcal{S}_1$ 
   $M \leftarrow \lceil s/m \rceil$ 
  Candidates  $\leftarrow \{\}$ 
  for  $i \leftarrow m, 2m, \dots, s$  do
     $\tau \leftarrow A_{\mathcal{S}_1}[i]$ 
     $Z \leftarrow \{O(x) : x \in \mathcal{S}_1 \wedge A(x) \geq \tau\}$ 
     $p_l \leftarrow \text{LB}(\hat{\mu}_Z, \hat{\sigma}_Z, |Z|, \delta/(2M))$  ▷ Precision Bound
    if  $p_l > \gamma$  then
      Candidates  $\leftarrow$  Candidates  $\cup \{\tau\}$ 
  return  $\min_{\tau}$  Candidates

```

We set the failure probability of each stage to $\delta/2$ which guarantees the overall failure probability of the algorithm via the union bound. The remaining arguments for high probability validity follows the argument for the unweighted algorithm.

Statistical Efficiency.

Algorithm. Theorem 1 formally states the optimal sampling weights used by our importance sampling τ estimation routines. We defer the proof to Kang et al. [95].

Theorem 1. For an importance sampling routine estimating $\mathbb{E}_{x \sim u}[f(x)]$, when $f(x) = \{0, 1\}$, $a(x)$ is a calibrated proxy $\Pr_{x \sim u}[f(x) = 1 | a(x)] = a(x)$, and we sample knowing $a(x), u(x)$, but not $f(x)$, then importance sampling with $w(x) \propto \sqrt{a(x)}u(x)$ minimizes the variance of the reweighted estimator.

We apply can this to our algorithms using $f(x) = O(x) \cdot 1_{A(x) \geq \tau}$ and $a(x) = A(x) \cdot 1_{A(x) \geq \tau}$. To illustrate the impact of these weights, we can quantify the maximum improvement in variance they provide. Compared with uniform sampling or sampling proportional to $a(x)$,

these weights provide a variance reduction of at least $\Delta_v = \text{Var}_{x \sim u}[\sqrt{a(x)}]$, which is significant when the proxy confidences are concentrated near 0 and 1, while the differences vanish when there is little variation in the proxy scores.

Intuition. In standard importance sampling, the variance minimizing weights are proportional to the function values. However, in our setting, we only have access to probabilities (i.e., $A(x)$) for the function we wish to compute expectations over (i.e., $O(x)$). Since $O(x)$ is a randomized realization of $A(x)$, up-weighting x proportionally to $A(x)$ results in “overconfident” sampling. Thus, the square root weights effectively down-weights the confidence that $A(x)$ accurately reflects $O(x)$.

Table 3.3: Summary of datasets, oracle models, proxy models, and true positive rates.

Dataset	Oracle	Proxy	TPR	Task description
ImageNet	Human labels	ResNet-50	0.1%	Finding hummingbirds in the ImageNet validation set
night-street	Mask R-CNN	ResNet-50	4%	Finding cars in the night-street video
OntoNotes	Human labels	LSTM	2.5%	Finding city relationships
TACRED	Human labels	SpanBERT	2.4%	Finding employees relationships
Beta(0.01, 1)	True values	Probabilities	0.5%	$A(x) = \text{Beta}(0.01, 1)$ and $O(x) = \text{Bernoulli}(A(x))$
Beta(0.01, 2)	True values	Probabilities	1%	We use the same procedure as directly above but with Beta(0.01, 2)

3.2.5 Evaluation

We evaluate our algorithms on six real-world and synthetic datasets. We describe the experimental setup, demonstrate that naive algorithms fail to respect failure probabilities, demonstrate that our algorithms outperform uniform sampling (as used by prior work), and that our algorithms are robust to proxy choices.

Experimental Setup

Metrics. Following the query definitions in Section 3.2.2, we are interested in two primary evaluation metrics. First, we measure the empirical failure rate of the different algorithms: the rate at which they do not achieve a target recall or precision. Second, we measure the quality of query results using achieved precision when there is a minimum target recall, and achieved recall when there is a minimum target precision.

Methods Evaluated. In our evaluation we compare methods that all select records based on a proxy threshold as in Algorithm 1. The methods differ in their sampling routine and routine for estimating the proxy threshold τ as described in Section 3.2.4. Note that NOSCOPe and probabilistic predicates correspond to the baseline algorithms U-NOCI-R and U-NOCI-P with no guarantees. We can extend these algorithms to provide probabilistic guarantees in the U-CI-R and U-CI-P algorithms. Finally, our system SUPG uses the IS-CI-R and IS-CI-P algorithms which introduce importance sampling.²

Many systems additionally compare against full scans. However, this baseline always requires executing the oracle model on the entire dataset \mathcal{D} , requiring $|\mathcal{D}|$ oracle model invocations. On large datasets, this approach was infeasible for our collaborators and industry contacts, so we exclude this baseline from comparison.

Datasets and Proxy Models. We show a summary of datasets used in Table 3.3.

Beta (synthetic). We construct synthetic datasets using proxy scores $A(x)$ drawn from a $\text{Beta}(\alpha, \beta)$ distribution, allowing us to vary the relationship between the proxy model and oracle labels. We assign ground truth oracle labels as independent Bernoulli trials based on the proxy score probability. These synthetic datasets have 10^6 records and we use two pairs of (α, β) : (0.01, 1) and (0.01, 2).

ImageNet and night-street (image). We use two real-world image datasets to evaluate SUPG. First, we use the ImageNet validation dataset [153] and select instances of hummingbirds. There are 50 instances of hummingbirds out of 50,000 images or an occurrence rate of 0.1%. The oracle model is human labeling. Second, we use the commonly used *night-street* video [30, 93, 94, 181] and select cars. The oracle model is an expensive, state-of-the-art object detection method [73]. We resample the positive instances of cars to set the true positive rate to 4% to better model real-world scenarios where matches are rare.

The proxy model for both datasets is a ResNet-50 [74], which is significantly cheaper than the oracle model.

OntoNotes and TACRED (text). We use two real-world text datasets (OntoNotes [81] with fine-grained entities [36] and TACRED [187]). The task for both datasets is relation extraction, in which the goal is to extract semantic relationships from text, e.g., “organization” and “founded by.” We searched for city and employees relationships for OntoNotes and TACRED respectively. The oracle model is human labeling for both datasets.

²Code for our algorithms is available at <https://github.com/stanford-futuredata/supg>.

Table 3.4: Summary of distributionally shifted datasets. These shifts are natural (weather related, different day of video) and synthetic.

Dataset	Shifted dataset	Description
ImageNet	ImageNet-C, Fog	ImageNet with fog
night-street	Day 2	Different days
Beta(0.01, 1)	Beta(0.01, 2)	Shifted β parameter

The proxy model for OntoNotes is a baseline provided by with the fine-grained entities [36]. The proxy model for TACRED is the state-of-the-art SpanBERT model [88]. We choose different models to demonstrate that SUPG is agnostic to proxy model choice.

Baseline Methods Fail to Achieve Guarantees

We demonstrate that baseline methods fail to achieve guarantees on failure probability. First, we show that U-NOCI (i.e., uniform sampling from the universe and choosing the empirical cutoff, Section 3.2.4) fails. Note that U-NOCI is used by prior work. Second, we show that using U-NOCI on other data, as other systems do, also fails to achieve the failure probabilities.

U-NOCI fails. To demonstrate that U-NOCI fails to achieve the failure probability, we show the distribution of precisions and recalls under 100 trials of this algorithm and SUPG’s optimized importance sampling algorithm. For SUPG, we set $\delta = 0.05$. We targeted a precision and recall of 90% for both methods.

As shown in Figures 3.7 and 3.8, U-NOCI can fail as much as 75% of the time. Furthermore, U-NOCI can catastrophically fail, returning recalls of under 20% when 90% was requested. In contrast, SUPG’s algorithms respect the recall targets within the given δ .

U-NOCI fails under model drift. We further show that U-NOCI on different data distributions also fails to achieve the failure probability. This procedure is used by existing systems such as NOSCOPE and probabilistic predicates on a given set of data; the cutoff is then used on other data. These systems assume the data distribution is fixed, a known limitation.

To evaluate the effect of model drift, we allow the U-NOCI to choose a proxy threshold using oracle labels on *the entire training dataset* and then perform selection on test datasets with distributional shift. We compare this with applying the SUPG algorithms using a limited number of oracle labels from the shifted test set as usual. We summarize the shifted datasets in Table 3.4. We use naturally occurring instances of drift (obscuration by fog [76],

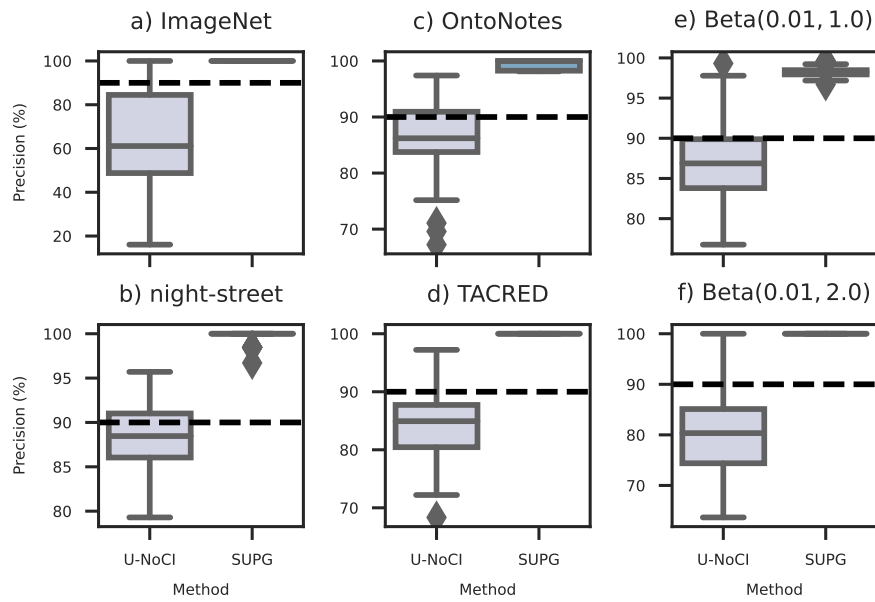


Figure 3.7: Precision box-plot of 100 trials of U-NoCI and SUPG’s importance sampling algorithm with a precision target of 90%. U-NoCI can fail up to 75% of the time. Furthermore, it can return precisions as low as 20%.

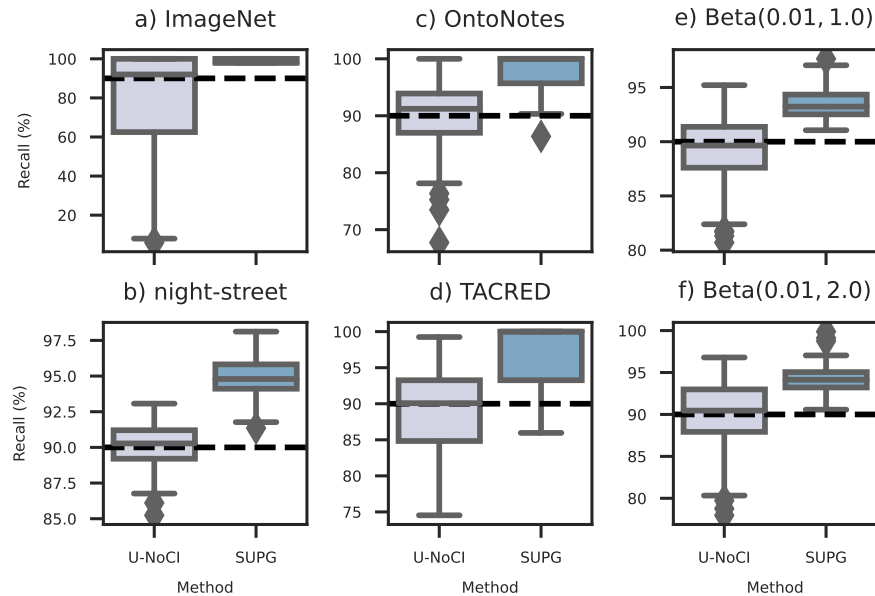


Figure 3.8: Recall of 100 trials of U-NoCI and SUPG’s sampling algorithm with a recall target of 90%. U-NoCI can fail up to 50% of the time and even catastrophically fail on ImageNet, returning a recall of as low as 20%.

Table 3.5: Achieved accuracy of queries when using the empirical cutoff method and SUPG on data with distributional shift. The naive algorithm *deterministically fails* to achieve the targets, i.e., has a failure rate of 100%.

Dataset	Query type	Target	Naive accuracy	SUPG accuracy
ImageNet-C	Precision	95%	77%	100%
ImageNet-C	Recall	95%	54%	100%
night-street	Precision	95%	89%	97%
night-street	Recall	95%	89%	96%
Beta	Precision	95%	89%	100%
Beta	Recall	95%	90%	98%

different day of video) and synthetic drift (change of Beta parameters).

As shown in Table 3.5, baseline methods that do not use labels from the shifted dataset fail to achieve the target in all settings, even under mild conditions such as different days of a video. In fact, using the empirical cutoff in U-NOCI can result in achieved targets as much as 41% lower. In contrast, our algorithms will always respect the failure probability despite model drift, addressing a limitation in prior work [11, 83, 93, 94, 124].

SUPG Outperforms Uniform Sampling

We show that SUPG’s novel algorithms for selection outperforms U-CI (i.e., uniform sampling with guarantees) in both the precision target and recall target settings. Recall that the goal is to maximize or minimize the size of the returned set in the precision target and recall target settings, respectively.

Precision target setting. For the datasets and models described in Table 3.3, we executed U-CI, one-stage importance sampling, and two-stage importance. We used a budget of 1,000 oracle queries for ImageNet and 10,000 for **night-street** and the synthetic dataset. We targeted precisions of 0.75, 0.8, 0.9, 0.95, and 0.99.

We show the achieved precision and recall for the various methods in Figure 3.9. As shown, the importance sampling method outperforms U-CI in all cases. Furthermore, the two-stage algorithm outperforms or matches the one-stage algorithm in all cases except ImageNet. While the specific recalls that are achieved vary per dataset, this is largely due to the performance of the proxy model.

We note that the ImageNet dataset and proxy model are especially favorable to SUPG’s importance sampling algorithms. This dataset has a true positive rate of 0.1% and a highly

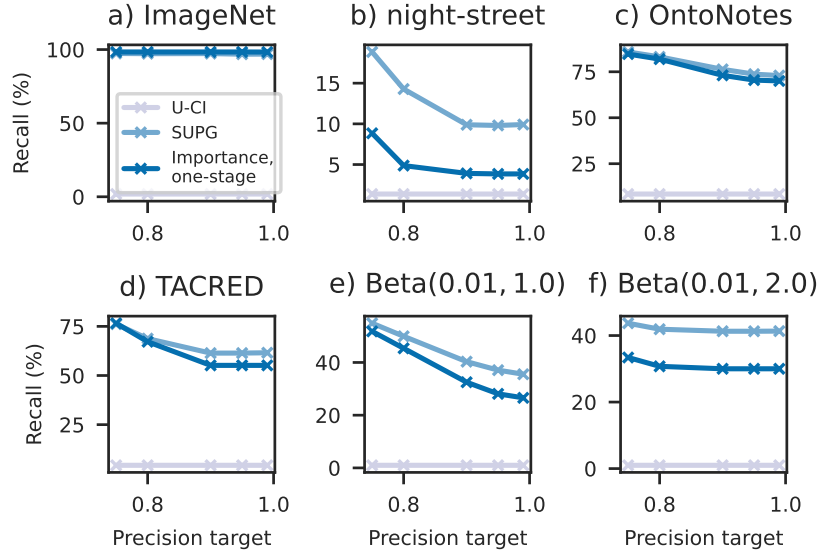


Figure 3.9: Targeted precision vs achieved recall. Both importance sampling methods outperform U-CI in all cases. Two-stage importance sampling outperforms all methods and matches the one-stage importance sampling for ImageNet.

calibrated proxy. A low true positive rate will result in uniform sampling drawing few positives. In contrast, a highly calibrated proxy will result in many positive draws for importance sampling.

Recall target setting. For the datasets and models in Table 3.3, we executed U-CI, standard importance sampling with linear weights $\propto A(x)$ (Importance, prop), and the SUPG methods that use sqrt weights. We used the same budgets as in the precision target setting. We targeted recalls of 0.5, 0.6, 0.7, 0.75, 0.8, 0.9, and 0.95.

We show the achieved recall and the returned set size for the various methods in Figure 3.10. As shown, the importance sampling method outperforms U-CI in all cases. Furthermore, using $\sqrt{A(x)}$ weights outperforms using linear weights in all cases.

3.2.6 Discussion

In this Section, we developed novel sample-efficient algorithms to execute approximate selection queries *with guarantees*. This is in contrast to prior work, including NOSCOPE, which provides best-effort guarantees. As we show, these best effort algorithms can catastrophically fail. We define query semantics for precision- and recall-target queries with guarantees on failure probabilities. We implement and evaluate our SUPG algorithms, showing that

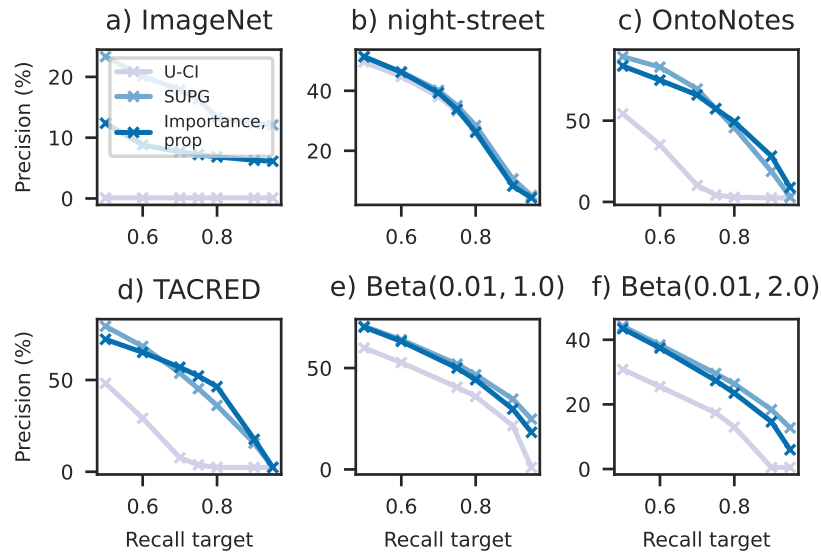


Figure 3.10: Targeted recall vs precision of the returned set. Up and to the right indicates higher performance. Importance sampling outperforms or matches U-CI in all cases. Our sqrt scaling outperforms proportional scaling for importance sampling in all cases, except for high recall settings.

they outperform existing baselines in prior work. These results indicate the promise of probabilistic algorithms to answer selection queries with statistical guarantees.

In the remainder of this chapter, we continue to develop algorithms for other query types, including aggregation and limit queries.

3.3 Approximate Aggregation with Predicates

In addition to answering selection queries, analysts are interested in computing statistics over large, unstructured datasets where only a fraction of the data is of interest (i.e., with a selective predicate) with low computational cost. For example, a media studies researcher may be interested in computing the average viewership (the statistic) of presidential candidates on TV news (the predicate) [80]. To answer such queries, the researcher may deploy an expensive face detection deep neural network (DNN) to find all faces in the dataset and filter by presidential candidates, e.g.,

```
SELECT AVG(views) FROM video
WHERE contains_candidate(frame, 'Biden')
```

As discussed in Section 3.2, these predicates are often incredibly expensive to execute. Due

to limited computational budgets, many organizations cannot exhaustively execute these expensive ML methods over the entirety of the dataset.

Fortunately, many applications can tolerate approximations (as is standard in the approximate query processing (AQP) literature [118]) so answering queries does not require exhaustively executing the expensive DNN. As is standard in AQP, a key requirement with approximate answers are statistical guarantees on query results. For example, the media studies researcher may require such guarantees to make precise claims about bias in TV news. Furthermore, these requirements are standard in scientific analyses. As such, we focus on queries with statistical guarantees in this work.

Unfortunately, standard techniques in AQP, ranging from histograms [142], sketches [24], and others [7], assume that the fields used in the predicates are already available, i.e., as structured records in a database. In contrast, we cannot precompute results as an expensive ML method is required to compute the predicate in our setting, e.g., we would have to execute an expensive face detector on every video frame to answer the query above. Recent work has focused on using cheap approximations (i.e., proxy models) to accelerate queries without having to pre-compute expensive DNNs [11, 83, 93–95]. For example, a proxy for presidential candidates might be a cheap classifier in contrast to a full object detection DNN. Unfortunately, existing work either does not provide statistical guarantees on query accuracy (e.g., NOSCOPe [94], Focus [83], Tahoma [11]) or accelerates other query types (e.g., selection queries [95], aggregation queries without predicates [93], and limit queries [93]).

We propose and analyze ABAE (**A**ggregation with **E**xpensive **B**inAry **P**rEdicates), a query processing algorithm leveraging stratified and pilot sampling [107] to accelerate linear aggregation queries (**SUM**, **COUNT**, and **AVG**) with expensive predicates and statistical guarantees on query accuracy. We further extend ABAE to support common aggregation patterns, including queries with multiple predicates and with group by keys.

ABAE leverages two key opportunities to accelerate such queries: proxy models and stratified sampling. That is, ABAE splits the dataset into disjoint groups (strata), samples within strata, and computes a weighted average to obtain the final answer. ABAE must account for three key challenges, as the predicate results are not available ahead of time: 1) strata selection, 2) budget allocation between strata, and 3) stochastic draws (i.e., sampling a record that may not match the predicate). We provide a principled stratification approach, leverage pilot sampling for budget allocation [107], and provide a novel analysis of stratified sampling with stochastic draws that shows that ABAE converges at an optimal rate.

To address strata selection, ABAE uses the proxy model. We assume the proxy provides information about the likelihood of a record satisfying the predicate [11, 83, 94, 95]. Since the proxy does not give information about the statistic, we stratify records by proxy score quantile. Under a mild monotonicity assumption on the proxy [67], this stratification will group records that are approximately equally likely to match the predicate in the same stratum. Intuitively, if the proxy is perfect (i.e., matches the predicate) and is independent of the statistic, this stratification will minimize the sampling variance. While ABAE performs best when given proxy models which approximate the expensive predicate well, ABAE still returns correct answers regardless of proxy model quality.

Given a stratification, our analysis shows that the optimal allocation depends on two key, per-strata quantities: the fraction of records that match the predicate (p_k) and the standard deviation of the statistic within a stratum (σ_k). Concretely, the optimal allocation is proportional to $\sqrt{p_k}\sigma_k$. However, we do not know these quantities ahead of time.

ABAE proceeds in two stages to address this challenge. First, ABAE will estimate p_k and σ_k using a fraction of the total sampling budget. Then, ABAE will allocate the sampling budget using our plug-in estimates of p_k and σ_k . We prove that ABAE’s algorithm matches the expected error rates of the optimal stratified sampling allocation given the key quantities. Finally, to provide confidence intervals, ABAE uses a bootstrapping procedure which only adds minimal computational overhead.

We also extend ABAE to support group bys (ABAE-GROUPBY) and complex expressions involving multiple Boolean predicates (ABAE-MULTIPRED). To support group by statements, we adapt our sample allocation strategy to minimize the maximum of the expected mean squared error of the groups (minimax error). We show a numerical optimization procedure can recover the optimal allocation for the minimax error. We also support combining multiple expensive predicates and their respective proxy models through negations, conjunctions, and disjunctions.

Finally, a key challenge in leveraging proxy models is to ensure efficient query answers despite potentially poor proxy model quality. Because ABAE always produces valid results, we only need to address efficiency. To address this challenge, we derive a formula which computes the relative gain of using a given proxy. Then, by using a cheap procedure which can estimate the quantities in the formula, ABAE can calculate expected performance gains of proxy models and select the best proxy model at query time.

We evaluate ABAE and its extensions on six real-world datasets spanning text, images,

and video. We show that ABAE outperforms uniform sampling by up to $2.3\times$. We also provide experiments to show that our methods for creating confidence intervals, executing group by aggregation queries, and forming complex predicates from multiple proxy models outperforms baselines.

In the remainder of this section, we describe ABAE’s query semantics, its query planning, our statistical analysis of ABAE, and our evaluation of ABAE.

3.3.1 Overview and Query Semantics

Overview

Target setting. ABAE targets aggregation queries that contain one or more predicates that are expensive to evaluate. These predicates typically require executing expensive DNNs or querying human labelers. We assume the statistic can be computed in conjunction with the predicates or is cheap to compute. We support aggregation queries targeting **AVG**, **SUM**, and **COUNT** statistics. We do not support other aggregation types, such as **COUNT DISTINCT** or **MAX**.

Proxies. We further assume access to a proxy model per predicate, which returns a continuous value between 0 and 1. While not necessary for correctness, high quality proxies will return scores that are correlated with the predicate. These proxies can be orders of magnitude cheaper than oracles (e.g., over 4,000 images/second for the proxy vs 3 fps for the oracle [100]). Thus, as is standard in the literature, we assume these proxies are substantially cheaper than the oracle methods so the proxies can be exhaustively executed over the entire dataset [30, 93, 94, 181].

TV news example. Consider a media studies researcher studying how the presence of presidential candidate affects viewership. The researcher is willing to query the expensive DNN at most 10,000 times and computes the average viewership with the following:

```
SELECT AVG(views) FROM news
WHERE contains_candidate(frame, 'Biden')
ORACLE LIMIT 10,000 USING proxy(frame)
WITH PROBABILITY 0.95
```

where `contains_candidate` is computed via a face detection DNN and the proxy may be trained via specialization [94].

```

SELECT {AVG | SUM | COUNT} ({{field | Expr(field)}})
FROM table_name WHERE filter_predicate
[GROUP BY key]
ORACLE LIMIT o USING proxy
WITH PROBABILITY p

```

Figure 3.11: Syntax for ABAE. Users provide a statistic to compute, an expensive predicate, an oracle limit, proxy scores, and a success probability. As is standard for aggregation queries, users may specify a group by key.

Query Syntax and Semantics

We show the query syntax for ABAE in Figure 3.11. As with standard AQP systems, ABAE accepts a sampling budget and a probability of error and will return an approximate answer to the query and a confidence interval (CI). Our CI semantics are the standard frequentist CI semantics provided by other AQP systems [7]. In particular, our CI semantics are valid regardless of proxy quality.

In contrast to standard AQP systems, ABAE assumes that the predicate is expensive to evaluate. We refer to the methods to execute the predicates as “oracles” [93, 95]. These oracles typically involve executing an expensive DNN and post-processing the result, e.g., executing Mask R-CNN to extract object types and positions from a frames of video and filtering by frames that contain at least two cars. Other use cases may require a human labeler. We further assume that the statistic is either cheap to compute or can be extracted by post-processing the oracle results.

To accelerate these queries, the user also provides a proxy function that computes per-record proxy scores for each predicate. These proxy scores are ideally correlated with the result of the predicate and substantially cheaper than the oracle predicates. Nonetheless, our algorithms will provide valid results even if the proxy scores are of poor quality: proxy correlation will only affect performance, not correctness.

ABAE aims to return query results that minimize the mean squared error (MSE) between the approximate result and the result when exhaustively executing the query. ABAE further aims to return CIs that are as tight as possible while maintaining the probability of success.

3.3.2 Query Formalism

Formally, let $\mathcal{D} = \{x_i\}$ be the set of data records, $O(x) \in \{0, 1\}$ be the oracle predicate, and $X_i = f(x_i) \in \mathbb{R}$ be the expression the query aggregates over. Let $\mathcal{D}^+ = \{x \in \mathcal{D} : O(x) = 1\}$.

Table 3.6: Summary of notation.

Symbol	Description
\mathcal{D}	Universe of data records
\mathcal{S}	Stratification, i.e., k strata
$\mathcal{P}(x)$	Proxy model
N	User-specified sampling budget
K	Number of strata
$\mathcal{O}(x)$	Oracle predicate
$X_{k,i}$	i th sample from stratum k
p_k	Predicate positive rate
p_{all}	$\sum_k p_k$
w_k	Normalized p_k , i.e., p_k/p_{all}
μ_k	$\mathbb{E}[X_{i,k}]$
μ_{all}	$\sum p_k \mu_k / p_{\text{all}}$
σ_k^2	$\text{Var}[X_{i,k}]$
N_1	Number of samples in Stage 1
N_2	Number of samples in Stage 2

Finally, let N be the sample budget.

ABAE computes $\mu = \sum_{x \in \mathcal{D}^+} f(x)/|\mathcal{D}^+|$ via an approximation, $\hat{\mu}$, with a fixed sampling budget N . We measure query result quality by the MSE, i.e., $|\mu - \hat{\mu}|^2$. ABAE returns a CI $[\underline{\mu}, \bar{\mu}]$. ABAE further aims to minimize the length of the CI $\bar{\mu} - \underline{\mu}$ subject to $\mu \in [\underline{\mu}, \bar{\mu}]$ with the specified probability and sample budget, over randomizations of the query procedure.

3.3.3 Algorithm Description and Query Processing

We describe ABAE for accelerating aggregation queries with expensive predicates. We describe accelerating queries with a single predicate in this manuscript and defer a full description of extensions (queries with a group by key, queries with multiple predicates, and estimating proxy quality) to Kang et al. [96].

Overview

ABAE leverages stratified sampling and pilot sampling [107] to accelerate aggregation queries with expensive predicates. Namely, ABAE splits the dataset into disjoint subsets called strata. Then, ABAE allocates sampling budget to the strata and combines the per-strata estimates to give the final estimate.

Our setting involves three distinct challenges. First, since not all records satisfy the

predicate, we may not sample a valid record. This change, while seemingly simple, changes the optimal allocation and requires new theoretical analysis to prove convergence rates. Second, we must construct the strata without knowing which records satisfy the predicate. Third, we do not know p_k (the predicate positive rate) and σ_k (the standard deviation), which are necessary for computing the optimal allocation.

To address these issues, we leverage a two-stage sampling algorithm. ABAE first estimates the key quantities necessary for optimal allocation: p_k and σ_k (also known as pilot sampling). ABAE then uses these estimates to allocate sampling budget in the Stage 2. We show in Section 3.3.4 that ABAE achieves an optimal rate.

Formal Description

Recall that p_k is the predicate positive rate and that σ_k^2 is the variance of the statistic. Furthermore, recall that \mathcal{D} is the full dataset, $O(x)$ is the oracle predicate, and X_i are the samples. Denote $X_{k,i}$ to be the i th *positive* sample from stratum k .

Additionally, denote K to be the number of strata, N_1 to be the number of samples in Stage 1, and N_2 to be the number of samples in Stage 2, which are parameters to ABAE. ABAE will compute several other quantities, including $p_{\text{all}} = \sum_k p_k$, $w_k = p_k/p_{\text{all}}$ the normalized p_k , and $\mu_k = \mathbb{E}[X_{k,i}]$ the per stratum mean. We summarize the notation in Table 3.6.

We present the pseudocode for the sampling algorithm in Algorithm 6. ABAE creates the strata by ordering the records by proxy score and splitting into K strata by quantile.

ABAE will then perform a two-stage sampling procedure. In Stage 1, ABAE samples N_1 samples from each of the K strata to estimate p_k and σ_k , which are the key quantities for determining optimal allocation. In Stage 2, ABAE will allocate the remaining samples proportional to our estimates of the optimal allocation.

ABAE construct plugin estimates for p_k and μ_k , denoted \hat{p}_k and $\hat{\mu}_k$ respectively. To compute its final estimates, ABAE will use all the samples from Stage 1 and Stage 2 to compute \hat{p}_k and $\hat{\mu}_k$. ABAE will return the estimate $\sum_k \hat{p}_k \hat{\mu}_k / \sum_k \hat{p}_k$ as the approximate answer.

As the final estimates are sensitive to the estimate of p_k , i.e., \hat{p}_k , we find that reusing samples between stages dramatically improves performance.

We defer the proofs of convergence and rates to Section 3.3.4.

Algorithm 6 Pseudocode for ABAE. ABAE proceeds in two stages. It first estimates p_k and σ_k . It then samples according to the estimated optimal allocation, $\hat{T}_k = \sqrt{\hat{p}_k \hat{\sigma}_k} / \sum_{i=1}^K \sqrt{\hat{p}_i \hat{\sigma}_i}$.

```

1: function ABAEINIT( $\mathcal{D}, \mathcal{P}, K$ )
2:    $\mathcal{D} \leftarrow \text{Sort}(\mathcal{D}, \text{key} = \text{lambda } x : \mathcal{P}(x))$ 
3:    $\mathcal{S}_1, \dots, \mathcal{S}_K \leftarrow \text{StratifyByQuantile}(\mathcal{D}, K)$ 
4:   return  $\mathcal{S}$ 
5:
6: function ABAESAMPLE( $\mathcal{S}, \mathcal{O}, K, N_1, N_2, \text{SampleFn}$ )
7:   for each  $k$  in  $[1, \dots, K]$  do ▷ Stage 1
8:      $R_k^{(1)} \leftarrow \text{SampleFn}(\mathcal{S}_k, N_1)$  ▷  $R_k$  are sampled records
9:      $X_k^{(1)} \leftarrow \{f(x) \mid x \in R_k^{(1)}, \mathcal{O}(x) = 1\}$ 
10:     $\hat{\mu}_k \leftarrow \sum_{i=1}^{|X_k^{(1)}|} X_{k,i}^{(1)} / |X_k^{(1)}|$  if  $|X_k^{(1)}| > 0$  else 0
11:     $\hat{p}_k \leftarrow |X_k^{(1)}| / |R_k^{(1)}|$ 
12:     $\hat{\sigma}_k^2 \leftarrow \sum_{i=1}^{|X_k^{(1)}|} \frac{(X_{k,i}^{(1)} - \hat{\mu}_k)^2}{|X_k^{(1)}| - 1}$  if  $|X_k^{(1)}| > 1$  else 0
13:   for each  $k$  in  $[1, \dots, K]$  do
14:      $\hat{T}_k \leftarrow \sqrt{\hat{p}_k \hat{\sigma}_k} / \sum_{i=1}^K \sqrt{\hat{p}_i \hat{\sigma}_i}$ 
15:   for each  $k$  in  $[1, \dots, K]$  do ▷ Stage 2
16:      $R_k^{(2)} \leftarrow R_k^{(1)} + \text{SampleFn}(\mathcal{S}_k, \lfloor N_2 \hat{T}_k \rfloor)$ 
17:      $X_k^{(2)} \leftarrow X_k^{(1)} + \{f(x) \mid x \notin R_k^{(1)}, x \in R_k^{(2)}, \mathcal{O}(x) = 1\}$ 
18:      $\hat{p}_k \leftarrow |X_k^{(2)}| / |R_k^{(2)}|$ 
19:      $\hat{\mu}_k \leftarrow \sum_{i=1}^{|X_k^{(2)}|} X_{k,i}^{(2)} / |X_k^{(2)}|$  if  $|X_k^{(2)}| > 0$  else 0
20:   return  $\sum_{k=1}^K \hat{p}_k \hat{\mu}_k / \sum_{k=1}^K \hat{p}_k, R^{(2)}$ 
21:
22: function ABAE( $\mathcal{D}, \mathcal{O}, \mathcal{P}, K, N_1, N_2$ )
23:    $\mathcal{S} \leftarrow \text{ABAEINIT}(\mathcal{D}, \mathcal{P}, K)$ 
24:    $\text{SampleFn} \leftarrow \text{SampleWithoutReplacement}$ 
25:    $\hat{\mu}, R^{(2)} \leftarrow \text{ABAESAMPLE}(\mathcal{S}, \mathcal{O}, K, N_1, N_2, \text{SampleFn})$ 
26:   return  $\hat{\mu}$ 

```

Confidence Intervals

We use the non-parametric bootstrap [51] to compute confidence intervals, which resamples existing samples. Since the per-stratum samples from both stages of ABAE are independent and identically distributed (i.i.d.), we resample from samples across both stages.

We present the pseudocode for the bootstrap procedure in Algorithm 7. ABAE bootstraps across both stages of the sampling algorithm to form CIs. We formally show the

Algorithm 7 Bootstrap procedure for computing CIs. We resample existing samples over both stages of the algorithm.

```

1: function BOOTSTRAP( $R^{(2)}, \mathcal{O}, K, N_1, N_2, \beta, \alpha$ )
2:   for each b in  $[1, \dots, \beta]$  do  $\triangleright \beta$  is # of bootstrap trials
3:     for each k in  $[1, \dots, K]$  do
4:        $R_k^* \leftarrow \text{SampleWithReplacement}(R_k^{(2)}, |R_k^{(2)}|)$ 
5:        $X_k^* \leftarrow \{f(x) \mid x \in R_k^*, \mathcal{O}(x) = 1\}$ 
6:        $\hat{p}_k^* \leftarrow |X_k^*|/|R_k^*|$ 
7:        $\hat{\mu}_k^* \leftarrow \sum_{i=1}^{|X_k^*|} X_{k,i}^*/|X_k^*|$  if  $|X_k^*| > 0$  else 0
8:        $\hat{\mu}_b \leftarrow \sum_{k=1}^K \hat{p}_k^* \hat{\mu}_k^* / \sum_{k=1}^K \hat{p}_k^*$ 
9:     return Percentile( $\alpha/2, \hat{\mu}$ ), Percentile( $1 - \alpha/2, \hat{\mu}$ )
10:
11: function ABAEWITHCI( $\mathcal{D}, \mathcal{O}, \mathcal{P}, K, N_1, N_2, \beta, \alpha$ )
12:    $\mathcal{S} \leftarrow \text{ABAEINIT}(\mathcal{D}, \mathcal{P}, K)$ 
13:   SampleFn  $\leftarrow \text{SampleWithoutReplacement}$ 
14:    $\hat{\mu}, R^{(2)} \leftarrow \text{ABAESAMPLE}(\mathcal{S}, \mathcal{O}, K, N_1, N_2, \text{SampleFn})$ 
15:   return  $\hat{\mu}, \text{BOOTSTRAP}(R^{(2)}, \mathcal{O}, K, N_1, N_2, \beta, \alpha)$ 

```

validity of the bootstrap in an extended technical report [98]. We further show that our procedure produces CIs that are nominally correct in Section 3.3.5.

In standard AQP, the bootstrap is considered an expensive procedure as it requires resampling and recomputing the statistic. However, in our setting, we assume that the oracle predicate is expensive to execute. As a result, the bootstrap is computationally cheap compared to the cost of obtaining the samples. Concretely, in several of our experiments, executing 1,000 bootstrap trials using unoptimized Python code on a single CPU core is as expensive as executing 2,500 oracle calls on an NVIDIA T4 accelerator, which corresponds to under 0.3% of a medium-sized dataset.

3.3.4 Theoretical Analysis

We present a statistical analysis of ABAE and its extensions. We first show that a related sampling procedure achieves rate $O(\frac{1}{N})$ assuming perfect knowledge of p_k and σ_k . We then show that our sampling procedure matches the rate of the optimal strategy. Finally, we show that our optimization procedure for allocation for group by keys is optimal for the deterministic setting.

We provide the intuition and theorem statements in this manuscript. We defer the full

proofs to an extended technical report [98].

Notation and Preliminaries

Notation. Recall the notation in Table 3.6. We emphasize that $X_{k,i}$ is the i th *positive* sample from stratum k , i.e., the i th sample that satisfies the predicate. Furthermore, recall that μ_k is the per-stratum mean, $p_{\text{all}} = \sum p_k$ be the sum of the p_k , and μ_{all} be the overall mean. Finally, recall that $w_k = p_k/p_{\text{all}}$, the normalized predicate positive rate, which corresponds to the weighting of μ_k to μ_{all} .

Assumptions and properties. We assume $X_{k,i}$ is sub-Gaussian with nonzero standard deviation, a standard assumption for stratified sampling [31]. Sums of sub-Gaussian variables converge with quantitative rates and this assumption widely holds in practice. Centered, bounded random variables are sub-Gaussian. The sub-Gaussian assumption gives the existence of universal constants such that $\mathbb{E}[|X_{k,i}|] \leq C^{(\mu)}$ and $\text{Var}[X_{i,k}] \leq C^{(\sigma^2)}$.

We further assume that $p_{\text{all}} \geq C^{p_{\text{all}}} > 0$, which enforces that at least one stratum has non-vanishing p_k .

Optimal Allocation with Deterministic Draws

We first analyze the setting where we assume perfect knowledge of p_k and σ_k and that we receive a deterministic, per-stratum number of draws given a sampling budget. Specifically, given a budget of $T_k N$ per stratum, we assume that we receive $p_k T_k N$ samples, rounded up. We prove the optimal allocation under a continuous relaxation and the rate when using this optimal allocation.

Proposition 1. Suppose p_k and σ_k are known and we receive $B_k = p_k T_k N$ samples per stratum (up to rounding effects). Then, the choice $T_k = T_k^*$ that minimizes the MSE for the unbiased estimator $\hat{\mu}_{\text{all}} = \sum_k p_k \hat{\mu}_k / \sum_k p_k$ is

$$T_k^* = \frac{\sqrt{p_k} \sigma_k}{\sum_{i=1}^K \sqrt{p_i} \sigma_i} \quad (3.13)$$

Proposition 2. Suppose the conditions in Proposition 1 hold. Then, the squared error

under the allocation T_k^* is

$$\mathbb{E}[(\hat{\mu}_{\text{all}} - \mu_{\text{all}})^2 | B_k = p_k T_k^* N] = \sum_{k=1}^K \frac{w_k^2 \sigma_k^2}{p_k T_k^* N} \quad (3.14)$$

$$= \frac{1}{N p_{\text{all}}^2} \cdot \left(\sum_{k=1}^K \sqrt{p_k} \sigma_k \right)^2 \quad (3.15)$$

Intuitively, these propositions say for deterministic draws, the optimal allocation down-weights the standard importance sampling allocation by a factor of $\sqrt{p_k}$. The resulting MSE decreases linearly with respect to the sample budget and a scaling factor.

We note that uniform sampling with deterministic draws converges at rate $\frac{\sigma^2}{N p_{\text{avg}}}$, where $p_{\text{avg}} = \sum p_k / K$. As a result, stratified sampling offers room for improvement. For example, suppose $p_1 = 1$, $p_k = 0$ for $k \neq 1$, and that $\sigma_k = 1$ for all k . This corresponds to a perfect proxy and conditionally independent draws and statistic. Then, uniform sampling converges at rate $\frac{K}{N}$, in contrast to stratified sampling's rate of $\frac{1}{N}$. This corresponds to a K -fold improvement in rate.

ABAE with a Single Predicate

We analyze ABAE's two stage sampling algorithm, in which we do not know p_k and σ_k . We provide the theorem statement, but defer the full proof to an extended technical report [98]. We assume that N_2 is suitably large relative to N_1 for the remainder of this chapter.

Theorem 2. With high probability over the draws made in Stage 1 and in expectation in Stage 2,

$$\mathbb{E}[(\hat{\mu}_{\text{all}} - \mu_{\text{all}})^2] \leq O\left(\frac{1}{N_1} + \frac{1}{N_2} + \frac{\sqrt{N_1}}{\sqrt{N_2}} \cdot \frac{1}{N_2}\right) \quad (3.16)$$

Furthermore, if $N_1 = N_2$

$$\mathbb{E}[(\hat{\mu}_{\text{all}} - \mu_{\text{all}})^2] \leq O\left(\frac{1}{N}\right) \quad (3.17)$$

Understanding ABAE

We provide an overall proof sketch of the analysis of ABAE and highlight several aspects of the analysis of broader interest.

Proof sketch. Our proof strategy proceeds as follows. We first show that our estimates \hat{p}_k and $\hat{\sigma}_k$ converge to p_k and σ_k in a quantitative way (i.e., with a specific rate). As a result, our estimate for the optimal allocation will also converge in a quantitative way.

Given the estimate for the optimal allocation, we show that the number of draws in Stage 2 for all strata will approach the deterministic number of draws, for p_k large enough (larger than $\frac{1}{N^2}$). We then show that the error converges appropriately for the strata with p_k large enough and that the error for the remaining strata becomes negligible. As a result, our final estimate converges with rate $O(\frac{1}{N})$.

Challenges. We describe several challenges in the analysis of ABAE. Prior work has focused on known, deterministic per-strata costs and variances. In contrast, our problem does not have a cost, but rather a stochastic probability of receiving useful information. We study this stochastic draw case and prove that using pilot sampling with plug-in estimates [107] is valid and near optimal.

Most work in stratified sampling assumes that features of the data distributions within each stratum are known and constructs optimal allocations of samples using this information. In our setting, these quantities must be estimated, which may not be possible when p_k is small. For example, if $p_k = \frac{1}{N^2}$ for some stratum, then we may not draw even a single positive record from that stratum, making p_k and σ_k impossible to estimate.

Furthermore, in contrast to standard stratified sampling, we may draw a record that does not satisfy the predicate. As a result, for a fixed number of draws, the number of records matching a predicate is stochastic. Most work on stratified sampling assumes a deterministic allocation of samples to strata.

When the number of draws for some arbitrary M from a stratum and the probability p_k of matching the predicate are both large, the number of positive records concentrates around $p_k M$ and the resulting estimator has similar properties to one with $p_k M$ deterministic draws. However, if $p_k M$ is small, this analysis breaks down.

Finally, show that ABAE converges at the optimal rate, we compare to the setting of deterministic draws and perfect information. Given perfect information of p_k and σ_k , the optimal allocation is given by Proposition 1 and its MSE is given by Proposition 2. However, this allocation cannot be achieved in general, as it results in fractional sampling. Nonetheless, we show that our sampling procedure, which rounds down the ideal fractional allocations, achieves the same $O(\frac{1}{N})$ rate. Thus, rounding does not affect the convergence rate of our procedure.

Statistical intuition. Our primary tool for dealing with unknown quantities and stochastic draws is dividing the strata into groups: where p_k is large and where p_k is small. Since the number of positive draws is Binomial, we apply standard convergence to the total number of positive draws when p_k is large. For stratum where p_k is small, the contribution of that stratum to the total error is at most $p_k C^{(\mu)}$, which does not increase the asymptotic error. To illustrate our technique, consider the following proposition.

Proposition 3. Recall that N_1 and N_2 are the number of samples in Stages 1 and 2 respectively. With high probability in Stage 1 and if N_1 is a constant multiple of N_2 as N grows, the MSE of the error in Stage 2 can be written as

$$\mathbb{E} [(\hat{\mu}_{\text{all}} - \mu_{\text{all}})^2] = \sum_{k=1}^K \hat{w}_k^2 \text{Var}(\hat{\mu}_k) + O\left(\frac{1}{N_1} + \frac{1}{N_2}\right) \quad (3.18)$$

where $\hat{w}_k = \hat{p}_k / \sum \hat{p}_k$.

As shown in Eq. 3.18, the overall MSE is bounded above by the sum of $\hat{w}_k^2 \text{Var}(\hat{\mu}_k)$, which are per-strata quantities. We then bound these quantities for strata where p_k is (quantitatively) large or small. Specifically, define $p_* = \frac{2 \ln(1/\delta) + 2\sqrt{\ln(1/\delta)} + 2}{N_1} = O\left(\frac{1}{N_1}\right)$ for failure probability δ . Furthermore, we assume that N_1 is a constant multiple of N_2 as N grows. We divide the strata into cases based on whether $p_k > p_*$ or $p_k \leq p_*$.

Consider the case where $p_k > p_*$. By standard concentration arguments, the number of positive samples in Stage 2 concentrates to its expectation, which is large. Thus, $\hat{w}_k^2 \text{Var}(\hat{\mu}_k)$ decays at rate (approximately) $O\left(\frac{1}{N_2}\right)$ by standard concentration arguments. For $p_k \leq p_*$, we can directly bound the contribution. To understand this, consider the following proposition.

Proposition 4.

$$\hat{w}_k^2 \text{Var}[\hat{\mu}_k] \leq \hat{w}_k^2 \left(\mathbb{E} \left[\frac{\sigma_k^2}{B_k^{(2)}} \mid B_k^{(2)} > 0 \right] + P(B_k^{(2)} = 0) \mu_k^2 \right) \quad (3.19)$$

$$\leq O\left(\frac{1}{N_1} + \frac{1}{N_2} + \frac{\sqrt{N_1}}{\sqrt{N_2}} \cdot \frac{1}{N_2}\right) \quad (3.20)$$

where $B_k^{(2)}$ is the number of positive draws in Stage 2.

Proof sketch. The key challenge is bounding quantities involving $B_k^{(2)}$. Suppose counterfactually that $B_k^{(2)}$ were deterministic: then the expression would correspond to the standard

variance of an i.i.d. estimator. The variance of an i.i.d. estimator decays as $1/B_k^{(2)}$. However, since we obtain a stochastic number of draws, we must condition on the event of non-zero draws and take the expectation. Since the draws are binomial in distribution, the leading order converges a.s. to its mean value, which would give the desired bound in this toy setting.

We now adapt this strategy to account for $B_k^{(2)}$ being stochastic in ABAE. For $p_k > p_*$, $B_k^{(2)}$ is approximately $p_k T_k^* N_2$ with high probability. As a result, with high probability, each stratum had sufficient samples to form estimates. We can complete the proof similarly to the toy setting with deterministic $B_k^{(2)}$.

However, if $p_k < p_*$, we may not draw the requisite number of samples. For example, if $p_k = \frac{1}{N^2}$, we would not obtain any samples on average. Thus, our analysis must consider the case where $p_k < p_*$ separately. When p_k is small, we can directly bound the contribution of the sum. Namely, $\hat{w}_k^2 = O(1/N^2)$ as $p_k \leq \frac{C_1}{N}$ and the remainder of the quantities are bounded by a constant.

The bound follows from considering the strata where p_k is small and where p_k is large. \square

3.3.5 Evaluation

We evaluate ABAE and its extensions on six real world datasets and synthetic datasets. We first describe the experimental setup and baselines. We then demonstrate that ABAE outperforms baselines in all settings we consider. For brevity, we defer experiments showing that ABAE’s sample reuse is effective and ABAE is not sensitive to hyperparameters.

Experimental Setup

Datasets, target DNNs, and proxies. We consider six real world datasets, including text, still images, and videos. We additionally consider synthetic datasets for some settings.

We used the `night-street` (also known as `jackson`) and `taipei` video datasets, which are commonly used for video analytics evaluation [30, 86, 93, 94, 181]. We executed the following query:

```
SELECT AVG(count_cars(frame)) FROM video
WHERE count_cars(frame) > 0
```

which computes the average number of cars in the video, where a car is present. We use Mask R-CNN to compute the oracle [73] and an efficient index for the proxy scores [97].

We used the `celeba` dataset [121], an image dataset of celebrity faces that contains annotations of celebrity names and other attributes, such as hair color. We executed the following query:

```
SELECT PERCENTAGE(is_smiling(img)) FROM images
WHERE hair_color(img) = 'blonde'
```

which computes the fraction of images where the celebrity is smiling conditioned on the celebrity having blonde hair. We used the human labels in the `celeba` dataset as the ground truth. We used a specialized MobileNetV2 [155] as the proxy.

We used the TREC public spam corpora from 2005 (`trec05p`) [43]. We used the SPAM25 subset. We executed the following query:

```
SELECT AVG(NB_LINKS(text)) FROM emails
WHERE is_spam(text)
```

which computes the average number of links for spam emails. We used human labels as ground truth. We used a manual, keyword-based proxy based on the presence of words (e.g., “money,” “please”).

We used Amazon movie reviews and posters, which was generated from the Amazon reviews dataset [132]. We scraped the movie posters from the metadata and excluded reviews that did not have posters. We executed the following query:

```
SELECT AVG(rating) FROM movies
WHERE face_exists(posters) AND gender(posters) = 'female'
```

which computes the average rating of posters with a female actress. We use MT-CNN to extract faces [185] and VGGFace pretrained from deepface [157] to classifier gender as the ground truth. We use a specialized MobileNetV2 as a proxy [155].

We used the Amazon reviews dataset [132] which is a dataset of textual reviews from Amazon. We subset to the office supplies reviews. We executed the following query:

```
SELECT AVG(rating) FROM data
WHERE sentiment(review) = 'strongly_positive'
```

which computes the average rating of reviews with strongly positive sentiment. We use a BERT-based sentiment classifier provided by FlairNLP to compute the oracle filter [8] and the NLTK sentiment predictor, a simple rule-based classifier, for the proxy [84].

Metrics. Our primary metric is the *RMSE* of the true and estimated values: we use the RMSE so that the units are on the same scale as the original value. We additionally compare

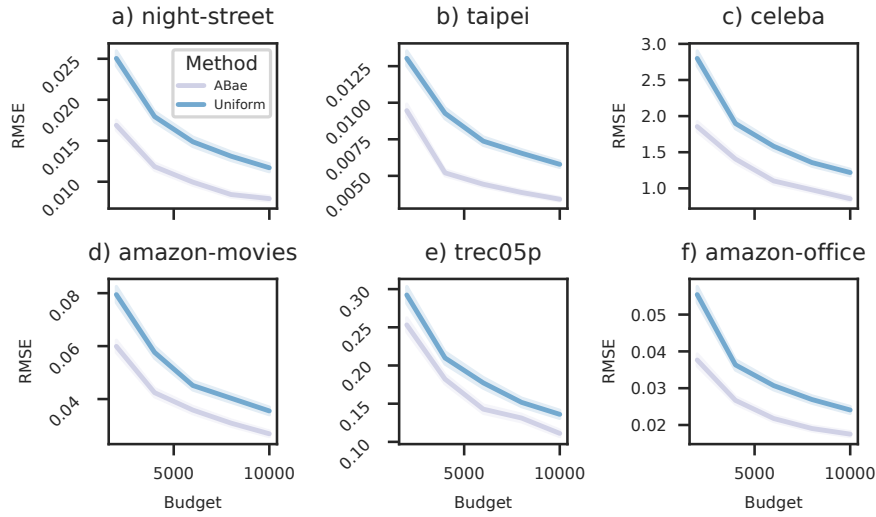


Figure 3.12: Sampling budget vs RMSE for uniform sampling and ABAE. ABAE outperforms on all budgets and datasets we evaluated on. ABAE can outperform by up to $1.5\times$ on RMSE at a fixed budget and achieve the same error with up to $2\times$ fewer samples.

the number of samples required to achieve a particular error target in some experiments. We measure the cost in terms of oracle predicate invocations as it is the dominant cost of query execution by orders of magnitude.

Methods evaluated. We compare ABAE to uniform sampling. A range of standard AQP techniques are not applicable to our setting, since the results of the predicate are not available at ingest time. For example, techniques that create histograms [45, 142, 144] or sketches [58, 59] as ingest time are not applicable.

Implementation. We implement ABAE’s sampling procedure in Python for ease of integration with deep learning frameworks. Our open-sourced code is available at <https://github.com/stanford-futuredata/abae>.

End-to-end Performance

Single predicate. We show that ABAE outperforms uniform sampling on the metric of RMSE. For each dataset and query, we executed ABAE and random sampling for sampling budgets of 2,000 to 10,000 in increments of 2,000. We used five strata and allocated half budget to each stage. We used a failure probability of 5% for every condition. We ran every condition 1,000 times.

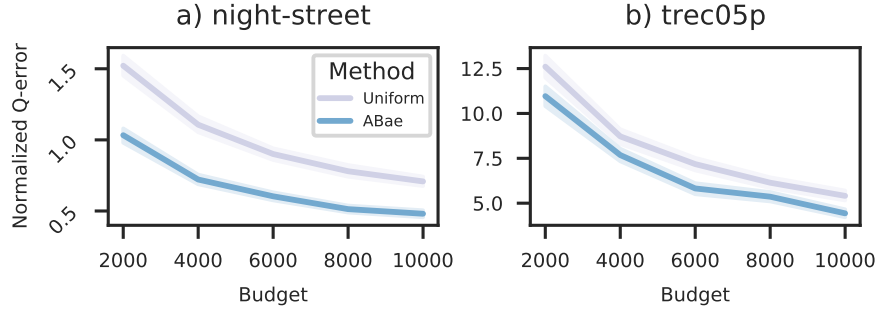


Figure 3.13: Sampling budget vs normalized Q-error for uniform sampling and ABAE, with the standard deviation shaded. We see that ABAE outperforms on Q-error. The same trends hold for all other datasets.

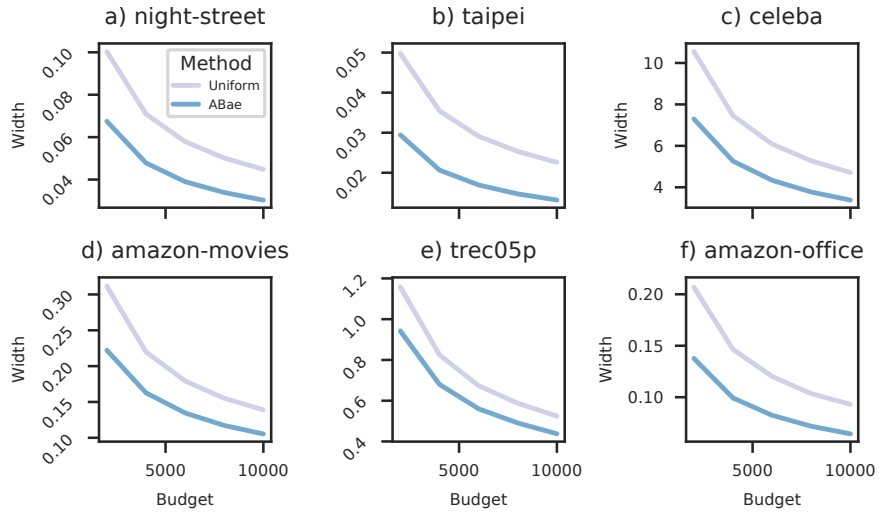


Figure 3.14: Sampling budget vs CI width for uniform sampling and ABAE. ABAE can outperform by up to $1.5\times$ on CI width at a fixed budget and achieve the same width with up to $2\times$ fewer samples.

As shown in Figure 3.12, ABAE outperforms for every dataset, query, and budget setting we consider. ABAE can achieve up to $2.3\times$ improvements in RMSE at a fixed budget or up to $2\times$ fewer samples at a fixed error rate.

We further show that ABAE outperforms on Q-error [130], a relative error metric that penalizes under- and over-estimation symmetrically. We show the normalized Q-error (i.e., $100 \times (q - 1)$) in Figure 3.13. As shown, ABAE outperforms on the two datasets we show—ABAE also outperforms on all the other datasets by 14-70%, which we omit for brevity. ABAE similarly outperforms on relative error by 13-76%.

We further show that ABAE outperforms on the metric of confidence interval (CI) width. For each dataset and query, we executed ABAE and random sampling with the parameters above. We ran every condition 1,000 times.

ABAE outperforms for every dataset, query, and budget setting we consider (Figure 3.14). ABAE can outperform by up to $1.5\times$ on CI width at a fixed budget. Furthermore, to achieve the same confidence interval width, ABAE can use up to $2\times$ fewer samples. Finally, ABAE satisfies the nominal coverage across all datasets and settings.

Discussion of results. To contextualize our results, we first note that relative errors for some of the datasets are as high as 12% (Figure 3.13b). A $2\times$ decrease in error (or number of samples at a fixed error) represents a substantial improvement. Our ongoing collaborations at Stanford University and elsewhere require expert human labeling as part of scientific studies. Requiring $2\times$ fewer human labels is a substantial decrease.

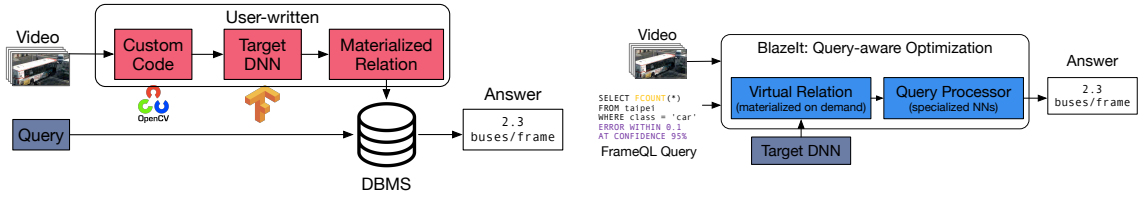
3.3.6 Discussion

In this section, we describe query semantics for approximate aggregation queries with expensive predicates. Unfortunately, prior work on approximate selection does not directly answer these queries with guarantees on query accuracy.

To address this, we develop novel sample-efficient algorithms to reduce the cost of approximate aggregation queries with expensive predicates. Our algorithm leverage proxy models to accelerate such queries. We provide proofs of convergence in our setting: stratified sampling with stochastic draws. We show that ABAE achieves optimal rates. ABAE outperforms baselines by up to $2.3\times$ on a wide range of domains and predicates.

3.4 BLAZEIT

In the proceeding sections, I have described how to generate proxies in a per-query manner, accelerate selection queries, and accelerate aggregation queries with a predicate. However, these optimizations do not handle two key classes of queries: aggregate and limit queries. For example, an analyst may want to count the average number of cars per frame (aggregate query) or manually inspect only 10 instances of a bus and five cars (limit query) to understand congestion. Approximate filtering is inefficient for these queries, e.g., filtering for cars will not significantly speed up counting cars if 90% of the frames contain cars. Furthermore, these optimizations still require non-expert users to write complex code to deploy.



(a) Schematic of the naive method of querying video. Naively using DNNs or human annotators is too expensive for many applications.

(b) Schematic of BLAZEIT. BLAZEIT will create optimized query plans and avoid calling the expensive DNN where possible.

Figure 3.15: Schematic of the naive method of querying video and BLAZEIT. BLAZEIT does not require writing complex code and does not require pre-materializing all the tuples.

To address these challenges, I describe BLAZEIT, a video analytics system with a declarative query language and two novel optimizations for aggregation and limit queries. BLAZEIT’s declarative query language, FRAMEQL, extends SQL with video-specific functionality and allows users familiar with SQL to issue video analytics queries. Since queries are expressed declaratively, BLAZEIT can *automatically* optimize them end-to-end with its query optimizer and execution engine. Finally, BLAZEIT provides two novel optimizations for aggregation and limit queries that outperforms prior work, including NOSCOPE [94] and approximate query processing (AQP), by up to $83\times$.

FRAMEQL allows users to query information of objects in video through a *virtual* relation. Instead of fully materializing the FRAMEQL relation, BLAZEIT uses optimizations to reduce the number of object detection invocations while meeting an accuracy guarantee based on the specification of the FRAMEQL query (Figure 3.15). FRAMEQL’s relation represents the information of positions and classes of objects in the video. Given this relation, FRAMEQL can express selection queries in prior work [11, 30, 94, 124], along with new classes of queries, including aggregation and limit queries (Section 3.4.2).

Our first optimization, to answer aggregation queries, uses query-specific NNs (i.e., specialized NNs [94]) as a control variate or to directly answer queries (Section 3.4.4). Control variates are a variance reduction technique that uses an auxiliary random variable that is correlated with the statistic of interest to reduce the number of samples necessary for a given error bound [69]. We show how to use specialized NNs as a control variate, a novel use of specialized NNs (which have been used for approximate filtering). In contrast, standard random sampling does not leverage proxy models and prior work (filtering) is inefficient when objects appear frequently.

Our second optimization, to answer cardinality-limited queries (e.g., a LIMIT query for 10 frames with at least three cars), evaluates object detection on frames that are more likely to contain the event using proxy models (Section 3.4.5). By prioritizing frames to search over, BLAZEIT can achieve exact answers while speeding up query execution. In contrast, filtering is inefficient for frequent objects and random sampling is inefficient for rare events.

Importantly, both of our optimizations provide exact answers or accuracy guarantees *regardless of the accuracy of the specialized NNs*. Furthermore, both of these optimizations can be extended to account for query predicates.

BLAZEIT incorporates these optimizations in an end-to-end system with a rule-based query optimizer and execution engine that efficiently executes FRAMEQL queries. Given query contents, BLAZEIT will generate an optimized query plan that avoids executing object detection wherever possible, while maintaining the user-specified accuracy (relative to the object detection method as ground truth).

We evaluate BLAZEIT on a variety of queries on four video streams that are widely used in studying video analytics [30, 83, 86, 94, 181] and two new video streams. We show that BLAZEIT can achieve up to 14× and 83× improvement over prior work in video analytics and AQP for aggregation and limit queries respectively.

In the remainder of this Section, we describe BLAZEIT’s architecture, FRAMEQL’s syntax and semantics, BLAZEIT’s query optimization strategies, and our evaluation of BLAZEIT.

3.4.1 BLAZEIT System Overview

BLAZEIT’s goal is to execute FRAMEQL queries as quickly as possible; we describe FRAMEQL in §3.4.2. To execute FRAMEQL queries, BLAZEIT uses a *target object detection method*, an entity resolution method, and the optional user-defined functions (UDFs). We describe the specification of these methods in this section. Importantly, we assume the object detection class types are provided.

BLAZEIT executes queries quickly by avoiding materialization using the techniques described §3.4.4 and §3.4.5. BLAZEIT uses proxy models, typically specialized neural networks [94, 159], to avoid materialization (Figure 3.15b), which we describe below.

Configuration. We assume the target object detection method is implemented with the following API:

$$\text{OD}(\text{frame}) \rightarrow \text{Set}\langle \text{Tuple}\langle \text{class}, \text{box} \rangle \rangle \quad (3.21)$$

and the object classes (i.e., types) are provided. We assume the entity resolution takes two nearby frames and boxes and returns true if the boxes correspond to the same object. While we provide defaults, the object detection and entity resolution methods can be changed, e.g., a license plate reader could be used for resolving the identity of cars. The UDFs can be used to answer more complex queries, such as determining color, filtering by object size or location, or fine-grained classification. UDFs are functions that accept a timestamp, mask, and rectangular set of pixels. For example, to compute the “redness” of an object, a UDF could average the red channel of the pixels.

Target-model annotated set (TMAS). At ingestion time, BLAZEIT will perform object detection over a small sample of frames of the video with the target object detection NN and will store the metadata as FRAMEQL tuples, which we call the *target-model annotated set (TMAS)*. This procedure is done when the data is ingested and not per query, i.e., it is performed once, offline, and shared for multiple queries later. For a given query, BLAZEIT will use the TMAS to materialize training data to train a query-specific proxy model; details are given in §3.4.4 and §3.4.5. The TMAS is split into training data and held-out data.

Proxy models and specialized NNs. BLAZEIT can infer proxy models and/or filters from query predicates, many of which must be trained from data. These proxy models can be used to accelerate query execution *with accuracy guarantees*.

Throughout, we use specialized NNs [94, 159], specifically a miniaturized ResNet [74], as proxy models. A specialized NN is a NN that mimics a larger NN (e.g., Mask R-CNN) on a simplified task, e.g., on a marginal distribution of the larger NN. As specialized NNs predict simpler output, they can run dramatically faster.

BLAZEIT will infer if a specialized NN can be trained from the query specification. For example, to replicate NOSCOPE’s binary detection, BLAZEIT would infer that there is a predicate for whether or not there is an object of interest in the frame and train a specialized NN to predict this. Prior work has used specialized NNs for binary detection [71, 94], but we extend specialization for aggregation and limit queries.

3.4.2 FRAMEQL: Expressing Complex Spatiotemporal Visual Queries

To address the need for a unifying query language over video analytics, we introduce FRAMEQL, an extension of SQL for querying spatiotemporal information of objects in video.

Table 3.7: FRAMEQL’s data schema contains spatiotemporal and content information related to objects of interest, as well as metadata (class, identifiers).

Field	Type	Description
<code>timestamp</code>	float	Time stamp
<code>class</code>	string	Object class (e.g., bus, car)
<code>mask</code>	(float, float)*	Polygon containing the object of interest, typically a rectangle
<code>trackid</code>	int	Unique identifier for a continuous time segment when the object is visible
<code>content</code>	float*	Content of pixels in mask
<code>features</code>	float*	The feature vector output by the object detection method.

By providing a table-like schema using the standard relational algebra, we enable users familiar with SQL to query videos, whereas implementing these queries manually would require expertise in deep learning, computer vision, and programming.

FRAMEQL is inspired by prior query languages for video analytics [49, 113, 115, 122], but FRAMEQL specifically targets information that can be populated automatically using computer vision methods. We discuss differences in detail at the end of this section.

FRAMEQL data model. FRAMEQL represents videos (possibly compressed in formats such as H.264) as virtual relations, with one relation per video. Each FRAMEQL tuple corresponds to a single object in a frame. Thus, a frame can have zero or more tuples (i.e., zero or more objects), and the same object can have one or more tuples associated with it (i.e., appear in several frames).

We show FRAMEQL’s data schema in Table 3.7. It contains fields relating to the time, location, object class, and object identifier, the box contents, and the features from the object detection method. BLAZEIT can automatically populate `mask`, `class`, and `features` from the object detection method (see Eq. 3.21), `trackid` from the entity resolution method, and `timestamp` and `content` from the video metadata. Users can override the default object detection and entity resolution methods. For example, an ornithologist may use an object detector that can detect different species of birds, but an autonomous vehicle analyst may not need to detect birds at all.

FRAMEQL query format. FRAMEQL allows selection, projection, and aggregation of objects, and, by returning relations, can be composed with standard relational operators. We show the FRAMEQL syntax in Figure 3.16. FRAMEQL extends SQL in three ways: `GAP`, syntax for specifying an error tolerance (e.g., `ERROR WITHIN`), and `FCOUNT`. Notably, we do not support joins as we do not optimize for joins in this work, but we describe how to extend

```

SELECT * | expression [, ...]
FROM table_name
[ WHERE condition ]
[ GROUP BY expression [, ...] ]
[ HAVING condition [, ...] ]
[ LIMIT count ]
[ GAP count ]
[ ERROR WITHIN tol AT CONFIDENCE conf ]

```

Figure 3.16: FRAMEQL syntax. As shown, FRAMEQL largely inherits SQL syntax.

Table 3.8: Additional syntactic elements in FRAMEQL.

Syntactic	Description
FCOUNT	Frame-averaged count (equivalent to time-averaged count), i.e., $\text{COUNT}(\ast) / \text{MAX}(\text{timestamp})$
ERROR WITHIN	Absolute error tolerance
FPR WITHIN	Allowed false positive rate
FNR WITHIN	Allowed false negative rate
CONFIDENCE	Confidence level
GAP	Minimum distance between returned frames

FRAMEQL with joins in an extended version of this chapter [93]. We show FRAMEQL’s extensions Table 3.8; several were taken from BlinkDB [7]. We provide the motivation behind each additional piece of syntax.

First, when the user selects timestamps, the **GAP** keyword ensures that the returned frames are at least **GAP** frames apart. For example, if 10 consecutive frames contain the event and **GAP** = 100, only one frame of the 10 frames would be returned.

Second, as in BlinkDB [7], users may wish to have fast responses to exploratory queries and may tolerate some error. Thus, we allow the user to specify error bounds in the form of maximum absolute error, false positive error, and false negative error, along with a specified confidence level. NOSCOPe’s pipeline can be replicated with FRAMEQL using these constructs. We choose absolute error bounds in this work as the user may inadvertently execute a query with 0 records, which would require scanning the entire video (§3.4.4).

We also provide a short-hand for returning a frame-averaged count, which we denote as **FCOUNT**. For example, consider two videos: 1) a 10,000 frame video with one car in every frame, 2) a 10 frame video with a car only in the first frame. **FCOUNT** would return 1 in the first video and 0.1 in the second video. As videos vary in length, this allows for a normalized way of computing errors. **FCOUNT** can easily be transformed into a time-averaged count. Window-based analytics can be done using the existing **GROUP BY** keyword.

Table 3.9: A comparison of object detection methods, filters, and speeds. More accurate object detection methods are more expensive.

Method	mAP	FPS
YOLOv2 [149]	25.4	80
Mask R-CNN [73]	45.2	3
Specialized NN	N/A	35k
Decoding low-resol video	N/A	62k
Color filter	N/A	100k

Comparison to prior languages. Prior visual query engines have proposed similar schemas, *but assume that the relation is already populated* [112, 117], i.e., that the data has been created through external means (typically by humans). In contrast, FRAMEQL’s relation can be automatically populated by BLAZEIT. However, as we focus on exploratory queries in this work, FRAMEQL’s schema is *virtual* and rows are only populated as necessary for the query at hand, which is similar to an unmaterialized view. This form of laziness enables a variety of optimizations via query planning.

3.4.3 Query Optimizer Overview

Overview. BLAZEIT’s primary challenge is executing FRAMEQL queries *efficiently*: recall that object detection is the overwhelming bottleneck (Table 3.9). To optimize and execute queries, BLAZEIT inspects query contents to see if optimizations can be applied. For example, BLAZEIT cannot optimize aggregation queries without error bounds, but can optimize aggregation queries with a user-specified error tolerance.

BLAZEIT leverages two novel optimizations to reduce the computational cost of object detection, targeting aggregation (§3.4.4) and limit queries (§3.4.5). As the filters and specialized NNs we consider are cheap compared to the object detection methods, they are almost always worth calling: a filter that runs at 100,000 fps would need to filter 0.003% of the frames to be effective (Table 3.9). Thus, we have found a rule-based optimizer to be sufficient in optimizing FRAMEQL queries.

Both of BLAZEIT’s novel optimizations share a key property: they still provide accuracy guarantees despite using potentially inaccurate specialized NNs. Specifically, both optimization will only speed up query execution and will not affect the accuracy of queries; full details are in §3.4.4 and §3.4.5.

BLAZEIT also can optimize exhaustive selection queries with predicates by implementing optimizations in prior work, such as using NOSCOPE’s specialized NNs as a filter [94, 124]. As this case has been studied, we defer the discussion of BLAZEIT’s query optimization for exhaustive selection to an extended chapter [93].

BLAZEIT’s rule-based optimizer will inspect the query specification to decide which optimizations to apply. First, if the query specification contains an aggregation keyword, e.g., FCOUNT, BLAZEIT will apply our novel optimization for fast aggregation. Second, if the query specification contains the LIMIT keyword, BLAZEIT will apply our novel optimization for limit queries. Finally, for all other queries, BLAZEIT will default to applying filters similar to NOSCOPE’s [94].

Work reuse. In addition to our novel optimizations, BLAZEIT can reuse work by storing the specialized NN model weights and their results. The specialized NNs BLAZEIT uses are small, e.g., < 2 MB, compared to the size of the video.

We describe the intuition, the physical operator(s), its time complexity and correctness, and the operator selection procedure for aggregates (§3.4.4) and limit queries (§3.4.5) below.

3.4.4 Optimizing Aggregates

Overview. In an aggregation query, the user is interested in some statistic over the data, such as the average number of cars per frame. To provide exact answers, BLAZEIT must call object detection on every frame, which is prohibitively slow. However, if the user specifies an error tolerance, BLAZEIT accelerate query execution using two novel optimizations.

We focus on optimizing counting the number of objects in a frame. BLAZEIT requires training data from the TMAS of the desired quantity (e.g., number of cars) to leverage specialized NNs. If there is insufficient training data, BLAZEIT will default to random sampling. If there is sufficient training data, BLAZEIT will first train a specialized NN to estimate the statistic: if the specialized NN is accurate enough, BLAZEIT can return the answer directly. Otherwise, BLAZEIT will use specialized NNs to reduce the variance of AQP via control variates [69], requiring fewer samples. We next describe these steps in detail.

Operator Selection. The process above is formalized in Algorithm 8. BLAZEIT will process the TMAS into training data for a specialized NN by materializing labels, i.e., counts. Given these labels, BLAZEIT first determines whether there is sufficient training data (> 1% of the data has instances of the object) to train a specialized NN. In cases

Algorithm 8 BLAZEIT’s aggregation query procedure. BLAZEIT will use specialized NNs for accelerated query execution via control variates or query rewriting where possible.

```

function BLAZEITAGGREGATION(TMAS, unseen video,  $uerr$ ,  $conf$ )
  if training data has instances of object then
    train specialized NN on TMAS
     $err \leftarrow$  specialized NN error rate
     $\tau \leftarrow$  average of specialized NN over unseen video
    if  $P(err < uerr) < conf$  then
      return  $\tau$ 
    else
       $\hat{m} \leftarrow$  result of Equation 3.22 (control variates)
      return  $\hat{m}$ 
  else
    return result of random sampling.

```

where the training data does not contain enough examples of interest (e.g., a video of a street intersection is unlikely to have bears), BLAZEIT will default to standard random sampling. We use an adaptive sampling algorithm that respects the user’s error bound but can terminate early based on the variance of the sample [129].

When there is sufficient training data, BLAZEIT will train a specialized NN and estimate its error rate on the held-out set. If the error is smaller than the specified error at the confidence level, it will then execute the specialized NN on the unseen data and return the answer directly. For specialized NN execution, BLAZEIT will subsample at twice minimum frequency of objects appearing; the minimum frequency is estimated from the TMAS. Sampling at this rate, i.e., the Nyquist rate [135], will ensure that BLAZEIT will sample all objects. As specialized NNs are significantly faster than object detection, this procedure results in much faster execution.

When the specialized NN is not accurate enough, it is used as a control variate: a cheap-to-compute auxiliary variable correlated with the true statistic. Control variates can approximate the statistic with fewer samples than naive random sampling.

Physical Operators. We describe the procedures for sampling, query rewriting, and control variates below.

Sampling. For approximate queries without sufficient training data for a specialized NN, BLAZEIT samples from the video, populating at most a small number of rows for faster execution. Similar to online aggregation [75], we provide absolute error bounds, but the

algorithm could be easily modified to give relative error bounds. BLAZEIT uses Empirical Bernstein stopping (EBS) [129], which allows for early termination based on the variance, which is useful for control variates. We specifically use Algorithm 3 in [129]; we give an overview of this algorithm in an extended version of this chapter [93].

EBS is an always valid, near-optimal stopping rule for bounded random variables. EBS is always-valid in the sense that when EBS terminates, it will respect the user’s error bound and confidence; the guarantees come from a union bound [129]. EBS is near-optimal in the following sense. Denote the user-defined error and confidence as ϵ and δ . Denote the range of the random variable to be R . EBS will stop within $c \cdot \log \log \frac{1}{\epsilon \cdot |\mu|}$ of any optimal stopping rule that satisfies ϵ and δ . Here, c is a constant and $|\mu|$ is the mean of the random variable.

Query Rewriting via Specialized NNs. In cases where the specialized NN is accurate enough (as determined by the bootstrap on the held-out set; the accuracy of the specialized NN depends on the noisiness of the video and object detection method), BLAZEIT can return the answer directly from the specialized NN run over all the frames for dramatically faster execution and bypass the object detection entirely. BLAZEIT uses multi-class classification for specialized NNs to count the number of objects in a frame.

To train the specialized NN, BLAZEIT selects the number of classes equal to the highest count that is at least 1% of the video plus one. For example, if 1% of the video contains 3 cars, BLAZEIT will train a specialized NN with 4 classes, corresponding to 0, 1, 2, and 3 cars in a frame. BLAZEIT uses 150,000 frames for training and uses SGD with momentum for one epoch with a learning rate of 0.1.

BLAZEIT estimates the error of the specialized NN on a held-out set using the bootstrap [51]. If the error is low enough at the given confidence level, BLAZEIT will process the unseen data using the specialized NN and return the result.

Control Variates. In cases where the user has a stringent error tolerance, specialized NNs may not be accurate enough to answer a query on their own. To reduce the cost of sampling from the object detector, BLAZEIT introduces a novel method of using specialized NNs while still guaranteeing accuracy. In particular, we adapt the method of control variates [69] to video analytics (to our knowledge, control variates have not been applied to database query optimization or video analytics). Specifically, control variates is a method of variance reduction [87, 152]) which uses a proxy variable correlated with the statistic of interest. Intuitively, by reducing the variance of sampling, we can reduce the number of frames that

have to be sampled and processed by the full object detector.

To formalize this intuition, suppose we wish to estimate the expectation m and we have access to an auxiliary variable a . The desiderata for a are that: 1) a is cheaply computable, 2) a is correlated with m (see time complexity). We further assume we can compute $\mathbb{E}[a] = \alpha$ and $\text{Var}(a)$ exactly. Then,

$$\hat{m} = m + c \cdot (a - \alpha) \tag{3.22}$$

is an unbiased estimator of m for any choice of c [69]. The optimal choice of c is $c = -\frac{\text{Cov}(m,a)}{\text{Var}(a)}$ and using this choice of c gives $\text{Var}(\hat{m}) = (1 - \text{Corr}(m,a)^2)\text{Var}(m)$. As an example, suppose $a = m$. Then, $\hat{m} = m + c(m - \mathbb{E}[m]) = \mathbb{E}[m]$ and $\text{Var}(\hat{m}) = 0$.

This formulation works for arbitrary a , but choices where a is correlated with m give the best results. As we show in Section 3.4.6, specialized NNs can provide a correlated signal to the ground-truth object detection method for all queries we consider.

As an example, suppose we wish to count the number of cars per frame. Then, m is the random variable denoting the number of cars the object detection method returns. In BLAZEIT, we train a specialized NN to count the number of cars per frame. Ideally, the specialized NN would exactly match the object detection counts, but this is typically not the case. However, the specialized NNs are typically correlated with the true counts. Thus, the random variable a would be the output of the specialized NN. As our choice of specialized NNs are extremely cheap to compute, we can calculate their mean and variance exactly on all the frames. BLAZEIT estimates $\text{Cov}(m,a)$ at every round.

Aggregation with query predicates. A user might issue an aggregation query with predicates, such as filtering for large red buses. In this case, BLAZEIT will execute a similar procedure above, but first applying the predicates to the training data. The key difference is that in cases where there is not enough training data, BLAZEIT will instead generate a specialized NN to count the most selective set of predicates that contains enough data.

For example, consider a query that counts the number of large red buses. If there is not enough data to train a specialized NN that counts the number of large red buses, BLAZEIT will instead train a specialized NN that counts the number of large buses (or red buses, depending on the training data). If there is no training data for the quantity of interest, BLAZEIT will default to standard sampling.

As control variates only requires that the proxy variable, i.e., the specialized NN in this case, be *correlated* with the statistic of interest, BLAZEIT will return a correct answer even

if it trains a specialized NN that does not directly predict the statistic of interest.

Correctness. The work in [129] proves that EBS is an always valid, near-optimal stopping rule. Briefly, EBS maintains an upper and lower bound of the estimate that always respects the confidence interval and terminates when the error bound is met given the range of the data. We estimate the range from the TMAS. Furthermore, while video is temporally correlated, we assume all the video is present, namely the batch setting. As a result, shuffling the data will result in i.i.d. samples. Control variates are an unbiased estimator for the statistic of interest [69], so standard proofs of correctness apply to control variates.

Query rewriting using specialized NNs will respect the requested error bound and confidence level under the assumption of no model drift.

Time and sample complexity. BLAZEIT must take $c_\delta \frac{\sigma^2}{\epsilon^2}$ samples from a random variable with standard deviation σ (c_δ is a constant that depends on the confidence level and the given video). Denote the standard deviation of random sampling as σ_a and from control variates as σ_c ; the amortized cost of running a specialized NN on a single frame as k_s and of the object detection method as k_o ; the total number of frames as F .

Control variates are beneficial when $k_s F < k_o \frac{c_\delta}{\epsilon^2} (\sigma_a^2 - \sigma_c^2)$. Thus, as the error bound decreases or the difference in variances increases (which typically happens when specialized NNs are more accurate or when σ_a is large), control variates give larger speedups.

While σ_a and σ_c depend on the query, we empirically show in Section 3.4.6 that control variates and query rewriting are beneficial.

3.4.5 Optimizing Limit Queries

Overview. In cardinality-limited queries, the user is interested in finding a limited number of events, (e.g., 10 events of a bus and five cars), typically for manual inspection. Limit queries are especially helpful for rare events. To answer these queries, BLAZEIT could perform object detection over every frame to search for the events. However, if the events occurs infrequently, naive methods of random sampling or sequential scans of the video can be prohibitively slow (e.g., at 30 fps, an event that occurs once every 30 minutes corresponds to a rate of 1.9×10^{-5}).

Our key intuition is to bias the search towards frames that likely contain the event. We use specialized NNs for biased sampling, in a similar vein to techniques from the rare-event simulation literature [90]. As an example of rare-event simulation, consider the probability

of flipping 80 heads out of 100 coin flips. Using a fair coin, the probability of encountering this event is astronomically low (rate of 5.6×10^{-10}), but using a biased coin with $p = 0.8$ can be orders of magnitude more efficient (rate of 1.2×10^{-4}) [90].

Physical operator and selection. BLAZEIT currently supports limit queries searching for at least N of an object class (e.g., at least one bus and at least five cars). In BLAZEIT, we use specialized NNs to bias which frames to sample:

- If there are no instances of the query in the training set, BLAZEIT will default to performing the object detection method over every frame and applying applicable filters as in prior work [94] (random sampling is also possible).
- If there are examples, BLAZEIT will train a specialized NN to recognize frames that satisfy the query.
- BLAZEIT rank orders the unseen data by the confidence from the specialized NN.
- BLAZEIT will perform object detection in the rank order until the requested number of events is found.

BLAZEIT trains a specialized NN to recognize frames that satisfy the query. The training data for the specialized NN is generated in the same way for aggregation queries (Section 3.4.4). While we could train a specialized NN as a binary classifier of the frames that satisfy the predicate and that do not, we have found that rare queries have extreme class imbalance. Thus, we train the specialized NN to predict counts instead, which alleviates the class imbalance issue; this procedure has the additional benefit of allowing the trained specialized NN to be reused for other queries such as aggregation. For example, suppose the user wants to find frames with at least one bus and at least five cars. Then, BLAZEIT trains a single specialized NN to separately count buses and cars. BLAZEIT use the sum of the probability of the frame having at least one bus and at least five cars as its signal. BLAZEIT takes the most confident frames until the requested number of frames is found.

In the case of multiple object classes, BLAZEIT trains a single NN to predict each object class separately (e.g., instead of jointly predicting "car" and "bus", the specialized NN would return a separate confidence for "car" and "bus"), as this results in fewer weights and typically higher performance.

After the results are sorted, the full object detector is applied until the requested number of events is found or all the frames are searched. If the query contains the `GAP` keyword,

once an event is found, the surrounding **GAP** frames are ignored.

Limit queries with multiple predicates. As with aggregation queries, a user might issue a limit query with predicates. If there is sufficient training data in the TMAS, BLAZEIT can execute the procedure above. If there is not sufficient training data, BLAZEIT will train a specialized NN to search for the most selective set of predicates that contains enough data in a similar fashion to generating an aggregation specialized NN.

Correctness. BLAZEIT performs object detection on all sampled frames, so it always returns an exact answer. All frames will be exhaustively searched if there are fewer events than the number requested, which will also be exact.

Time complexity. Denote K to be the number of events the user requested, N the total number of matching events, and F the total number of frames in the video. We denote, for event i , f_i as the frame where the event occurred. Once an event is found, the **GAP** frames around the event can be ignored, but this is negligible in practice so we ignore it in the analysis.

If $K > N$, then every method must consider every frame in the video, i.e., F frames. From here on, we assume $K \leq N$. For sequential scans, f_K frames must be examined. For random sampling, consider the number of frames to find a single event. In expectation, random sampling will consider $\frac{F}{N}$ frames. Under the assumption that $K \ll N \ll F$, then random sampling will consider approximately $\frac{K \cdot F}{N}$ frames.

While using specialized NNs to bias the search does not guarantee faster runtime, we show in Section 3.4.6 that it empirically can reduce the number of frames considered.

3.4.6 Evaluation

We evaluated BLAZEIT on a variety of aggregation and limit FRAMEQL queries on real-world video streams.

Experimental Setup

Evaluation queries and videos. We evaluated BLAZEIT on six videos shown in Table 3.10, which were scraped from YouTube. `taipei`, `night-street`, `amsterdam`, and `archie` are widely used in video analytics systems [30, 83, 86, 94, 181] and we collected two other streams. We only considered times where the object detection method can perform well

Table 3.10: Video streams and object labels queried in our evaluation.

Video name	Object	Occupancy	Avg. duration of object	Distinct count	Resol.	FPS	# Eval frames	Length (hrs)	Detection method	Thresh
taipei	bus	11.9%	2.82s	1749	720p	30	1188k	33	FGFA	0.2
	car	64.4%	1.43s	32367						
night-street	car	28.1%	3.94s	3191	720p	30	973k	27	Mask	0.8
rialto	boat	89.9%	10.7s	5969	720p	30	866k	24	Mask	0.8
grand-canal	boat	57.7%	9.50s	1849	1080p	60	1300k	18	Mask	0.8
amsterdam	car	44.7%	7.88s	3096	720p	30	1188k	33	Mask	0.8
archie	car	51.8%	0.30s	90088	2160p	30	1188k	33	Mask	0.8

(due to lighting conditions), which resulted in 6-11 hours of video per day. These datasets vary in object class (car, bus, boat), occupancy (12% to 90%), and average duration of object appearances (1.4s to 10.7s). For each webcam, we use three days of video: one day for training labels, one day for threshold computation, and one day for testing, as in [94].

We evaluate on a variety of aggregation and limit queries, with the video and object class changed.

Target object detection methods. For each video, we used a pretrained object detection method as the target object detection method, as pretrained NNs do not require collecting additional data or training: collecting data and training is difficult for non-experts. We selected between Mask R-CNN [73] pretrained on MS-COCO [119], FGFA [190] pretrained on ImageNet-Vid [153], and YOLOv2 [149] pretrained on MS-COCO.

We labeled part of each video using Mask R-CNN [73], FGFA [190], and YOLOv2 [149], and manually selected the most accurate method for each video. Mask R-CNN and FGFA are significantly more accurate than YOLOv2, so we did not select YOLOv2 for any video. The chosen object detection method per video was used for all queries for that video.

In timing the naive baseline, we only included the GPU compute time and exclude the time to process the video and convert tuples to FRAMEQL format, as object detection is the overwhelming computational cost.

Evaluation metrics. We computed all accuracy metrics with respect to the object detection method, i.e., we treated the object detection method as ground truth. For aggregation queries, we report the absolute error. For limit queries, we guarantee only true positives are returned, thus we only report throughput.

We have found that modern object detection methods can be accurate at the frame level. Thus, we considered accuracy at the *frame level*, in contrast to the one-second binning that is used in [94] to mitigate label flickering for NOSCOPE.

We measured throughput by timing the complete end-to-end system excluding the time

taken to decode video, as is standard [94, 124]. We assume the TMAPS is computed offline once, so we excluded the time to generate the TMAPS. Unlike in [94], we also show runtime numbers *when the training time of the specialized NN is included*. We include this time as BLAZEIT focuses on exploratory queries, whereas NOSCOPE focuses on long-running streams of data. We additionally show numbers where the training time is excluded, which could be achieved if the specialized NNs were indexed ahead of time.

Hardware environment. We performed our experiments on a server with a single NVIDIA Tesla P100 GPU and two Intel Xeon E5-2690v4 CPUs (56 threads). The system has 504 GB of RAM.

Binary oracle configuration. Many prior visual analytics systems answer binary classification queries, including NOSCOPE, TAHOMA, and probabilistic predicates [83, 94, 124] which are the closest systems to BLAZEIT. These systems cannot directly answer queries in the form of aggregate or limit queries for multiple instances of an object or objects.

As binary classification is not directly applicable to the tasks we consider, where relevant, we compared against a *binary oracle*, a method that returns (on a frame-by-frame basis) whether or not an object class is present in the scene. We assume the oracle is free to query. Thus, this oracle is strictly more powerful—in terms of accuracy and speed—than existing systems. We describe how the binary oracle can be used to answer each type of query.

Aggregates. Binary oracles cannot distinguish between one and several objects, so object detection must be performed on every frame with an object to identify the individual objects. Thus, counting cars in `taipei` would require performing object detection on 64.4% of the frames, i.e., the occupancy rate.

Cardinality-limited queries. As above, a binary oracle can be used to filter frames that do not contain the objects of interest. For example, if the query were searching for at least one bus and at least five cars in `taipei`, a binary oracle can be used to remove frames that do not have a bus and a car. Object detection will then be performed on the remaining frames until the requested number of events is found.

Aggregate Queries

We evaluated BLAZEIT on six aggregate queries across six videos. We ran five variants of each query: 1) Naive: we performed object detection on every frame, 2) Binary oracle: we performed object detection on every frame with the object class present, 3) Naive AQP:

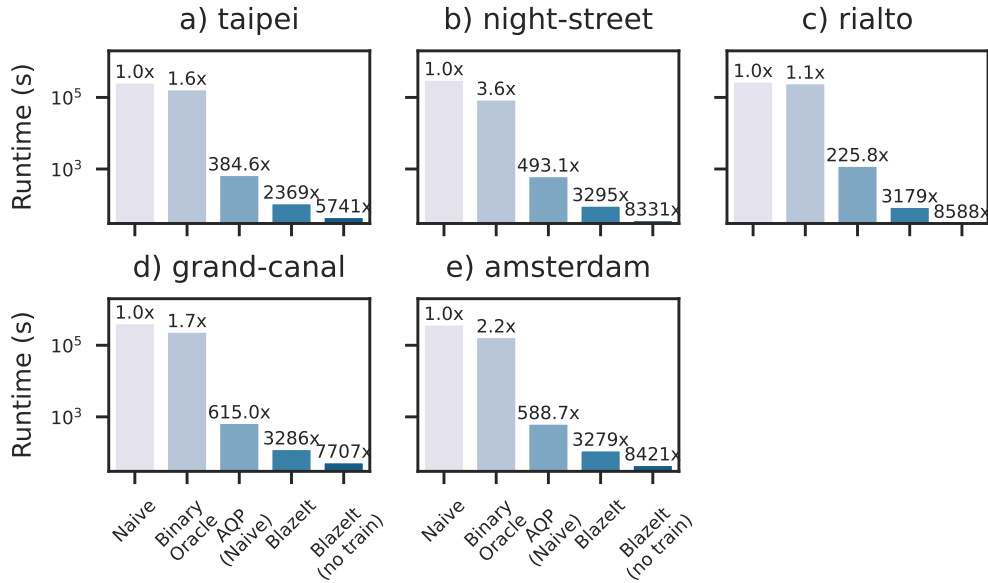


Figure 3.17: End-to-end runtime of aggregate queries where BLAZEIT rewrote the query with a specialized network, measured in seconds (log scale). BLAZEIT outperforms all baselines. All queries targeted $\epsilon = 0.1$.

we randomly sampled from the video, 4) BLAZEIT: we used specialized NNs and control variates for efficient sampling, and 5) BLAZEIT (no train): we excluded the training time.

There are two qualitatively different execution modes: 1) where BLAZEIT rewrites the query using a specialized NN and 2) where BLAZEIT samples using specialized NNs as control variates (Section 3.4.4). We analyzed these cases separately.

Query rewriting via specialized NNs. We evaluated the runtime and accuracy of specialized NNs when the query can be rewritten by using a specialized NN. We ran each query with a target error rate of 0.1 and a confidence level of 95%. We show the average of three runs. Query rewriting was unable to achieve this accuracy for *archie*.

As shown in Figure 3.17, BLAZEIT can outperform naive AQP by up to 14 \times even when including the training time and time to compute thresholds, which the binary oracle does not include. The binary oracle baseline does not perform well when the video has many objects of interest (e.g., *rialto*).

While specialized NNs do not provide error guarantees, we show that the absolute error stays within the 0.1 for the given videos in Table 3.11. This shows that specialized NNs can be used for query rewriting while respecting the user’s error bounds.

Table 3.11: Average error of 3 runs of query-rewriting using a specialized NN for counting. These videos stayed within $\epsilon = 0.1$.

Video Name	Error
taipei	0.043
night-street	0.022
rialto	-0.031
grand-canal	0.081
amsterdam	0.050

Table 3.12: Estimated and true counts for specialized NNs run on two different days of video. In parentheses are the day of video.

Video	Pred (1)	Actual (1)	Pred (2)	Actual (2)
taipei	0.86	0.85	1.21	1.17
night-street	0.76	0.84	0.40	0.38
rialto	2.25	2.15	2.34	2.37
grand-canal	0.95	0.99	0.87	0.81

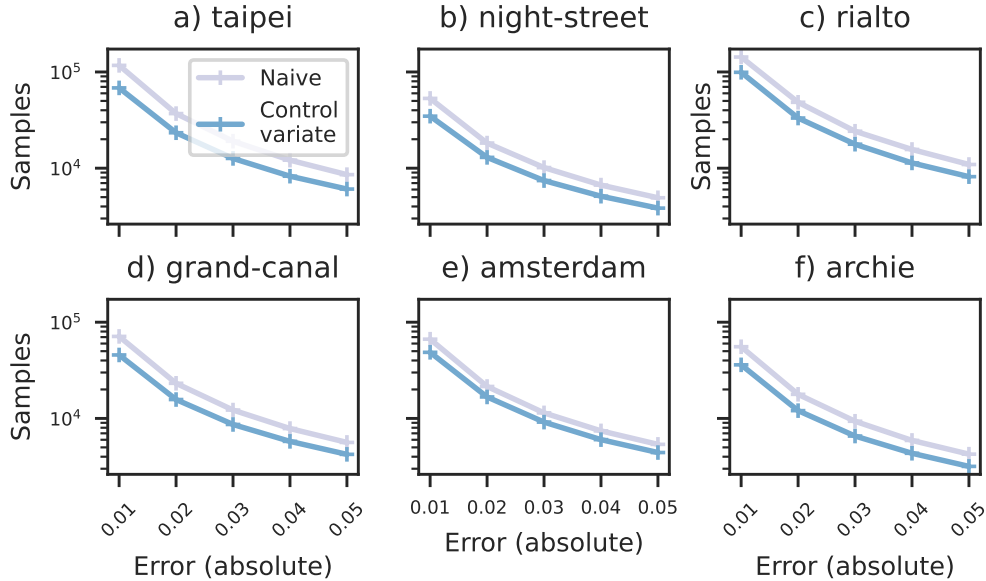


Figure 3.18: Sample complexity of random sampling and BLAZEIT with control variates. Control variates via specialized NNs consistently outperforms standard random sampling. Note the y-axis is on a log scale.

Sampling and control variates. We evaluated the runtime and accuracy of sampling with specialized NNs as a control variate. Because of the high computational cost of running object detection, we ran the object detection method once and recorded the results. The run times in this section are estimated from the number of object detection invocations.

Table 3.13: Query details and number of instances for limit queries.

Video name	Object	Number	Instances
taipei	car	6	70
night-street	car	5	29
rialto	boat	7	51
grand-canal	boat	5	23
amsterdam	car	4	86
archie	car	4	102

We targeted error rates of 0.01, 0.02, 0.03, 0.04, and 0.05 with a confidence level of 95%. We averaged the number of samples for each error level over 100 runs.

Cardinality-limited Queries

We evaluated BLAZEIT on limit queries, in which frames of interest are returned to the user, up to the requested number of frames. We show in Table 3.13 the query details and the number of instances of each query. If the user queries more than the maximum number of instances, BLAZEIT must inspect every frame. Thus, we chose queries with at least 10 instances of the query.

BLAZEIT will only return true positives for limit queries (Section 3.4.5), thus we only report the runtime. Additionally, if we suppose that the videos are indexed with the output of the specialized NNs, we can simply query the frames using information from the index. This scenario might occur when the user executed an aggregate query as above. Thus, we additionally report sample complexity.

We ran the following variants: naive (object detection sequentially until the requested number of frames is found), binary oracle (object detection over the frames containing the object class(es) of interest until the requested number of frames is found), sampling (random sampling the video until the requested number of events is found), BLAZEIT, and BLAZEIT indexed.

Figure 3.19 shows that BLAZEIT can achieve over a 1000 \times speedup compared to baselines. We see that the baselines do poorly in finding rare objects, where BLAZEIT’s specialized NNs can serve as a high-fidelity signal.

We additionally conducted experiments with varying the number of objects, searching for objects with additional predicates, and multiple objects. For brevity, we defer these experiments to Kang et al. [93].

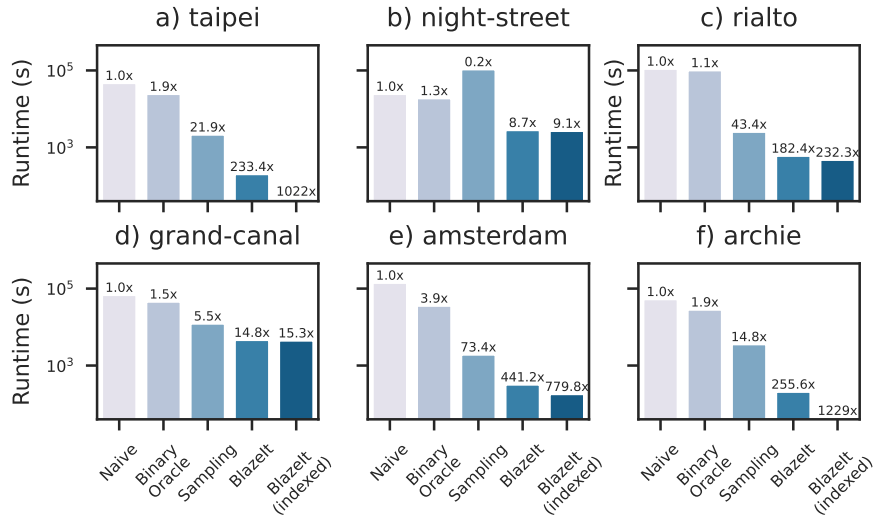


Figure 3.19: End-to-end runtime of baselines and BLAZEIT on limit queries; BLAZEIT outperforms all baselines. The y-axis is log-scaled. All queries looked for 10 events.

3.4.7 Discussion

In this section, we present BLAZEIT, a optimizing video analytics system with a declarative language, FRAMEQL. We introduce two novel optimizations for aggregation and limit queries, which are not supported by prior work. These techniques can run orders of magnitude faster than baselines while retaining accuracy guarantees, despite potentially inaccurate proxy models.

As we have shown in this Section, proxy-based methods can improve on a range of queries with guarantees on accuracy. In the following Section, we describe how to generate these proxies efficiently and execute end-to-end queries efficiently.

Chapter 4

Generating Proxy Scores Efficiently

As I described in Chapter 3, proxies can accelerate unstructured data queries by approximating expensive target models and combining them with sampling algorithms. However, these methods require *high quality* proxies that are ideally *generated efficiently*.

In this chapter, we describe how to generate high-quality proxies efficiently. We first describe a novel approach (TASTI) that uses embeddings to construct an index for proxy-based queries. TASTI can generate proxy scores that can be simultaneously $10\times$ cheaper to construct and $24\times$ more efficient at query time. We then describe our system SMOL, which addresses bottlenecks in end-to-end proxy generation. SMOL can achieve up to $5.9\times$ improved throughput at a fixed accuracy by optimizing across all stages of proxy generation.

4.1 TASTI: Semantic Indexes for Unstructured Data

As we have described, recent work has proposed *query-specific proxy models* to approximate high-quality *target labelers* to reduce query costs. Low-cost proxy models can be used for selecting data records that match a predicate, aggregation queries, and limit queries [11, 16, 83, 93–95, 124]. For each query, a proxy model is trained to generate *proxy scores* for data records, in which the goal is to approximate the result of executing the target labeler on that data record for the particular query. These scores are then used in various algorithms depending on query type. For example, BLAZEIT [93] will train a proxy model to (approximately) count the number of cars per frame of a video to answer the car counting query, and selection algorithms (e.g., NOSCOPE [94], probabilistic predicates [124], SUPG [95], and Tahoma [11]) will train a separate proxy model to (approximately) filter frames

with cars and bicycles for selecting such frames.

Unfortunately, methods based on query-specific proxy models have three key drawbacks. First, obtaining large amounts of training data from the target labeler to train the proxy models can be expensive. For example, BLAZEIT and NOSCOPE require hours of GPU compute to execute the target labeler to produce labels to train the proxy models [93, 94] and other systems require expensive human annotations [11, 83, 124]. Second, these systems require new training procedures for each query type, which can be difficult to develop. Third, query-specific proxy models cannot easily share computation across different queries or query types. Thus, using proxy models can be challenging and computationally expensive.

We observe that this prior work ignores a key opportunity: redundancy present in the *target labeler outputs* of many datasets. For example, two frames with visually distinct cars in the bottom left would have the same result for many queries, e.g., counting the number of cars or selecting cars in the bottom left. Namely, the structured outputs (i.e., target labeler results) of many data records are semantically similar. While fully computing target labeler outputs for all records is expensive, query processing systems would ideally use this similarity to avoid repeated work and target labeler invocations.

To address these issues and leverage this opportunity, we propose **TrAinable SemanTic Indexes** (TASTI). TASTI is an indexing method over unstructured data for accelerating downstream proxy score-based query processing methods via embeddings (i.e., vectors in \mathbb{R}^n). Given the target labeler and a user-provided closeness function over target labeler outputs, TASTI produces embeddings for each unstructured data record (e.g., frame of video), with the desideratum that close records have close embeddings. TASTI’s required closeness function is often easy to specify, e.g., that frames of a video with similar object types and object positions are close (Section 4.1.2).¹ TASTI then uses the embeddings and a small set of records annotated by the target labeler to answer downstream queries.

Specifically, we propose a method of using TASTI’s embeddings and the labeled records (i.e., cluster representatives) to generate proxy scores *automatically*, including for proxy-based aggregation, selection, and limit query processing algorithms (Section 4.1.3) [11, 93–95, 124]. TASTI generates per-record proxy scores by propagating annotations from the cluster representatives to the unlabeled records. For example, we could assign an unannotated frame the number of cars in the closest cluster representative for counting cars. These

¹TASTI can also be used without training an embedding by using pre-trained embeddings, although query performance will generally be better with its training method.

scores can then be used in query processing algorithms, such as those in BLAZEIT, probabilistic predicates, Tahoma, etc. Moreover, as the target labeler is executed over more data during query processing, we can incrementally improve TASTI’s clustering, which improves performance (i.e., TASTI’s indexes support “cracking” [85]).

To understand TASTI’s performance, we provide a theoretical analysis of TASTI and downstream query accuracy (Section 4.1.4). We prove that queries that are Lipschitz-continuous functions of the data will achieve *exact* results when using TASTI (with sufficiently dense clustering) under 0 training loss, and quantitative bounds when the training loss is not 0. Although our assumptions are strong, our analysis provides statistically grounded intuition for why TASTI can outperform baselines. We validate our intuition with extensive experiments (Section 4.1.5).

We implemented TASTI in a prototype system and evaluated it on four datasets, including widely studied video datasets [30, 86, 93, 95, 181], a text dataset [188], and a speech dataset [12]. We integrated TASTI into query processing algorithms for aggregation, selection, and limit queries and executed these queries over the datasets. We show that TASTI’s indexes require up to $10\times$ *fewer* labels from the target labeler to construct than generating training data for per-query proxy model methods, as TASTI leverages redundancy in the datasets. Furthermore, TASTI outperforms on query runtime across all queries and datasets we evaluate on by up to $24\times$ over previous optimized systems.

In the remainder of this section, we give further context on proxy-based query processing, describe TASTI’s system architecture, index construction, and query processing, provide a theoretical analysis of TASTI, and evaluate TASTI on real-world datasets.

4.1.1 Overview and Example

Background and Problem

We first describe how target labelers and proxy scores are used in analytics systems before describing an overview of TASTI.

Many analytics applications over unstructured data are powered by expensive DNNs that extract structured data from these unstructured sources or human labelers (e.g., ground-truth annotations for studying social or life sciences [96]). We refer to these expensive DNNs and human labelers as *target labelers*. These target labelers *induce a schema* over the extracted data. For example, object detection DNNs can extract information about

object types and positions from frames of a video. The schema would contain columns for object type, object (x, y) -positions, and timestamps. Unfortunately these high-quality target labelers, such as Mask R-CNN or BERT, can be expensive and dominate query execution costs.

Thus, recent work attempts to accelerate queries with target labelers by using proxy scores (e.g., BLAZEIT, NOSCOPE, probabilistic predicates, Tahoma, SUPG, etc.). The most common method of producing proxy scores is to train a smaller DNN (also called a “specialized NN” or “proxy model”) that will produce a proxy score per data record. These proxy models are typically trained to approximate the output of the target labeler for the query at hand, e.g., a count for an aggregation query, and can yield substantial query speedups. Constructing the training data can be expensive as the training data must reflect a wide range of potential queries.

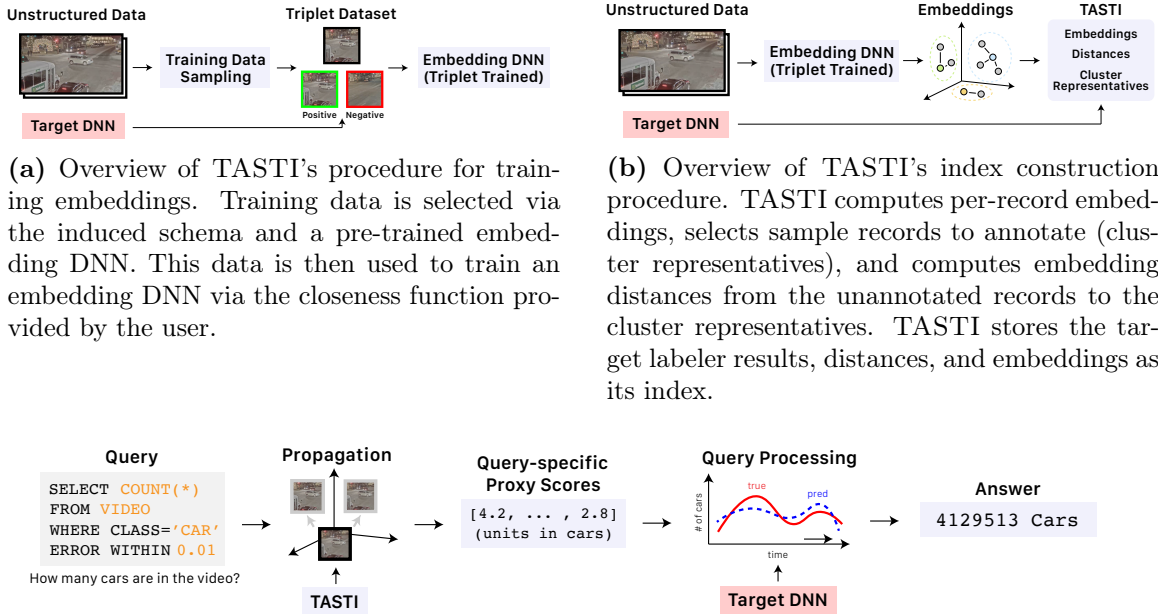
We describe two examples of using proxy scores to accelerate queries, both of which train a new proxy model per query. In both cases (and more generally), the goal is to generate proxy scores that are highly correlated with the target labeler outputs: these algorithms will adaptively improve with better proxy scores.

Approximate aggregation. Suppose the user issues a query for the average number of cars per frame in a video, as studied by BLAZEIT [93]. BLAZEIT takes as input an error target and proxy scores.

To optimize this query, BLAZEIT trains a cheap proxy model whose output is the predicted number of cars per frame using a sample of frames annotated by the target labeler. This proxy model is then used to generate a query-specific proxy score per frame. BLAZEIT then uses these scores as a “control variate” [69] to reduce the variance in estimation. Proxy scores that are more correlated with the true count will result in faster query execution.

Approximate selection. Suppose the user issues a query to select 90% of frames with cars with 95% probability of success, as studied by the “recall target” setting in SUPG [95]. Specifically, SUPG takes as input a target recall and (in contrast to BLAZEIT) a fixed target labeler budget. SUPG will train a proxy model that estimate the probability of a record matching a predicate.

Given proxy scores, SUPG will use importance sampling and return a set of records achieving the recall target. Other recent proxy-based systems accelerate selection queries without guarantees [11, 16, 83, 124].



(a) Overview of TASTI’s procedure for training embeddings. Training data is selected via the induced schema and a pre-trained embedding DNN. This data is then used to train an embedding DNN via the closeness function provided by the user.

(b) Overview of TASTI’s index construction procedure. TASTI computes per-record embeddings, selects sample records to annotate (cluster representatives), and computes embedding distances from the unannotated records to the cluster representatives. TASTI stores the target labeler results, distances, and embeddings as its index.

(c) Overview of TASTI’s query processing. Given a query, TASTI will compute exact results on the cluster representatives. It will then produce proxy scores on unannotated records by propagating the exact scores via embedding distance. These scores can then be used in existing downstream query processing algorithms based on proxy scores.

Figure 4.1: TASTI system overview.

TASTI’s Inputs, Outputs, and Goals

Overview. As input, TASTI takes a target labeler, an induced schema, a target labeler invocation budget for index construction, a closeness function over the induced schema, and a parameter k that specifies how many distances to store for each record. The primary cost in index construction are the target labeler invocations. TASTI will produce an embedding-based index subject to the budget that can produce proxy scores for a range of queries.

TASTI’s primary goal is to produce high quality proxy scores for query processing algorithms, as with per-query proxy models, but *without* training a new model per query.

Supported queries. We demonstrate how to generate proxy scores for selection queries, aggregation queries, and limit queries [11, 93–95, 124] with TASTI. TASTI can be used with other queries requiring proxy scores. Since the initial draft, other work has used TASTI to support aggregation queries with predicates [96].

Example

Consider constructing an index for visual data, in which queries over object types and positions are issued. In this case, the target labeler (e.g., Mask R-CNN) takes an unstructured frame of video and returns a structured set of records that contains fields about the positions and types of objects in the frame. Consider two queries, one of which counts the number of cars per frame (aggregation query as supported by BLAZEIT) and one that selects frames with cars (selection query as supported by NOSCOPE, probabilistic predicates, SUPG, etc.). To understand how the index construction procedure and query processing works, we describe the intuition below.

Index construction. TASTI builds its index by training an embedding DNN for the input data and then clustering results based on it. Ideally, semantically similar records are grouped together, e.g., all frames with two cars might form one cluster, all frames with one bike and one car might form a cluster, etc.

To train an embedding DNN, TASTI requires a heuristic for “close” and “far” target labeler outputs, either as a Boolean function or as a cutoff based on a continuous distance measure. One such heuristic for our video application is to group frames with the same number of objects and similar positions together. The grouping of “close” frames can be specified in pseudocode as follows:

```
def is_close(frame1: List[Box], frame2: List[Box]) -> bool:
    if len(frame1) != len(frame2):
        return False
    return all_boxes_close(frame1, frame2)
```

where `all_boxes_close` is a helper function that returns true if all boxes in `frame1` have a corresponding “close” box in `frame2`. Given the closeness function, TASTI trains a low-cost embedding DNN via the triplet loss [176], which separates “far” frames. Then, TASTI computes embeddings over all frames of the video with the embedding DNN and select a set of frames to annotate with the target labeler. It will then store the target labeler’s outputs (object types and positions) for each cluster representative. TASTI uses the cluster representatives and distances from unannotated frames’ embeddings for downstream query processing.

Query processing. TASTI can now be used to produce proxy scores for a range of downstream queries using existing proxy-based algorithms. First, TASTI generates exact scores on cluster representatives, e.g., the exact count of the number of cars from the cached

target labeler outputs. Then, TASTI propagates these scores to the remainder of the records, e.g., producing approximate counts for the unannotated records. We describe the intuition behind two example queries: in both of these examples, TASTI need not train a new proxy model per query and can reuse its index.

Approximate aggregation. Consider the query of counting the average number of cars per frame. TASTI computes the query-specific proxy score as the distance-weighted average of the number of cars in the k closest annotated frames (see Section 4.1.3 for pseudocode). This will produce an estimate of the number of cars in a given unannotated frame. These scores can then be used by BLAZEIT’s query processing algorithm [93].

Approximate selection. To estimate the probability of matching the predicate (i.e., query-specific proxy score) for SUPG, TASTI will compute the weighted average as above, except that annotated frames that contain a car receive a score of 1 and annotated frames that do not contain a car receive a score of 0. These scores can then be used by SUPG’s selection algorithm [95].

We now discuss TASTI’s index construction method (Section 4.1.2) and TASTI’s query processing method (Section 4.1.3).

4.1.2 Index Construction

We describe how TASTI constructs indexes, which can be used to produce high quality proxy scores without the use of query-specific proxy models. Many queries only require a low dimensional representation of data records to answer, such as object types and positions (as opposed to raw pixels in a video). Furthermore, in many applications, this low dimensional representation has a natural closeness function, which can be directly used to construct high quality proxy scores. TASTI attempts to construct representations that reflect these heuristics by grouping close records and separating far records.

We show a schematic of the training in Figure 4.1a, the index construction in Figure 4.1b, and the overall algorithm in Algorithm 9. TASTI’s index construction procedure consists of optionally training an embedding DNN via the triplet loss, producing embeddings per record, selecting cluster representatives, and computing statistics over the cluster representatives.

Throughout, we use the furthest point first (FPF) clustering algorithm [62]. FPF iteratively chooses the furthest point from the existing cluster representatives as the new representative. FPF performs well in practice, is computationally efficient, and provides a

Algorithm 9 Pseudocode for TASTI’s index construction procedure. Given a dataset X , training points N_1 , number of cluster representatives N_2 , and min- k to retain, TASTI will construct the index as follows. FPF is the furthest point first algorithm, where the arguments are the embeddings and the number of points to select.

```

function MAKE TASTI INDEX( $X, N_1, N_2, k$ )
  PretrainedEmbeddings[ $i$ ]  $\leftarrow$  PretrainedModel( $X[i]$ )
  TrainingPoints  $\leftarrow$  FPF(PretrainedEmbeddings,  $N_1$ )
  TripletModel  $\leftarrow$  Finetune(TrainingPoints, PretrainedModel)
  Embeddings[ $i$ ]  $\leftarrow$  TripletModel( $X[i]$ )
  ClusterRepresentatives  $\leftarrow$  FPF(Embeddings,  $N_2$ )
  MinKDistances[ $i$ ]  $\leftarrow$  ClosestKDistances( $X[i]$ , ClusterRepresentatives,  $k$ )
return ClusterRepresentatives, MinKDistances

```

2-approximation to the optimal maximum intra-cluster distance (which our analysis uses).

Training the Embeddings

TASTI optionally trains a mapping between data records (e.g., frames of a video) and semantic embeddings. The semantic embeddings have the desideratum that data records that have similar extracted attributes are close in embedding space, and vice versa for records that have dissimilar extracted attributes. For example, consider queries over object type and position. A frame with a single car in the upper left should be close to another frame with a single car in the upper left, but far from a frame with two cars in the bottom right.

We describe our training method via domain-specific triplet losses and show a schematic in Figure 4.1a. We note that TASTI’s training procedure is optional: pre-trained embeddings can be also be used for the index if training is expensive.

Domain-specific triplet loss. To train the embedding DNN, TASTI uses the triplet loss [176]. The triplet loss takes an anchor point, a positive example (i.e., a close example), and a negative example (i.e., a far example). It penalizes examples where the anchor point and the positive point are further apart than the anchor point and the negative point (see Section 4.1.4).

A key choice in using the triplet loss is selecting points that are “close” and those that are “far.” This choice is application specific, but many applications have natural choices. For example, any frame of video with different numbers of objects may be far (see Section 4.1.1 for pseudocode). Furthermore, frames with the same number of objects, but where the objects are far apart may also be considered far.

Training data selection (FPF mining). Training via the triplet loss requires invocations of the target labeler to determine whether pairs of records are close or not. Due to the cost of the target labeler, TASTI must sample records to be selected for training; we assume the user provides a budget of target labeler invocations. While TASTI could randomly sample data points, randomly sampled points may mostly sample redundant records (e.g., majority of empty frames) and miss rare events. We empirically show that randomly sampling training data results in embeddings that perform well on average, but can perform poorly on rare events (Section 4.1.5).

To produce embeddings that perform well across queries, we would ideally sample a diverse set of data records. For example, suppose 80% of a video were empty: selecting frames at random would mostly sample empty frames. Selecting frames with a variety of car numbers and positions would be more sample efficient.

When available, TASTI uses a pre-trained DNN to select such diverse points. These pre-trained DNNs are widely available, e.g., DNNs pre-trained on ImageNet [74] or on large text corpora (BERT) [48]. Pre-trained DNNs produce embeddings that are semantically meaningful, although not adapted to the specific induced schema.

To produce training data that results in embeddings that perform well on rare events, TASTI performs the following selection procedure. First, TASTI uses a pre-trained DNN to generate embeddings over the data records. Then, TASTI executes the FPF algorithm to select the training data. TASTI constructs triplets from the training data via target labeler annotations. TASTI will first bucket records by the closeness function. To construct a triplet, TASTI will sample two different buckets at random: it will select the anchor and positive record at random from the first bucket and a negative record at random from the second bucket.

Clustering

TASTI produces clusters via the embedding DNN. As we describe in Section 4.1.3, TASTI propagates annotations/scores from cluster representatives to unannotated data records.

A key choice is which data records to select as cluster representatives. Similar to selecting training data, TASTI could select a set of cluster representatives at random. While random sampling may do well on average at query time, it may perform poorly on rare events (i.e., outliers).

To address this issue, TASTI selects cluster representatives via FPF. FPF chooses points

that are far apart in embedding space. Thus, if the embeddings are semantically meaningful, then FPF will select data records that are diverse. Finally, we mix a small fraction of random clusters, which helps “average-case performance” queries.

TASTI stores the distances of all embeddings to each cluster representative. As we describe in Section 4.1.3, TASTI uses the k nearest cluster representatives for query processing.

Cracking TASTI Indexes

In contrast to prior work, which can only share work between queries in an ad-hoc manner, TASTI’s proxy scores will improve as queries are executed. In particular, when any query executes the target labeler on a data record, TASTI can cache the target labeler result. The records over which the target labeler are executed can then be added as new cluster representatives. Computing the distance to the new cluster representative is computationally efficient and trivially parallelizable. We note that this is a form of “cracking” [85].

Computational Performance

Suppose there are N data records, D dimensions, L training iterations, and a total target labeler budget of C . Denote the costs of the target labeler, embedding DNN, and distance computation as c_T , c_E , and c_D respectively. The total cost of index construction is $O(C \cdot c_T + L \cdot c_E + N \cdot c_E + NCD \cdot c_D)$ assuming the cost of a training iteration is proportional to the cost of the forward pass [91].

The ratio of these steps depends on the relative computational costs. In many applications, the cost of embedding is less expensive than the cost of the target labeler. For example, Mask R-CNN can execute as slow as at 3 fps, compared to an embedding DNN which executes at 12,000 fps [99]. Furthermore, human labelers are orders of magnitude more expensive than embedding DNNs (up to 100,000× more expensive).

4.1.3 Query Processing with TASTI

How can TASTI indexes accelerate query processing? We propose automatic methods of construct query-specific proxy scores with TASTI, which can then be passed to existing proxy score-based algorithms. These query-specific proxy scores are an approximation of the result of executing the target labeler on the data records for the particular query. Consider

an aggregation query counting the average number of cars per frame [93]. The query-specific proxy scores would be an estimate of the number of cars in a given frame.

Many downstream query processing algorithms only require proxy scores and the target labeler. For example, selection without guarantees (e.g., binary detection) [11, 94, 124], selection with statistical guarantees [95], aggregation [93], and limit queries [93] only require query-specific proxy scores and the target labeler.

TASTI provides a default method of taking the target labeler output and producing a numeric score, which can support aggregation, selection and limit queries. Its default method produces exact scores on the cluster representatives and propagates using the distance-weighted average for numeric columns and distance-weighted majority vote for categorical columns. If desired, a developer may also implement custom functions to produce proxy scores for other queries. We describe several examples of how proxy scores can be computed and used for common query types, and then describe the interface for implementing custom proxy scores. We show a schematic of the query processing procedure in Figure 4.1c.

Query Processing Examples

We provide examples of the query-specific scoring functions, score propagation, and downstream query processing for several classes of queries below.

Approximate aggregation. Consider the example of counting the average number of cars per frame, as studied by BLAZEIT [93]. The scoring function would take the detected boxes in a frame and return the count of the boxes matching “car,” as shown above. For $k = 1$, the query-specific proxy score would be the count for the nearest cluster representative and for $k > 1$, it would be the distanced-weighted mean count of the nearest k cluster representatives for a given frame.

The query-specific proxy scores can be used to answer the query with statistical error bounds, e.g., used as a control variate by the BLAZEIT’s query processing algorithm. The scores could also be used to directly answer the query.

Selection. Consider a query that selects all frames of a video with a car, as studied by prior work [11, 94, 95, 124]. The scoring function would take the detected boxes in a frame and return 0 if there are no cars and 1 if there is a car in the frame. The query-specific proxy score can be smoothed for $k > 1$.

The query-specific proxy scores can be used as input to SUPG, in which sampling is

used to achieve statistical guarantees on the recall or precision of selected records [95]. These scores can also be used directly to answer the query (i.e., return the records with value above some threshold, either ad-hoc or computed over some validation set), as other systems do [11, 94, 124].

Limit queries. Consider a query that selects 10 frames containing at least 5 cars [93]. Such queries are often used to manually study rare events. In this case, the scoring function and query-specific proxy scores would be the same as for aggregation. For limit queries, we generally recommend using $k = 1$, since this query is typically focused on ranking rare events. The query processing algorithm will examine frames with the target labeler as ordered by the query-specific proxy scores. The algorithm will terminate once the requested number of frames is found.

Custom Proxy Scores

TASTI has built-in functionality to compute and propagate scores for selection, aggregation and limit queries using the methods described in the previous section. In addition, developers may specify custom scores to extend TASTI to supporting other queries.

The API for specifying scoring functions is as follows. Denote the type of the output of the target labeler as `TargetLabelerOutput` (e.g., a list of bounding boxes) and the type of the score as `ScoreType` (e.g., a float). Using Python typing, the developer would implement:

```
def Score(target_output: TargetLabelerOutput) -> ScoreType
```

These functions can be implemented in few lines of code. We show the pseudocode for the example above:

```
def CountCarScore(boxes: Sequence[Boxes]) -> int:
    return len([box for box in boxes if box.object_type == 'car'])
```

Other queries, e.g., over object positions, can be implemented similarly with few lines of code.

Score Propagation

Given the query-specific scoring functions, TASTI will execute the scoring functions on the cluster representatives (as the target labeler outputs are available for these data records). In order to execute downstream query processing, TASTI must also materialize approximate scores for the remainder of the data records.

To produce these query-specific proxy scores, TASTI will propagate scores from the cluster representatives to the unannotated records. The score for each data record will be the inverse distance-weighted mean of the nearest k cluster representatives for numeric scores. For categorical scores, TASTI will take the distance-weighted majority vote. Since the distances to cluster representatives are cached, this process is computationally efficient. A developer may also implement a custom method of propagating scores. We show an example of such a method in Section 4.1.5 for limit queries.

4.1.4 Theoretical Analysis

We present a statistical performance analysis of our methods to better understand resulting query quality. Intuitively, if the original data records have a metric structure and the triplet loss recovers this structure, we expect downstream queries to behave well. Specifically, we provide guarantees on query quality (typically accuracy) when using TASTI directly. We show that accuracy for a natural class of “smooth” queries is directly connected to the triplet loss and the density of clustering. While the assumptions in our analysis may not hold in practice, we conduct our analysis to provide statistically grounded intuition for why TASTI can outperform baseline methods. We validate our intuition with extensive experiments (Section 4.1.5).

We formalize this intuition by analyzing how downstream queries behave under the triplet loss. We specifically analyze the case where $k = 1$, i.e., using a single cluster representative in query processing.

Notation and Preliminaries

Notation. We define the set of data records as $\mathcal{D} := \{x_1, \dots, x_N\}$, the scoring function $f(x_i) : \mathcal{D} \rightarrow \mathbb{R}$, and the embedding function $\phi(x_i) : \mathcal{D} \rightarrow \mathbb{R}^d$. Denote the cluster representatives as $R := \{x_r : r \in \mathcal{R}\} \subset \mathcal{D}$ for some set $\mathcal{R} \subset \{1, \dots, N\}$. Given this set, we denote the representative mapping function as $c(x_i) : \mathcal{D} \rightarrow R$, which maps a data record to the nearest cluster representative, and the query-specific scores as $\hat{f}(x) := f(c(x))$.

Suppose there is a query-specific loss function $\ell_Q(x_i, y_i) : \mathcal{D} \times \mathbb{R} \rightarrow \mathbb{R}$ where $y_i \in \mathbb{R}$ is the predicted label. ℓ_Q will be used to evaluate the quality of f and \hat{f} as $\ell_Q(x, f(x))$ and $\ell_Q(x, \hat{f}(x))$.

We define the per-example triplet loss as

$$\ell_T(x_a, x_p, x_n; \phi, m) := \max(0, m + |\phi(x_a) - \phi(x_p)| - |\phi(x_a) - \phi(x_n)|)$$

where we omit ϕ and m where clear. Define the ball of radius M as $B_M(x) = \{x' : d(x, x') < M\}$ and its complement \bar{B}_M . For random variables $x_a \sim \mathcal{D}$, $x_p \sim B_M(x_a)$, and $x_n \sim \bar{B}_M(x_a)$ drawn uniformly from the sets, we define the population triplet loss as

$$L(\phi; M, m) := \mathbb{E}_{x_a, x_p, x_n}[\ell_T(x_a, x_p, x_n; \phi, m)] \quad (4.1)$$

for some margin $m > 0$.

Assumptions and properties. We make the following assumptions. We first assume that there is a metric $d(x_i, x_j)$ on \mathcal{D} and that \mathcal{D} is compact with metric d . We further assume that $\ell_Q(x, y)$ is Lipschitz in x and y with constant $K_Q/2$, in both arguments.

For both of our proofs, we assume the triplet loss is low and the cluster representatives are dense enough under ϕ . Low triplet loss controls the quality of the embeddings with respect to the original metric d . The density of the cluster representatives controls how close the unannotated records are from the cluster representatives in the original space.

Example. Consider the video setting described in Section 4.1.1. \mathcal{D} is the set of frames, ϕ is the trained embedding DNN, and we use the metric induced by closeness function also described in Section 4.1.1. Consider the two queries: aggregation queries for the number of cars and selecting frames of cars. For the aggregation query, f maps frames to the number of cars. For the selection query, f maps frames with cars to 1 and frames without cars to 0.

Theorem Statements

We defer all proofs to Kang et al. [97].

Zero loss case. To theoretically analyze our index and query processing algorithms, we first consider the case where the embedding achieves zero triplet loss (we generalize to non-zero loss below). We show the following positive result: using the query-specific proxy scores in this setting will achieve bounded loss. In fact, for ℓ_Q that are identically 0 (e.g., for the example above), TASTI will achieve *exact* results.

We now state the main theorem for the zero-loss case.

Theorem 3 (Zero loss). Let ϕ be an embedding that achieves $L(\phi; M, m) = 0$ and c be such that $\max_{x \in \mathcal{D}} |\phi(x) - \phi(c(x))| < m$. Then, the query procedure will suffer an expected loss gap of at most

$$\mathbb{E}[\ell_Q(x, \hat{f}(x))] \leq \mathbb{E}[\ell_Q(x, f(x))] + M \cdot K_Q. \quad (4.2)$$

Generalization to Non-zero Loss. We generalize our analysis to the non-zero loss case below. We show that the loss in queries is bounded by the triplet loss and several other natural quantities.

Theorem 4 (Non-zero loss). Consider an embedding ϕ that achieves $L(\phi; M, m) = \alpha$ and a clustering c such that $\max_{x \in \mathcal{D}} |\phi(x) - \phi(c(x))| < m$. Assume that the query loss ℓ_Q is upper bounded by C . Then, query procedure will suffer an expected loss gap of at most

$$\mathbb{E}[\ell_Q(x, \hat{f}(x))] \leq \mathbb{E}[\ell_Q(x, f(x))] + M \cdot K_Q + \frac{C \sup_x |\bar{B}_M(x)|}{m} \alpha. \quad (4.3)$$

Discussion

We have shown that many classes of queries will have bounded loss (i.e., discrepancy from exact answers). However, we note that our analysis has several limitations. First, TASTI uses the nearest $k = 5$ cluster representatives to generate the query-specific proxy scores by default, not $k = 1$ as used in our analysis. Second, the triplet loss may be large in practice. Third, not all queries admit Lipschitz losses. Nonetheless, we believe our analysis provides intuition for why TASTI outperforms even recent state-of-the-art. We defer a more detailed analysis to future work.

4.1.5 Evaluation

We evaluated TASTI on five real world datasets with three query types. We demonstrate that TASTI’s index construction is cheaper than recent state-of-the-art executed end-to-end and that TASTI’s proxy scores outperforms per-query proxies on all settings we consider. We defer extended experiments to Kang et al. [97]. Our code is available at <https://github.com/stanford-futuredata/tasti>.

Experimental Setup

Datasets, target labelers, and triplet loss. We considered three video datasets, a text dataset, and a speech dataset. We used the `night-street`, `taipei`, and `amsterdam` videos [93]. The `night-street` dataset is widely used in video analytics evaluations [30, 93, 94, 181]. The `taipei` dataset has two object classes (car and bus) and we use the *same* set of embeddings for both. We used Mask R-CNN as the target labeler and ResNet-18 as our embedding DNN. The closeness function separates frames with objects that are far apart and frames with different numbers of objects (when also considering object types).

For the text dataset, we used a semantic parsing dataset [188]. The dataset consists of pairs of natural language questions and corresponding SQL statements. We assumed the SQL statements are not known at query time and must be annotated by crowd workers (i.e., that crowd workers are the target labeler). We used BERT [48] for the embedding DNN. We considered queries over SQL operators and number of predicates. The closeness function separates questions over different SQL operators and number of predicates.

For the speech dataset, we used the Common Voice dataset [12]. The dataset consists of short speech snippets. We assumed that the attributes of speaker gender and age are not known at query time and must be annotated by crowd workers. We used an audio ResNet-22 [108] for the embedding DNN. The closeness function separates records by gender and discretized age bucket.

Queries and metrics. We evaluated TASTI and per-query proxies on three classes of queries using three recently proposed algorithms: aggregation, selection, and limit queries.

Our primary cost metric across all queries is the number of target labeler invocations and also report end-to-end costs for certain experiments. We use target labeler invocations as the primary metric for several reasons. First, in many cases, the target labeler is actually a human labeler, particularly when used in social or life sciences [96]. Second, the target labelers we evaluate are thousands to hundreds of times more expensive than query processing costs and proxy models [99], and thus make up the majority of query costs. In addition, this strictly benefits systems that use per-query proxies which must be executed at query time: TASTI does not train a model per query.

Aggregation. For aggregation queries, we queried for an approximate statistic of the target labeler executed on the unstructured data records. We computed the average number of objects per frame for the video datasets, the average number of predicates per query for the

WikiSQL dataset, and the fraction of male speakers in the Common Voice dataset.

For all settings, we used the EBS sampling as used by the BLAZEIT system [93], which provides guarantees on error. EBS sampling uses the proxy scores to guide target labeler sampling. Better proxy scores will result in fewer target labeler invocations. As such, we measured the number of target labeler invocations (lower is better).

We additionally compare TASTI to approximate aggregation without statistical guarantees, which uses the proxy scores to answer queries directly (as used by BLAZEIT).

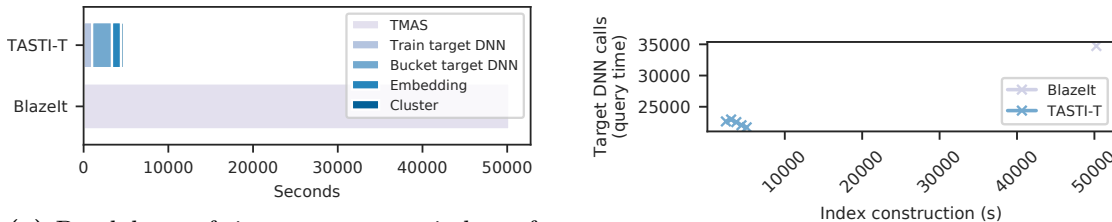
Selection. For selection queries, we executed approximate selection queries with recall targets (SUPG queries [95]). We selected for frames with objects for video datasets, natural language questions that are parsed into selection SQL statements for the WikiSQL dataset, and male speakers in the Common Voice dataset.

Given a target labeler budget, these queries return a set of records matching a predicate with a given recall target with a given confidence level (e.g., “return 90% of instances of cars with 95% probability of success”): these queries are useful in scientific applications or mission-critical settings [95]. In contrast to queries that do not provide statistical guarantees, SUPG guarantees the recall target with high probability. Since recall SUPG queries fix the number of target labeler invocations, we measured the false positive rate (lower is better).

We additionally compare TASTI to approximate selection without statistical guarantees, which uses the proxy scores to answer queries directly. We slightly modify the query processing algorithms of NOSCOPe, Tahoma, and probabilistic predicates to directly use proxy scores and use the accuracy metric of F1 score.

Limit. For limit queries, we used the ranking algorithm proposed by Kang et al. [93]. This ranking algorithm examines data records that are likely to match the predicate of interest in descending order by the proxy score. Proxy scores that have high recall for given number of records will perform better. As such, we measured the number of target labeler invocations (lower is better).

Methods evaluated. We used the query processing methods above and use the per-query proxies as used in Kang et al. [93] (aggregation and limit queries) and Kang et al. [95] (selection queries). We use the exact proxy models for the video datasets (a “tiny ResNet”), logistic regression over FastText embeddings [23] for the WikiSQL dataset, and a smaller CNN (CNN-10) [108] for the Common Voice dataset. FastText embeddings are less expensive than BERT embeddings.



(a) Breakdown of time to construct indexes for TASTI and for BLAZEIT on the `night-street` dataset. The BLAZEIT index is the “target-model annotated set” (TMAS) [93]. Similar results hold for other datasets.

(b) Index construction time vs performance of TASTI and BLAZEIT for aggregation queries on the `night-street` dataset. Similar results hold for other datasets.

We refer to TASTI when using a pre-trained DNN as the embedding DNN as “TASTI-PT” (pre-trained) and TASTI when using a triplet-loss trained embedding DNN as “TASTI-T” (trained). We demonstrate that TASTI-T generally outperforms TASTI-PT.

Hardware and timing. We evaluated TASTI on a private server with a single NVIDIA V100 GPU, 2 Intel Xeon Gold 6132 CPUs (56 hyperthreads), and 504GB of memory. In contrast to prior work, we timed end-to-end query processing times for TASTI, *including* the video loading and embedding DNN execution times, which is excluded in prior work [93].

Due to the large cost of executing the target labeler, we simulated its execution by caching target labeler results and computing the average execution time for the target labeler. For baselines, we only timed the target labeler computation and exclude the computational cost of proxy models, which strictly improves the baselines. We excluded the cost of query processing [93, 95] as it is negligible in all cases. Namely, the query processing is over orders of magnitude less expensive than target labeler invocation for all queries we consider.

Index Construction Performance

To understand the index construction performance, we measured the wall clock time to construct TASTI indexes. We compared to BLAZEIT, which constructs indexes by executing the target labeler on a subset of the data (referred to as the “TMAS” [93]). For BLAZEIT, we only considered the cost of constructing the TMAS. For TASTI, we measured the full index construction time, including the embedding DNN training and distance computation times. We computed the construction times on the `night-street` dataset; similar results hold for other datasets.

We show the breakdown of index construction time for TASTI and BLAZEIT in Figure 4.2a using the parameters in Section 4.1.5. TASTI requires far fewer target labeler invocations for index construction, so is substantially faster than BLAZEIT.

We additionally show the index construction time vs performance for BLAZEIT and a range of parameters for TASTI (Figure 4.2b). TASTI can outperform or match BLAZEIT performance with up to $10\times$ less expensive index construction times.

End-to-end Performance

We show that TASTI outperforms recent state-of-the-art per-query proxy methods for approximate aggregation, selection with guarantees, and limit queries. For all video datasets in this section, we used 3,000 training records, 7,000 cluster representatives, and an embedding size of 128. To show the generality of TASTI, we used a single set of embeddings/distances for both `taipei` classes. For the WikiSQL and Common Voice datasets, we used 500 training examples and 500 cluster representatives. We measured the query processing costs or query accuracy in this section.

Approximate aggregation. For approximate aggregation queries, we compared TASTI to using no proxy (random sampling) and an ad-hoc trained proxy model. We used the exact experimental setup as BLAZEIT [93] for video datasets, which targeted an error of 0.01 and a success probability of 95%. We aggregated over the average number of objects per frame for all video datasets (cars or buses), the number of clauses per statement in the WikiSQL dataset, and the fraction of male speakers in the Common Voice dataset.

As shown in Figure 4.3, TASTI outperforms for aggregation queries on all datasets. In particular, TASTI outperform state-of-the-art per-query proxies for aggregation queries (BLAZEIT) by up to $2\times$ with less expensive index construction costs. Further, TASTI outperforms no proxy by up to $3\times$.

TASTI’s improved performance comes from better query-specific proxy scores (ρ^2 of 0.91 vs 0.55). As the correlation of the proxy scores with the target labeler increases, the control variates variance decreases. Reduced variance results in fewer samples, as the EBS stopping algorithm is adaptive with the variance.

Selection. For selection queries with statistical guarantees (SUPG queries), we compared TASTI to using an ad-hoc trained proxy model (standard random sampling is not appropriate for SUPG queries). We used the exact same experimental setup as in SUPG [95]

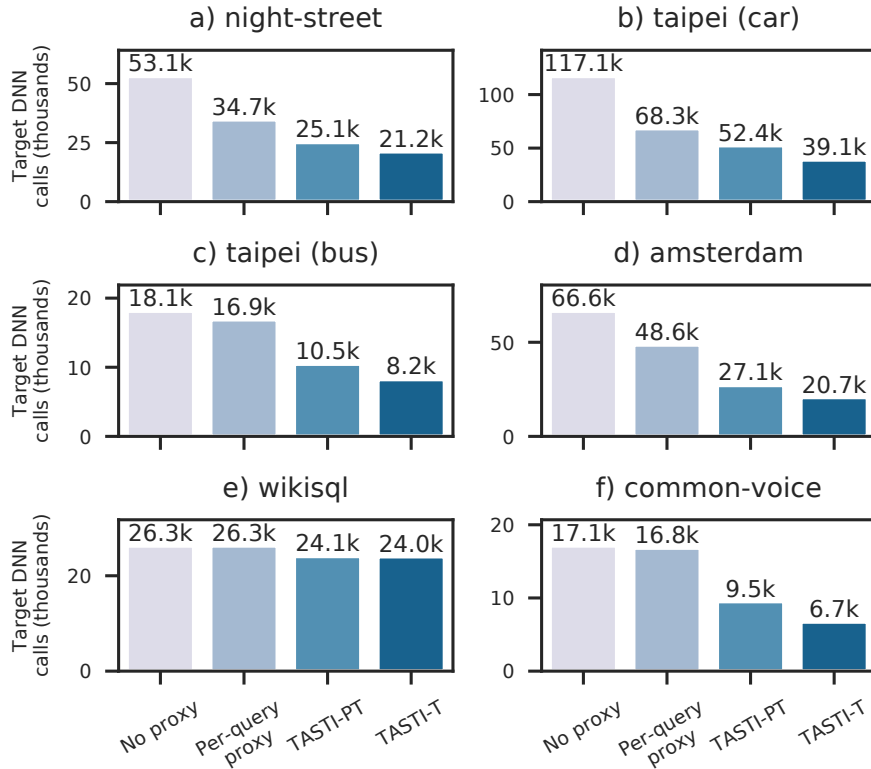


Figure 4.3: Number of target labeler invocations for baselines and TASTI for approximate aggregation queries (lower is better). TASTI outperforms baselines in all cases, including prior, per-query proxy state-of-the-art by up to $2\times$.

for the video datasets. For all queries, we used a recall target of 90% with a confidence of 95%, as used in [95]. We search for cars or buses in the video datasets, star operators for WikiSQL, and male speakers in the Common Voice dataset.

As shown in Figure 4.4, TASTI outperforms on all datasets. In particular, TASTI can improve the false positive rate by almost $21\times$ over recent state-of-the-art. We further show that the triplet training improves performance. As with aggregation queries, TASTI’s improved performance comes from better query-specific proxy scores (ρ^2 of 0.90 vs 0.79).

Limit queries. For limit queries, we used the ranking algorithm proposed by BLAZEIT [93]. We use the exact same experimental setup as BLAZEIT for the video datasets (including the query configurations, e.g., number of objects, etc.). For limit queries, we use a custom scoring function which is the regular scoring function with $k = 1$ and ties broken by distance to the cluster representatives.

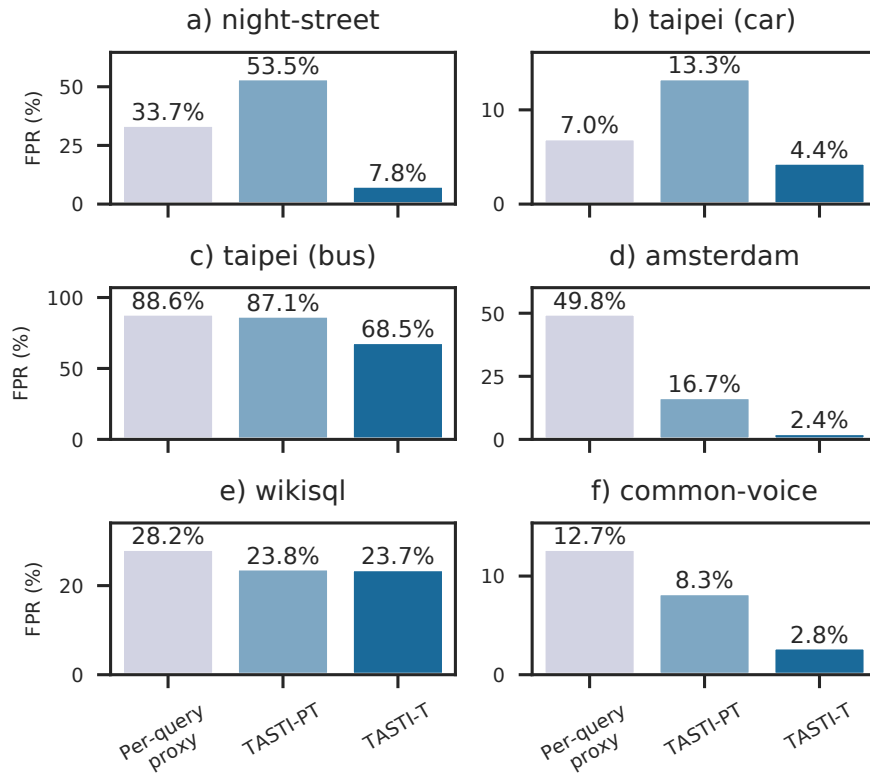


Figure 4.4: False positive rate for recall-target SUPG queries (lower is better). We show the performance of baselines and TASTI. As shown, TASTI outperforms baselines in all cases.

Figure 4.5 shows TASTI outperforms on all datasets. TASTI can improve performance by up to $24\times$ compared to recent state-of-the-art. As we demonstrate, TASTI’s FPF mining and FPF clustering are critical for performance when searching for rare events. The FPF algorithm naturally produces clusters that are far apart, which is beneficial when searching for rare events.

4.1.6 Discussion

To reduce the cost of queries using expensive target labelers, we introduce a method of constructing indexes for unstructured data. TASTI relies on the key property that queries only require access to target labelers outputs, which are often highly redundant. TASTI uses an embedding DNN and target labeler annotated cluster representatives as its index, which allows for more accurate and generalizable proxy scores across a range of query types.

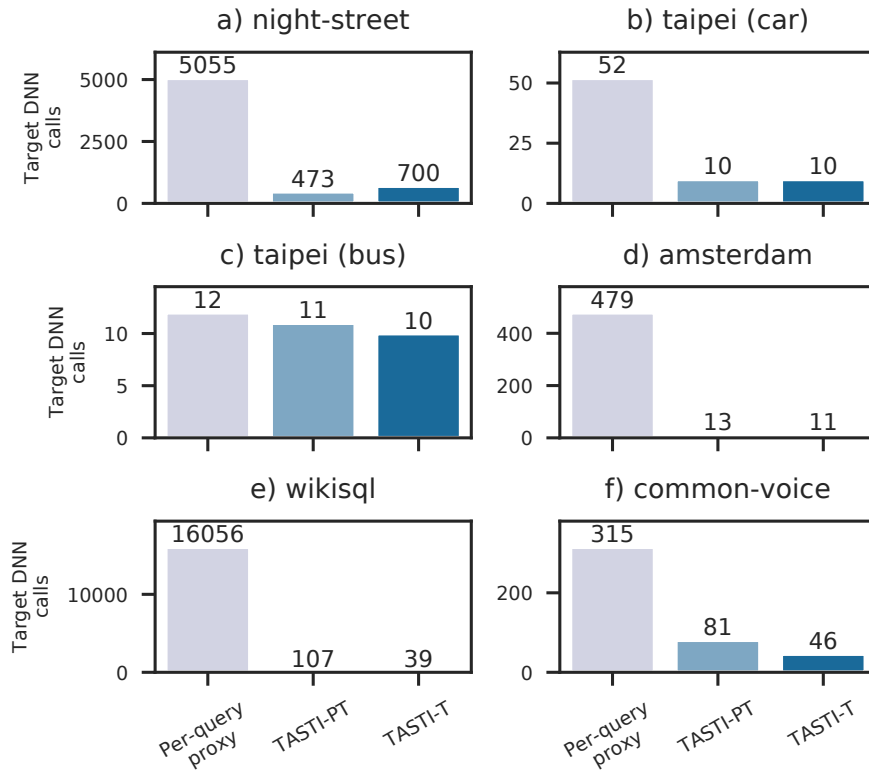


Figure 4.5: Number of target labeler invocations for baselines and TASTI for limit queries (lower is better). TASTI outperforms baselines in all cases, including prior state-of-the-art by up to $34\times$.

We theoretically analyze TASTI to understand its statistical properties, in particular how index quality relates to query accuracy. We show TASTI indexes can be constructed up to $10\times$ more efficiently than recent work. We further show that they can be used to answer queries up to $24\times$ more efficiently than recent state-of-the-art.

4.2 SMOL: Hardware Efficient Proxy Generation

Deep neural networks (NNs) now power a range of visual analytics tasks and systems [11, 83, 93, 94] due to their high accuracy, but state-of-the-art DNNs can be computationally expensive. For example, accurate object detection methods can execute as slow as 3-5 frames per second (fps) [73, 165].

To execute visual analytics queries efficiently, systems builders have developed optimizations to trade off accuracy and throughput [11, 83, 93, 94, 124]: more accurate DNNs are

more computationally expensive [74, 164, 165]. Many of these systems (e.g., NOSCOPE, BLAZEIT, TAHOMA, and probabilistic predicates) accelerate visual analytics queries by using proxy or *specialized NNs*, which approximate larger target DNNs. These specialized NNs can be up to 5 orders of magnitude cheaper to execute than their target DNNs and are used to filter inputs so the target DNNs will be executed fewer times [11, 83, 93, 94, 124].

This prior work focuses solely on reducing DNN execution time. These systems were built before recent DNN accelerators were introduced and were thus benchmarked on older accelerators. In this context, these systems correctly assume that DNN execution time is the overwhelming bottleneck. For example, TAHOMA benchmarks on the NVIDIA K80 GPU, which executes ResNet-50 (a historically expensive DNN [2, 40, 41]) at 159 images/second.

However, as accelerators and compilers have advanced, these systems ignore a key bottleneck in *end-to-end* DNN inference: preprocessing, or the process of decoding, transforming, and transferring image data to accelerators. In the first measurement study of its kind, we show that preprocessing costs often *dominate end-to-end DNN inference* when using advances in hardware accelerators and compilers. For example, the historically expensive ResNet-50 [2, 41] has improved in throughput by $28\times$ on the inference-optimized NVIDIA T4 GPU. As a result, ResNet-50 is now $9\times$ higher throughput than CPU-based image preprocessing, making *preprocessing the bottleneck*, on the inference-optimized `g4dn.xlarge` Amazon Web Services (AWS) instance, which has a NVIDIA T4. This boost in efficiency translates to both power and dollar costs: preprocessing requires approximately $2.3\times$ as much power and costs $11\times$ as much as DNN execution. Similar results hold for Google Cloud’s T4 inference optimized instances. These imbalances become only higher with smaller specialized NNs that recent visual analytics systems use.

In light of these observations, we examine opportunities for more principled *joint optimization* of preprocessing and DNN execution, especially for preprocessing-bound, high-throughput batch analytics workloads. We leverage two insights: a) the accuracy and throughput of a DNN is closely coupled with its input format and b) preprocessing operations can be placed on both CPUs and accelerators. Thus, rather than treating the input format as fixed, we consider methods of using inputs as a key step in DNN architecture search and training.

This yields two novel opportunities for accelerating inference: a) cost-based methods that leverage low-resolution visual data for higher accuracy or improved throughput and b) input- and hardware-aware methods of placing preprocessing operations on the CPU or

accelerator and correctly pipelining computation.

A critical component to leverage these opportunities is a cost model to select query plans. We correct the erroneous assumption in prior work that DNN execution dominates *end-to-end* DNN inference. We instead propose a cost model that is preprocessing aware and validate that our cost model is more accurate than prior cost models. While our preprocessing aware cost model is simple, it enables downstream optimizations, described below.

First, we propose methods of using natively present, low resolution visual data for more efficient, input-aware accuracy/throughput trade offs. Image and video serving sites often have natively present low resolution data, e.g., Instagram has thumbnails [13] and YouTube stores multiple resolutions of the same video. Even when low resolution data is not natively present, we can partially decode visual data (e.g., omitting the deblocking filter in H.264 decoding). As such, we can use natively present data or partial decoding for reduced preprocessing costs. However, naively using this reduced fidelity data can reduce accuracy. To recover accuracy, we propose an augmented DNN training procedure that explicitly uses data augmentation for the target resolution. Furthermore, we show that using larger, more accurate DNNs on low resolution data can result in higher accuracy than smaller DNNs on full resolution data. Enabled by our new preprocessing-aware cost model, we can select input formats and DNN combinations that achieve better accuracy/throughput trade offs.

Second, we decide to place preprocessing operations on the CPU or accelerator to balance the throughput of DNN execution and preprocessing. To enable high-performance pipelined execution, we build an optimized runtime engine for end-to-end visual DNN inference. Our optimized runtime engine makes careful use of pipelined execution, memory management, and high-performance threading to fully utilize available hardware resources.

We implement these optimizations in SMOL, a runtime engine for end-to-end DNN inference that can be integrated into existing visual analytics systems. We use SMOL to implement the query processing methods of two modern visual analytics systems, BLAZEIT [93] and TAHOMA [11], and evaluate SMOL on eight visual datasets, including video and image datasets. We verify our cost modeling choices through benchmarks on the public cloud and show that SMOL can achieve up to $5.9\times$ improved throughput on recent GPU hardware compared to recent work in visual analytics.

In the remainder of this section, we describe trends in accelerator efficiency, SMOL’s overview and optimizations, and our evaluation of SMOL.

4.2.1 Measurement Study of End-to-End DNN Inference

We benchmark DNNs and visual data preprocessing on the public cloud, showing that *preprocessing costs* can now dominate end-to-end DNN inference. We show that these trends arise from dramatically improved new accelerators reducing dollar and power costs of DNN execution, and efficient use of hardware.

We benchmark throughputs on the inference-optimized T4 GPU with a dollar cost-balanced number of vCPU cores on an AWS instance. Our benchmarks show that preprocessing dominates in both dollar cost and power costs. Preprocessing requires $2.2\times$ as much power (158W vs 70W) and costs $11\times$ as much for ResNet-50 (\$2.37 vs \$0.218). These trends are similar for other cloud providers (e.g., Google Cloud Platform’s T4-attachable instances and Microsoft Azure’s newly announce T4 instances) and instance types.

Experimental setup. We benchmarked the popular ResNet-50 model for image classification [74], which has widely been used in benchmarking [2, 41] and has been considered expensive. Specialized NNs are typically much cheaper than ResNet-50.

We benchmarked the time for only DNN execution and the time for preprocessing separately to isolate bottlenecks.

We benchmarked on the publicly available inference-optimized NVIDIA T4 GPU [134]. We used the `g4dn.xlarge` AWS instance which has 4 vCPU cores (hyperthreads): this configuration is cost balanced between vCPUs and the accelerator. This instance type is optimized for DNN inference; similar instances are available on other cloud providers. We used the TensorRT compiler [3] for optimized execution. While we benchmarked on the T4, other contemporary, non-public accelerators report similar or improved results [56, 89].

Effect of software on throughput. We benchmarked ResNet-50 throughput on the inference-optimized T4 GPU using three software systems for DNNs to show how more efficient software affects throughput. We benchmark using Keras [37], PyTorch [139], and TensorRT [3]. We note that Keras was used by TAHOMA and TensorRT is an optimized DNN computational graph compiler.

As shown in Table 4.1, efficient use of accelerators via optimized compilers (TensorRT) can result in up to a $10\times$ improvement in throughput. Importantly, preprocessing becomes the bottleneck with the efficient use of accelerators.

Breakdown of end-to-end DNN inference. DNN inference includes preprocessing. For the standard ResNet-50 configuration, the preprocessing steps are [74, 127]: 1) Decode the

Execution environment	Throughput (im/s)
Keras	243
PyTorch	424
TensorRT	4,513

Table 4.1: Throughput of ResNet-50 on the T4 with three different execution environments. Keras was used in [11]. The efficient use of hardware can result in over a $17\times$ improvement in throughput.

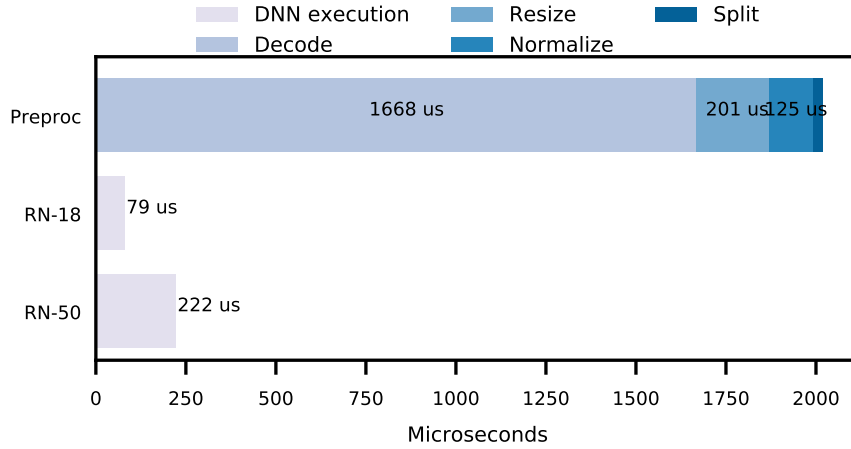


Figure 4.6: Breakdown of end-to-end inference of ResNet-50 and 18 for a batch size of 64 on the inference-optimized AWS `g4dn.xlarge` instance type. The execution of the DNN is $7.1\times$ and $22.9\times$ faster than preprocessing data for ResNet-50 and 18 respectively.

compressed image, e.g., JPEG compressed, 2) Resize the image with an aspect-preserving resize such that the short edge of the image is 256 pixels. Centrally crop the image to 224×224 , 3) Convert the image to float32. Divide the pixel values by 255, subtract a per-channel value, and divide by a per-channel value, and 4) Rearrange the pixel values to channels-first.

To see the breakdown of preprocessing the costs, we implemented these preprocessing steps in hand-optimized C++, ensuring best practices for high performance C++, including reuse of memory. We used `libturbo-jpeg`, a highly optimized library for JPEG decompression, for decoding the JPEG images. We used OpenCV’s optimized image processing libraries for the resize and normalization. For DNN execution, we executed the DNNs with TensorRT and multiple CUDA streams on synthetic images. We run all benchmarks on a standard `g4dn.xlarge` AWS instance and use multithreading to utilize all the cores.

As shown in Figure 4.6, simply decoding the JPEG files achieves lower throughput than

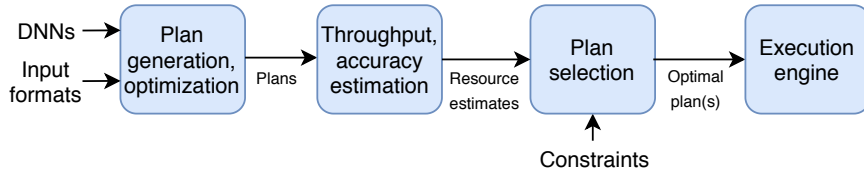


Figure 4.7: System diagram of SMOL. As input, SMOL takes a set of DNNs, visual input formats, and optional constraints. As output, SMOL returns an optimal set of plans or plan, depending on the constraints. SMOL will generate plans, estimate the resources for each plan, and select the Pareto optimal set of plans.

the throughput of ResNet-50 execution. All together, preprocessing achieves $7.1\times$ lower throughput than ResNet-50 execution. These overheads increase to up to $22.9\times$ for ResNet-18. As discussed, preprocessing dominates in terms of power and dollar costs as well.

Similar results hold for other networks, such as the MobileNet-SSD [82, 120] used by MLPerf Inference [148]. This DNN executes at 7,431 im/s, compared to a preprocessing throughput of 397 im/s on the MS-COCO dataset.

Discussion. Several state-of-the-art DNNs execute far slower than the DNNs benchmarked in this section, e.g., a large Mask R-CNN may execute at 3-5 fps. However, many systems use specialized NNs to reduce invocations of these large DNNs. For example, the BLAZEIT system uses a specialized NN to approximate the larger DNN, which reduces the number of large DNN invocations [93]. As these specialized NNs are small (potentially much smaller than even ResNet-50), we believe our benchmarks are of wide applicability to DNN-based visual analytics.

4.2.2 SMOL Overview

To reduce the imbalance between preprocessing and DNN execution, we develop a novel system, SMOL. SMOL’s goal is to execute *end-to-end* batch visual analytics queries. Unlike prior work, SMOL aims to optimize end-to-end query time, including the computational cost of preprocessing in addition to the computational cost of DNN execution.

To execute these visual analytics queries, SMOL uses a cost-based model to generate query plans that span preprocessing and DNN execution. SMOL executes these plans in its optimized end-to-end inference engine. For a given query system (e.g., TAHOMA or BLAZEIT), SMOL’s cost model must be integrated into the system.

We show a schematic of SMOL’s architecture in Figure 4.7.

Model	Throughput	Preproc. bound?
MobileNet-V1	16,885	Yes
VGG19	1,889	Yes
Inception V4	1,339	Yes
ResNet-50	4,513	Yes
ResNeXt-101	1,724	Yes
SSD MobileNet-V1 (300)	7,609	Yes
SSD ResNet (1200)	137	No
Mask R-CNN (1200)	14	No

Table 4.2: Throughput of various models on the T4 GPU (classification models on the top and detection models on the bottom) [178]. As shown, all but the largest, state-of-the-art detection models are preprocessing bound.

System Overview

Deployment setting. In this work, we focus on high throughput batch settings, as recent work does [11, 93, 95, 181]. SMOL’s goal is to achieve the highest throughput on the available hardware resources. For example, a visual analytics engine might ingest images or videos daily and run a batch analytics job each night. SMOL is most helpful for preprocessing-bound workloads (Table 4.2). As we describe, SMOL accepts models exported from training frameworks (e.g., PyTorch, TensorFlow, or Keras) and optimizes its inference. As such, it is designed to be used at inference time, not with training frameworks.

Nonetheless, several of our techniques, particularly in jointly optimizing preprocessing and inference, also apply to the low-latency or latency-constrained throughput settings.

In high throughput batch settings, visual data is almost always stored in compressed formats that require preprocessing. Uncompressed visual data is large: a single hour of 720p video is almost 900GB of data whereas compressed video can be as small as 0.5GB per hour. Similarly, JPEG images can be up to $10\times$ smaller than uncompressed still images.

SMOL inference. As inputs, SMOL will take a set of trained DNNs and a set of natively available visual data formats (e.g., full resolution JPEG images, thumbnail JPEGs). We denote the set of DNNs as \mathcal{D} and the set of visual data formats as \mathcal{F} . SMOL further takes a set of calibration images (i.e., a validation set) to estimate accuracy.

Given these inputs, SMOL will estimate costs to select a plan (concretely, a DNN and an input format). SMOL will then optimize this plan and execute it.

SMOL optionally takes a throughput or accuracy constraint at inference time. If a constraint is specified, SMOL will select an optimized execution plan that respects these constraints. Otherwise, SMOL will execute the highest throughput plan. SMOL can be integrated with other systems by returning a Pareto optimal set of plans (in accuracy and throughput). The calling system will then select a plan that SMOL will execute.

SMOL training. While the user can provide the set of trained DNNs, SMOL can optionally train specialized NNs as well. Given a set of DNN architectures (e.g., ResNets) and the natively available formats, SMOL will choose to train some or all of the DNNs. Given the initial set of models on full resolution data, SMOL will fine-tune the networks on the cross product of \mathcal{D} and resolutions (SMOL will use the same model for different formats of the same resolution). As SMOL fine-tunes, this process adds at most a 30% overhead in training in the settings we consider. SMOL can also train these network at execution time [93].

Components. SMOL implements the training phase as other systems do [11, 93, 94]. As training specialized NNs has been studied in depth in prior work, we defer discussion to this prior work. SMOL differs from these systems only in its low-resolution augmented training (discussed below).

At inference time, SMOL contains three major components: 1) a plan generator, 2) a cost estimator, and 3) an execution engine. We show these components in Figure 4.7.

SMOL first generates query plans from \mathcal{D} and \mathcal{F} by taking $\mathcal{D} \times \mathcal{F}$. For each plan, SMOL will estimate the relative costs of preprocessing and DNN execution and decide where to place preprocessing operations (i.e., on the CPU or accelerator) for highest throughput. Given these optimized plans, SMOL will estimate the accuracy and throughput of these plans using its cost model. This process is cheap compared to training, so SMOL exhaustively benchmarks the Pareto frontier of $\mathcal{D} \times \mathcal{F}$. SMOL uses a preprocessing-aware cost model, in contrast to prior work that ignores these costs. Finally, SMOL will return the best query plan if a constraint is specified or the Pareto optimal set of query plans if not.

Optimizations. To efficiently execute queries, SMOL has several optimizations for improved accuracy/throughput trade offs and an efficient DNN execution engine.

Briefly, SMOL achieves improved accuracy and throughput trade offs by considering an expanded set of DNNs and leveraging natively present low-resolution data (Section 4.2.4). In contrast, prior work considers only one input format. From the selected DNN and input format, SMOL will efficiently execute such plans by placing preprocessing operations on CPUs

or accelerators in a hardware- and input-aware manner, efficiently pipelining computation stages, and optimizing common preprocessing operations (Section 4.2.5). We describe these optimizations in detail below.

Examples

Classification example. SMOL can be incorporated into prior work that uses specialization for classification queries [11, 94, 124]. These queries are often binary classification queries, e.g., the presence or absence of a car in a video. We describe TAHOMA in this example, but note that other systems are similar in spirit.

TAHOMA uses a fixed target model and considers a fixed input format, namely the provided input format of full-resolution JPEG images. TAHOMA considers 24 specialized NN, each of which are cascaded with the target DNN. Thus, $|\mathcal{F}| = 1$ and $|\mathcal{D}| = 24$. TAHOMA aims to return the configuration with the highest throughput for a given accuracy. TAHOMA estimates the throughput of $D_i \in \mathcal{D}$ by *adding* preprocessing costs, which we show leads to inaccurate throughput estimates. We further note that TAHOMA considers downsampling full resolution images for improved DNN execution, but not for reduced preprocessing costs.

In contrast, SMOL can use natively present thumbnail images, which would expand \mathcal{F} . Decoding these thumbnail images is significantly higher throughput.

Aggregation example. SMOL can be incorporated into prior work that uses specialized NNs for aggregation queries over visual data, e.g., the number of cars in a video. The recent BLAZEIT system uses specialized NNs as a control variate to reduce the variance in sampling [93]. As the variance is reduced, this procedure results in fewer target model invocations compared to standard random sampling. BLAZEIT trains a single specialized NN ($|\mathcal{D}| = 1$) and uses a fixed input format ($|\mathcal{F}| = 1$).

In contrast, SMOL can use an expanded set of videos which are encoded at different resolutions. Namely, SMOL considers $|\mathcal{F}| > 1$. These other formats are natively present in many serving applications, e.g., for thumbnail or reduced bandwidth purposes.

4.2.3 Cost Modeling for Visual Analytics

When deploying DNN-based visual analytics systems, application developers have different resource constraints. As such, these systems often expose a way of trading off between accuracy and throughput. Higher accuracy DNNs require more computation: we demonstrate

ResNet	Throughput	Accuracy
ResNet-18	12,592	68.2%
ResNet-34	6,860	71.9%
ResNet-50	4,513	74.34%

Table 4.3: Throughput and top-one accuracy for ResNets of different depths. As shown, there is a trade off between accuracy and throughput (i.e., computation).

Config.	Preproc (im/s)	DNN execution (im/s)	Pipelined throughput (im/s)	SMOL estimate (% error)	BLAZEIT estimate (% error)	TAHOMA estimate (% error)
Balanced	4001	4999	4056	1.4% , 4001	23.2%, 4999	44.8%, 2222
Preproc-bound	534	4999	557	4.1% , 534	797.5%, 4999	9.3%, 482
DNN-bound	5876	1844	1720	7.2% , 1844	7.2% , 1884	22.7%, 1403

Table 4.4: Measurements of preprocessing, DNN execution, and pipelined end-to-end DNN inference for three configurations of DNNs and input formats: balanced, preprocessing-bound, and DNN-execution bound. As shown, SMOL matches or ties the most accurate estimate for all conditions.

this property on the popular ImageNet dataset [47] with standard ResNets in Table 4.3. Prior work has designed high throughput specialized DNNs for filtering [11, 93, 94]. We do not focus on the design of DNNs in this work and instead use standard DNNs.

One popular method for DNN selection is to use a cost model [11, 94]. We describe cost modeling for DNNs and how prior work estimated the throughput of DNN execution. Critically, these prior cost models ignore preprocessing costs or ignore that preprocessing can be pipelined with DNN execution. We show that ignoring these factors can lead to inaccurate throughput estimations (Table 4.4). We then describe how to make cost models preprocessing-aware.

Cost models. Given resource constraints and metrics to optimize, a system must choose which DNNs to deploy. For example, one popular resource constraint is a minimum throughput and one popular metric is accuracy. As such, we focus on throughput-constrained accuracy and accuracy-constrained throughput.

Specifically, denote the possible set of system configurations as C_1, \dots, C_n . Denote the resource consumption estimate of each configuration as $R(C)$ and the resource constraint as R_{\max} . Denote the metric to optimize as $M(C)$.

In its full generality, the optimization problem is

$$\begin{aligned} \max_i M(C_i) \\ \text{s.t. } R(C_i) \leq R_{\max}. \end{aligned} \tag{4.4}$$

In this framework, both accuracy and throughput can either be constraints or metrics. For

example, for throughput-constrained accuracy, $R(C_i)$ would be an estimate of the throughput of C_i and $M(C_i)$ would be an estimate of the accuracy of C_i . Similarly, for accuracy-constrained throughput, $R(C_i)$ would be an estimate of the accuracy and $M(C_i)$ would be an estimate of the throughput.

As an example, TAHOMA generates $C_i = [D_{i,1}, \dots, D_{i,k}]$ to be a sequence of k models, $D_{i,j}$, that are executed in sequence. The resource $R(C_i) = A(C_i)$ is the accuracy of configuration C_i and the metric $M(C_i) = T(C_i)$ is the throughput of configuration C_i .

Prior work has focused on expanding the set of C_i or evaluating $R(C_i)$ and $M(C_i)$ efficiently [11, 30, 94, 124]. A common technique is to use a smaller model (e.g., a specialized NN) to filter data before executing a larger, target DNN in a *cascade*. For example, when detecting cars in a video, NOSCOPE will train an efficient model to filter out frames without cars [94]. Cascades can significantly expand the feasible set of configurations.

For cost models to be effective, the accuracy and throughput measurements must be accurate. We discuss throughput estimation below. Accuracy can be estimated using best practices from statistics and machine learning. A popular method is to use a held-out validation set to estimate the accuracy [22]. Under the assumption that the test set is from the same distribution as the validation set, this procedure will give an estimate of the accuracy on the test set.

Throughput estimation. A critical component of cost model for DNNs is the throughput estimation of a system configuration C_i ; recall that C_i is represented as a sequence of one or more DNNs, $D_{i,j}$. Given a specific DNN $D_{i,j}$, estimating its throughput simply corresponds to executing the computation graph on the accelerator and measuring its throughput. As DNN computation graphs are typically fixed, this process is efficient and accurate.

Estimation ignoring preprocessing. Prior work [93, 94, 124] has used the throughput of $D_{i,j}$ to estimate the throughput of end-to-end DNN inference. Specifically, BLAZEIT and NOSCOPE estimates the throughput, $\hat{T}(C_i)$ as

$$\hat{T}(C_i) \approx \frac{1}{\sum_{j=1}^k \frac{1}{\alpha_j^{-1} T_{\text{exec}}(D_{i,j})}} \quad (4.5)$$

where α_j is the pass-through rate of DNN $D_{i,j}$ and $T_{\text{exec}}(D_{i,j})$ is the throughput of executing $D_{i,j}$. $T_{\text{exec}}(D_{i,j})$ can be directly measured using synthetic data and α_j can be estimated with a validation set. This approximation holds when the cost of preprocessing is small compared to the cost of executing the DNNs.

However, this cost model ignores preprocessing costs. As a result, it is inaccurate when

preprocessing costs dominate DNN execution costs or when preprocessing costs are approximately balanced with DNN execution costs (Table 4.4).

Estimation ignoring pipelining. Other systems (e.g., TAHOMA) [11] estimate end-to-end DNN inference throughput as

$$\hat{T}(C_i) \approx \frac{1}{\frac{1}{T_{\text{preproc}}(C_i)} + \frac{1}{T_{\text{exec}}(C_i)}}. \quad (4.6)$$

This approximation ignores that preprocessing can be pipelined with DNN execution. As a result, this approximation holds when either preprocessing or DNN execution is the overwhelming bottleneck, but is inaccurate for other conditions, namely when preprocessing costs are approximately balanced with DNN execution costs (Table 4.4).

These throughput approximations (that ignore preprocessing costs and ignore pipelining) ignore two critical factors: 1) that input preprocessing can dominate inference times and 2) that input preprocessing can be pipelined with DNN execution on accelerators. We now describe a more accurate throughput estimation scheme.

Corrected throughput estimation. For high throughput DNN inference on accelerators, the DNN execution and preprocessing of data can be pipelined. As a result, SMOL uses a more accurate throughput estimate for a given configuration:

$$\hat{T}(C_i) \approx \min \left(T_{\text{preproc}}(C_i), \frac{1}{\sum_{j=1}^k \frac{1}{\alpha_j^{-1} T_{\text{exec}}(D_{i,j})}} \right) \quad (4.7)$$

Importantly, preprocessing can dominate end-to-end DNN inference (Section 4.2.1). While there are some overheads in pipelining computation, we empirically verify the min approximation (Section 4.2.6).

If preprocessing costs are fixed, then it becomes optimal to maximize the accuracy of the DNN subject to the preprocessing throughput. Namely, the goal is to pipeline the computation as effectively as possible. We give two examples of how this can change which configuration is chosen.

First, when correctly accounting for preprocessing costs in a throughput-constrained accuracy deployment, it is not useful to select a throughput constraint higher than the throughput of preprocessing. Second, for an accuracy-constrained throughput deployment, the most accurate DNN subject to the preprocessing throughput should be selected.

4.2.4 Input-aware Methods for Accuracy and Throughput Trade Offs

Given the corrected cost model, SMOL’s goal is to maximize the minimum of the preprocessing and DNN execution throughputs. However, if the input format and resolution are fixed, preprocessing throughputs are fixed and can be lower than DNN execution throughputs.

To provide better accuracy and throughput trade-offs, we propose three techniques: 1) expanding the search space of specialized DNNs, 2) using natively present, low resolution visual data, and 3) a DNN training technique to recover accuracy loss from naively using low resolution visual data.

Expanding search space

As described, many systems only consider cheap, specialized NNs. Concretely, BLAZEIT and TAHOMA considers specialized NNs that can execute up to 250,000 images/second, which far exceeds preprocessing throughputs for standard image and video encodings. As DNNs are generally more accurate as they become more expensive, these systems use specialized NNs that are less accurate relative to preprocessing throughput-matched NNs.

In contrast, SMOL considers NNs that have been historically considered expensive. We have found that standard ResNet configurations [74] (18 to 152) strongly outperforms specialized NNs used in prior work. Furthermore, ResNet-18 can execute at 12.6k images/second, which generally exceeds the throughput of preprocessing. Thus, SMOL currently uses these ResNets as the specialized NNs. As hardware advances, other architectures (e.g., ResNeXt [180]) may be appropriate.

Low-resolution data

Overview. Many visual data services store the data at a range of resolutions. Low-resolution visual data is typically stored for previewing purposes or for low-bandwidth situations. For example, Instagram stores 161x161 previews of images [13]. Similarly, YouTube stores several resolutions of the same video for different bandwidth requirements.

Decoding low-resolution visual data is more efficient than decoding full resolution data. SMOL could decode and then upscale the low-resolution visual data for improved preprocessing throughput. However, we show that naively upscaling gives low accuracy results. Instead, SMOL will train DNNs to be aware of low-resolution data, as described below.

Recent work uses lower resolution data to improve NN throughput, *but not to reduce preprocessing costs* [11, 183]. These systems decode full-resolution data and downsamples the data, which does not improve preprocessing throughput.

Selecting DNNs and resolution jointly. Many systems provide accuracy and throughput trade-offs by cascading a specialized NN and a more accurate, target DNN [11, 94, 124]. However, these specialized NNs are often bottlenecked by preprocessing costs.

Instead, SMOL uses low resolution data reduce preprocessing costs, and therefore end-to-end execution costs. However, low resolution visual data discards visual information and can result in lower accuracy in many cases. Nonetheless, SMOL can provide accuracy and throughput trade-offs by carefully selecting DNN and input format combinations.

As a motivating example, consider ResNet-34 and 50 as the DNNs, and full resolution and 161x161 PNG thumbnails as the input formats. ResNet-34 and ResNet-50 execute at 6,861 and 4,513 images/second. On full resolution data, they achieve 72.72% and 75.16% accuracy on ImageNet, respectively. On low resolution data, they achieve 72.50% and 75.00% accuracy, respectively (when upscaling the inputs to 224x224 and using SMOL’s augmented training procedure). Full resolution and 161x161 thumbnails decode at 527 and 1,995 images/second, respectively. In this example, executing ResNet-50 on 161x161 thumbnails outperforms executing ResNet-34 on full resolution data, as end-to-end execution is bottlenecked by preprocessing costs.

Thus, SMOL jointly considers both the input resolution format and the DNN. For classification, SMOL also considers using a *single* DNN for accuracy/throughput trade-offs, instead of cascading a specialized DNN and target DNN.

For a given input format, SMOL will only consider DNNs that exceed the throughput of the preprocessing costs and select the highest accuracy DNN subject to this constraint. As we have demonstrated, in certain cases, this will result in selecting lower resolution data with more expensive DNNs, contrary to prior work.

Training DNNs for Low-resolution

As described above, SMOL can use low-resolution visual data to decrease preprocessing costs. However, naively using low-resolution can decrease accuracy, especially for target DNNs. For example, using a standard ResNet-50 with native 161x161 images upscaled to the standard 224x224 input resolution results in a *10.8% absolute drop in accuracy*. This drop in accuracy

is larger than switching from a ResNet-50 to a ResNet-18, i.e., nearly reducing the depth by a third. To alleviate the drop in accuracy, SMOL can train DNNs to be aware of low-resolution. This procedure can recover, or even exceed, the accuracy of standard DNNs.

SMOL trains DNNs to be aware of low-resolution by augmenting the input data at training time. At training, SMOL will downsample the full-resolution inputs to the desired resolution and then upsample them to the DNN input resolution. SMOL will do this augmentation in addition to standard data augmentation. By purposefully introducing downsampling artifacts, these DNNs can be trained to recover high accuracy on low-resolution data.

We show that this training procedure can recover the accuracy of full resolution DNNs when using lossless low-resolution data, e.g., PNG compression. However, when using lossy low-resolution data, e.g., JPEG compression, low-resolution DNNs can suffer a drop in accuracy. Nonetheless, we show that using lossy low-resolution data can be more efficient than using smaller, full-resolution DNNs.

4.2.5 An Optimized Runtime Engine for End-to-End Visual Inference

In order to efficiently execute *end-to-end* visual inference in the high-throughput setting, we must make proper use of all available hardware. We describe how to efficiently pipeline preprocessing and DNN execution for full use of hardware resources, how to optimize common preprocessing operations, how to place operations on CPUs or accelerators, and methods of partially decoding visual data. Several of these optimizations have been explored in other contexts, but not for end-to-end DNN inference [54, 57].

Efficient Use of Hardware

In order to efficiently use all available hardware resources, SMOL must efficiently pipeline computation, use threads, and use/reuse memory.

As executing DNNs requires computation on the CPU and accelerator, SMOL must overlap the computation. To do this, SMOL uses a multi-producer, multi-consumer (MPMC) queuing system to allow for multithreading. The producers decode the visual data and the consumers perform DNN execution. SMOL uses multiple consumers to leverage multiple CUDA streams. As preprocessing is data parallel and issuing CUDA kernels is low overhead, we find that setting the number of producers to be equal to the number of vCPU cores to be an efficient heuristic for non-NUMA servers.

An important performance optimization to effectively use the MPMC queuing system is reusing memory and efficient copying to the accelerator. Prior work that focuses on efficient preprocessing for training must pass memory buffers that contain the preprocessed images to the caller, which does not allow for efficient memory reuse. In contrast, the caller to SMOL only requires the result of inference, not the intermediate buffers. As a result, SMOL can reuse these buffers. Furthermore, accelerators require pinned memory for efficient memory transfer. Reusing pinned memory results in substantially improved performance. SMOL will further over-allocate memory to ensure that producer threads will not contend on consumers.

Optimizing Preprocessing Operations

A large class of common visual DNN preprocessing operations fall under the steps described in Section 4.2.1. Briefly, they include resizing, cropping, pixel-level normalization, data type conversion, and channel reordering. We can optimize these operations at inference time by fusing, reordering, and pre-computing operations.

To optimize these steps, SMOL will accept the preprocessing steps as a computation directed, acyclic graph (DAG) and performs a combination of rule-based and cost-based optimization of these steps. To optimize a computation DAG, SMOL will exhaustively generate possible execution plans, apply rule-based optimization to filter out plans, and perform cost-based optimization to select between the remaining plans.

SMOL contains rules of allowed operation reordering to generate the possible set of execution plans: 1) Normalization and data type conversion can be placed at any point in the computation graph, 2) Normalization, data type conversion, and channel reordering can be fused, 3) Resizing and cropping can be swapped.

Once SMOL generates all possible execution plans, SMOL will then apply the following rules to prune plans: 1) Resizing is cheaper with fewer pixels, 2) Resizing is cheaper with smaller data types (e.g., INT8 resizing is cheaper than FLOAT32 resizing), 3) Fusion always improves performance. We currently implement fusion manually, but code generation could also be applied to generate these kernels [138]. Given a set of plans after rule-based pruning, SMOL approximates the cost by counting the number of arithmetic operations in each plan for the given data types. SMOL will select the cheapest plan.

Preprocessing Operator Placement

In addition to optimizing common preprocessing operations, SMOL can place preprocessing operations on the CPU or accelerator. Depending on the input format/resolution and DNN, the relative costs of preprocessing and DNN execution may differ. For example, small specialized NNs may execute many times faster than preprocessing, but a state-of-the-art Mask R-CNN may execute slower than preprocessing.

As a result, to balance preprocessing and DNN execution costs, it may be beneficial to place operations on either the CPU or accelerator. Furthermore, many preprocessing operations (e.g., resizing, normalization) are efficient on accelerators, as the computational patterns are similar to common DNN operations.

If DNN execution dominates, then SMOL will place as many operations on the CPU as possible, to balance costs. If preprocessing cost dominate, then SMOL will place as many operations on the accelerator as possible. Since preprocessing operations are sequential, SMOL need only consider a small number (typically under 5) configurations for a given model and image format.

Partial and Low-Fidelity Decoding

Overview of Visual Compression Formats. We briefly describe salient properties of popular visual compression formats, including the popular JPEG, HEVC/HEIC, and H.264 compression formats. We describe the decoding of the data and defer a description of encoding to other texts [141, 161, 177]. Decoding generally follows three steps: 1) entropy decoding, 2) inverse transform (typically DCT-based), and 3) optional post-processing for improved visual fidelity (e.g., deblocking).

Importantly, the entropy decoders in both JPEG and HEVC (Huffman decoding and arithmetic decoding respectively) are not efficient on accelerators for DNNs as it requires substantial branching. Furthermore, certain parts of decoding can be omitted, e.g., the deblocking filter, for reduced fidelity but faster decoding times.

Leveraging partial decoding. When low-resolution visual data is not available, SMOL optimizes preprocessing by partially decoding visual data. Many DNNs only require a portion of the image for inference, or *regions of interest (ROI)*. For example, many image classification networks centrally crop images, so the ROI is the central crop. Computing face embeddings crops faces from the images, so the ROIs are the face crops. Furthermore,

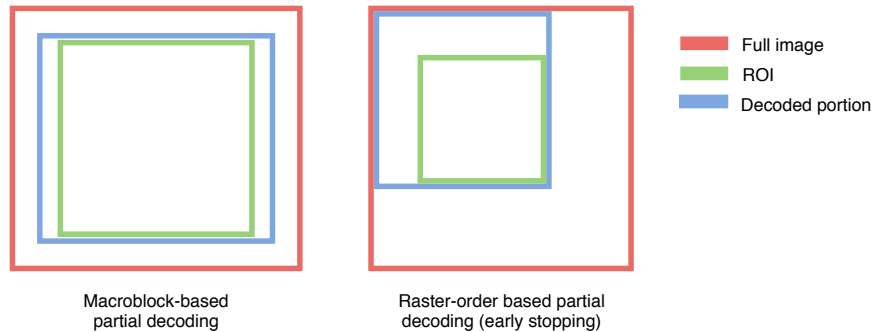


Figure 4.8: Examples of partial decoding for images. On the left, the ROI is the central crop of the image. For JPEG images, SMOL can decode only the macroblocks that intersect the ROI. For image formats that do not allow for independently decoding macroblocks, SMOL can partially decode based on raster order (right).

these networks often take standard image sizes, e.g., 224×224 . We show two examples in Figure 4.8. Computing ROIs may require expensive upstreaming processing in some applications, e.g., executing a detection DNN.

Many image compression formats allow for partial decoding explicitly in the compression standard and all compression formats we are aware of allow for early stopping of decoding. We give three instantiations of partial decoding in popular visual compression formats and provide a list of popular visual data compression formats and which features they contain in Table 4.5. We then describe how to use these decoding features for optimized preprocessing.

First, for the JPEG image compression standard, each 8×8 block, or *macroblock*, in the image can be decoded independently (partial decoding) [172]. Second, the H.264 and HEVC video codecs contain deblocking filters, which can be turned off at the decoding stage for reduced computational complexity at the cost of visual fidelity (reduced fidelity decoding) [161, 177]. Third, the JPEG2000 image compression format contains “progressive” images, i.e., downsampled versions of the same image, that can be partially decoded to a specific resolution (multi-resolution decoding) [166].

SMOL accepts as an optional input an ROI for a given image. If an ROI is specified, SMOL will only decode the parts of the image necessary to process the ROI.

Partial decoding. We present two methods of partially decoding visual data. We show examples of each in Figure 4.8.

ROI decoding. When only a portion of the image is needed, e.g., for central cropping or when selecting a region of interest (ROI), only the specified portion of the image need be

Format	Type	Low-fidelity features
JPEG	Image	Partial decoding
PNG, WebP	Image	Early stopping
HEIC/HEVC	Image/Video	Reduced fidelity decoding
H.264	Video	Reduced fidelity decoding
VP8	Video	Reduced fidelity decoding
VP9	Video	Reduced fidelity decoding

Table 4.5: A list of popular visual data formats and their low-fidelity features.

decoded. To decode this portion of the image, SMOL will first find the smallest rectangle that aligns with the 8x8 macroblock border and contains the region. Then, SMOL will decode the rectangle and return the crop.

Early stopping. For compression formats that do not explicitly allow for partial decoding, SMOL can terminate decoding on parts of the image that are not necessary. For example, if only the top $N \times N$ pixels are required for inference, SMOL will terminate decoding after decoding the top $N \times N$ pixels.

Reduced-fidelity decoding. Several visual compression formats contain options for reduced fidelity decoding. While there are several ways to reduce the fidelity of decoding for decreased preprocessing costs, we focus on methods that are easily specified with existing decoding APIs. Specifically, we explore reduced fidelity in the form of disabling the deblocking filter. SMOL will profile the accuracy of the specialized and target NNs with and without the deblocking filter and choose the option that maximizes throughput.

4.2.6 Evaluation

We evaluated SMOL on eight visual datasets and show that SMOL can outperform baselines by up to $5.9\times$ for image datasets and $10\times$ for video datasets at a fixed accuracy level.

Experimental Setup

Overview. We evaluate our optimizations on four image datasets and four video datasets. The task for the image datasets is image classification. The task for the video datasets is an aggregation query for the number of target objects per frame. For classification, we use accuracy and throughput as our primary evaluation metrics. For the aggregation queries, we measure query runtime as the error bounds were respected.

Dataset	# of classes	# of train im.	# of test im.
bike-bird	2	23k	1k
animals-10	10	25.4k	2.8k
birds-200	200	6k	5.8k
imagenet	1,000	1.2M	50K

Table 4.6: Summary of dataset statistics for the still image datasets we used in our evaluation. The datasets range in difficulty and number of classes. `bike-bird` is the easiest dataset to classify and `imagenet` is the hardest to classify.

Datasets. We use `bike-bird` [26], `animals-10` [9], `birds-200` [170], and `imagenet` [47] as our image datasets. These datasets vary in difficulty and number of classes (2 to 1,000). In contrast, several recent systems study only binary filtering [11, 30, 124]. We summarize dataset statistics in Table 4.6. We used thumbnails encoded in a standard short size of 161 in PNG, JPEG ($q = 75$), and JPEG ($q = 95$).

For the video datasets, we used `night-street`, `taipei`, `amsterdam`, and `rialto` as evaluated by BLAZEIT [93]. We used the original videos as evaluated by BLAZEIT and further encoded the videos to 480p for the low-resolution versions.

Model configuration and baselines. For SMOL, we use the standard configurations of ResNets (18, 34, and 50). We find that these models span a range of accuracy and speed while only requiring training three models. We note that if further computational resources are available at training time, further models could be explored.

Image datasets. For the image datasets, we use the following two baselines. First, we use standard ResNets and vary their depths, specifically choosing 18, 34, and 50 as these are the standard configurations [74]. We refer to this configuration as the naive baseline; the naive baseline does not have access to other image formats. Second, we use TAHOMA as our other baseline, specifically a representative set of 8 models from TAHOMA cascaded with ResNet-50, our most accurate model. We choose 8 models due to the computational cost of training these models, which can take up to thousands of GPU hours for the full set of models. We use ROI decoding for SMOL as these datasets use central crops.

Video datasets. We used the original BLAZEIT code, which uses a “tiny ResNet” as the specialized NN and a state-of-the-art Mask R-CNN [73] and FGFA [190] as target networks. We replicate the exact experimental conditions of BLAZEIT, except we use SMOL’s optimized runtime engine, which is substantially more efficient than BLAZEIT.

Hardware environment. We use the AWS `g4dn.xlarge` instance type with a single NVIDIA T4 GPU attached unless otherwise noted. The `g4dn.xlarge` has 4 vCPU cores with 15 GB of RAM. A vCPU is a hyperthread, so 4 vCPUs consists of 2 physical cores. Compute intensive workloads, such as image decoding, will achieve sublinear scaling compared to a single hyperthread. The `g4dn.xlarge` instance is approximately cost balanced between vCPU cores and the accelerator.

`g4dn.xlarge` is optimized for DNN inference. Namely, the T4 GPU is significantly more power efficient than GPUs designed for training, e.g., the V100. However, they achieve lower throughput as a result; our results are more pronounced when using the V100 (e.g., using the `p3.2xlarge` instance). Our baselines use CPU decoding as a case study, as not all visual formats are supported by hardware decoders, e.g., the popular HEIC (used by all new iPhones) and WebP (used by Google Chrome) formats. Finally, we note that throughputs can be converted to power or cost by using more vCPU cores, but we use a single hardware environment for ease of comparison.

Further experiments. Due to limited space, we include experiments comparing SMOL to other frameworks in an extended version of this chapter [99].

Cost Models and Benchmarking SMOL

We investigated the efficiency of pipelining in SMOL and our choice of using min in cost modeling (Section 4.2.3). We measured the throughput of SMOL when only preprocessing, only executing the DNN computational graph, and when pipelining both stages.

We first consider low-resolution images (JPEG $q = 75$) to ensure the system was under full load. Preprocessing, DNN execution, and end-to-end inference achieve 5.9k, 4.2k, and 3.6k im/s respectively. Even at full load, SMOL only incurs a 16% overhead compared to the throughput predicted by its cost model. In contrast, TAHOMA’s cost model would predict a throughput of 2.5k im/s, a 30% error.

Furthermore, across all ResNet-50 configurations, SMOL’s cost model (i.e., min) achieves the lowest error compared to other heuristics (i.e., DNN execution only and sum). Its average error is 5.9%, compared to 217% (DNN execution only) and 23% (sum).

Image Analytics Experiments

End-to-end speedups. We evaluated SMOL and baselines (TAHOMA and standard

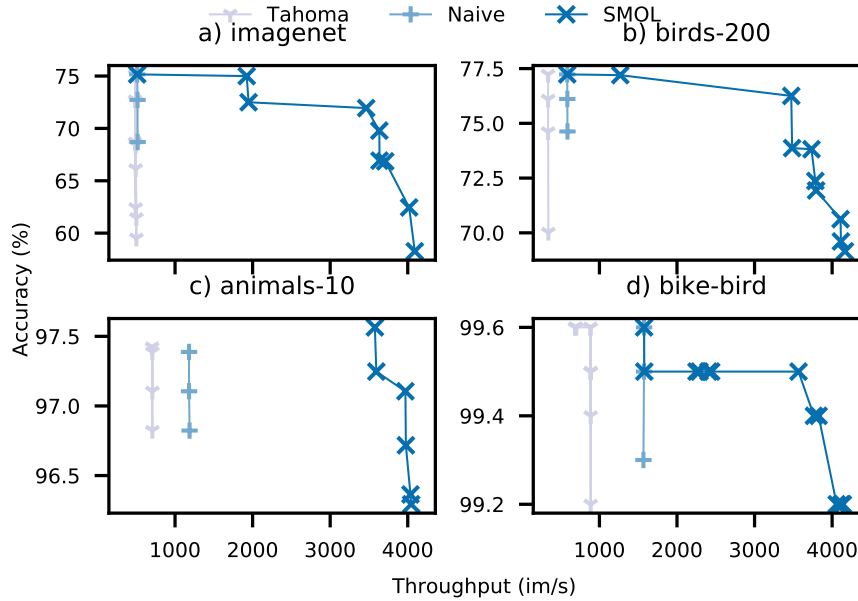


Figure 4.9: Throughput vs accuracy for the naive baseline, TAHOMA, and SMOL on the four image datasets (Pareto frontier only). SMOL can improve throughput by up to $5.9\times$ with no loss in accuracy.

ResNets on full resolution data) on the image datasets shown in Table 4.6.

We first investigated whether SMOL outperforms baselines when all optimizations were enabled. We plot in Figure 4.9 the Pareto frontier of baselines and SMOL for different input format and DNN configurations. As shown, SMOL can improve throughput by up to $5.9\times$ with no loss in accuracy relative to ResNet-18 and up to $2.2\times$ with no loss in accuracy relative to ResNet-50. Furthermore, SMOL can improve the Pareto frontier compared to all baselines. Notably, TAHOMA’s specialized models performs poorly on complex tasks and are bottlenecked on image preprocessing.

Importantly, we see that the naive baselines (i.e., all ResNet depths) *are bottlenecked by preprocessing* for all datasets. Any further optimizations to the DNN execution alone, including model compression, will *not improve end-to-end throughputs*. The differences in baseline throughputs are due to the native resolution and encoding of the original datasets: `birds-200` contains the largest average size of images. The throughput variation between ResNets depths is due to noise; the variation is within margin of error.

While we show below that both low resolution data and preprocessing optimizations contribute to high throughput, we see that SMOL’s primary source of speedups depends

Format	Acc (reg, 50)	Acc (low-resol, 50)	Acc (reg, 34)	Acc (low-resol, 34)
Full resol	75.16%	57.72%	72.72%	64.76%
161, PNG	70.92%	75.00%	68.30%	72.50%
161, JPEG ($q = 95$)	68.93%	71.94%	66.92%	69.79%
161, JPEG ($q = 75$)	64.02%	63.23%	62.45%	62.45%

Table 4.7: Effect of training procedure and input format on accuracy for ResNet-50 and ResNet-34 on `imagenet`. SMOL can achieve an accuracy throughput trade-offs by changing the input format, achieving no accuracy loss for easier datasets.

on the dataset. First, SMOL can achieve the same or higher accuracy by simply using low resolution data for some `bike-bird` and `animals-10`. Second, for `imagenet`, a fixed model will result in slightly lower accuracy ($\leq 1\%$) when using lossless image compression. However, when using a *larger* model, SMOL can recover accuracy.

Comparison against TAHOMA. TAHOMA underperforms the naive solution of using a single, accurate DNN for preprocessing bound workloads. This is primarily due to overheads in cascades, namely coalescing and further preprocessing operations. Specifically, TAHOMA cascades a small DNN into a larger DNN. These smaller DNNs are less accurate than the larger DNNs and thus require many images to be passed through the cascade for higher accuracy, especially on the more complex tasks. The images that are passed through must be copied again and further resized if the input resolutions are different.

Effect of training procedure. We investigated the effect of the training procedure for low-resolution input formats. We trained ResNet-50 on: 1) full resolution, 2) 161 short-side PNG, 3) 161 short-side JPEG ($q = 95$), and 4) 161 short-side JPEG ($q = 75$).

We show the accuracy of these conditions in Table 4.7 for `imagenet`, our hardest dataset. As shown, low-resolution aware training can nearly recover the accuracy of full resolution data even on this difficult dataset. Low-resolution training can fully recovery accuracy on `bike-bird` and `animals-10`.

Video Analytics Experiments

We evaluated SMOL on the four video datasets described above. We used the exact experimental configuration from BLAZEIT as the baseline, with the exception of executing BLAZEIT’s specialized NNs in SMOL’s optimized runtime engine. SMOL’s runtime engine is substantially more efficient than BLAZEIT’s.

As shown in Figure 4.10, SMOL can improve throughput by up to $2.5\times$ at a fixed error

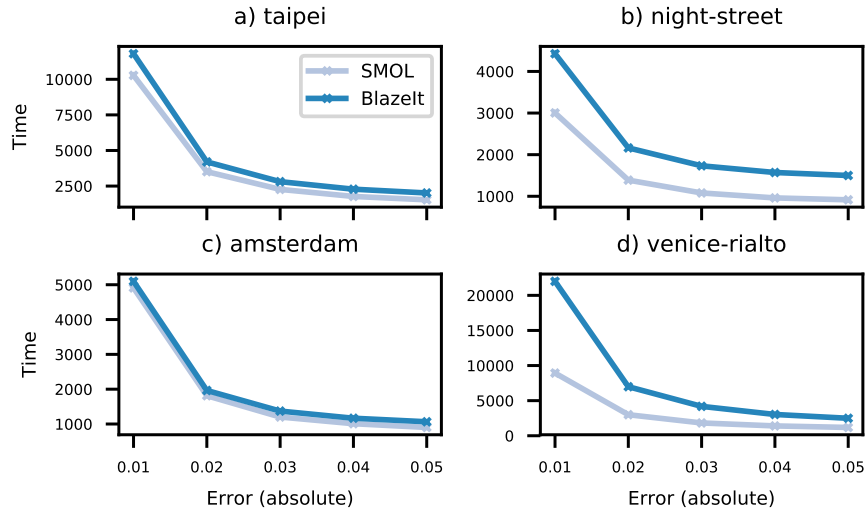


Figure 4.10: Query execution time vs requested error for BLAZEIT and SMOL on the four video datasets we evaluated. As shown, SMOL consistently outperforms BLAZEIT by using more accurate specialized NNs, which reduces sampling variance, and lower resolution data, which reduces preprocessing costs.

level. Furthermore, SMOL outperforms BLAZEIT in all settings. SMOL’s primary speedups for *night-street* and *rialto* come from more accurate, but more expensive specialized NNs. Despite the specialized NNs being more expensive, they reduce sampling variance more, as they are more accurate. As a result, fewer samples are necessary for a fixed error target. SMOL’s primary speedups for *taipei* and *amsterdam* come from leveraging low resolution video, as it is cheaper to preprocess.

4.2.7 Discussion

In this Section, we show that preprocessing can be the bottleneck in end-to-end DNN inference. We show that the preprocessing costs are accounted for incorrectly in cost models for selecting models in visual analytics applications. To address these issues, we build SMOL, an optimizing runtime engine for end-to-end DNN inference. SMOL contains two novel optimization for end-to-end DNN inference: 1) an improved cost model for estimating DNN throughput and 2) joint optimizations for preprocessing and DNN execution that leverage low-resolution data. We evaluate SMOL and these optimizations and show that SMOL can achieve up to $5.9\times$ improved throughput on end-to-end DNN inference.

Chapter 5

Specifying Errors in ML Deployments

All of the methods we have discussed in the previous two chapters rely on an accurate target model. Unfortunately, as we discuss in Chapter 1, ML models can be unreliable, returning inaccurate results. These inaccurate results can degrade the accuracy of unstructured data queries relative to the ground truth.

Beyond causing errors in analytics, errors in ML models can have cascading consequences in other deployments. For example, errors in ML models have already cause fatal autonomous vehicle accidents [171]. As a result, it is critical to find errors in these ML models and their root causes.

To find these errors, I propose two abstractions for allowing domain experts to specify when errors in ML models and the data used to train these models may be occurring. The first abstraction I have developed, model assertions, allows users to manually specify when errors may be occurring. These assertions can find errors with high precision and with few lines of code. They can further be used to retrain models in a cost efficient manner. The second abstraction I have developed, learned observation assertions (LOA), learns from existing labels where errors in newly labeled data may occur. LOA can be used to find errors with up to $2\times$ higher precision than baselines.

5.1 Model Assertions

ML is increasingly deployed in complex contexts that require inference about the physical world, from autonomous vehicles (AVs) to precision medicine. However, ML models can misbehave in unexpected ways. For example, AVs have accelerated toward highway lane

dividers [116] and can rapidly change their classification of objects over time, causing erratic behavior [39, 133]. 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: ML models fail for diverse and reasons unknown before deployment. Thus, existing solutions that focus on verifying training, including formal verification [103], whitebox testing [140], monitoring training metrics [151], and validating training code [136], *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 [17, 143] 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 [17] only apply to deployments that have ground truth labels, e.g., click-through rate prediction, but not to deployments that *lack labels*.

As a step towards more robust QA for complex ML applications, we have found that ML developers can often specify *systematic* errors made by ML models: certain classes of errors are repetitive and can be checked automatically, via code. For example, in developing a video analytics engine, we noticed that object detection models can identify boxes of cars that flicker rapidly in and out of the video (Figure 5.1), indicating some of the detections are likely wrong. Likewise, our contacts at an AV company reported that LIDAR and camera models sometimes disagree. While seemingly simple, similar errors were involved with a fatal AV crash [133]. 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*, an abstraction to monitor and improve ML model quality. Model assertions are inspired by program assertions [61, 168], one of the most common ways to monitor software. A model assertion is an arbitrary function over a model’s input and output that returns a Boolean (0 or 1) or continuous (floating point) severity score to indicate when faults may be occurring. For example, a model assertion that checks whether an object flickers in and out of video could return a Boolean value over each frame or the number of objects that flicker. While assertions may not offer a complete specification of correctness, we have found that assertions are easy to specify in many domains (Section 5.1.1).

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

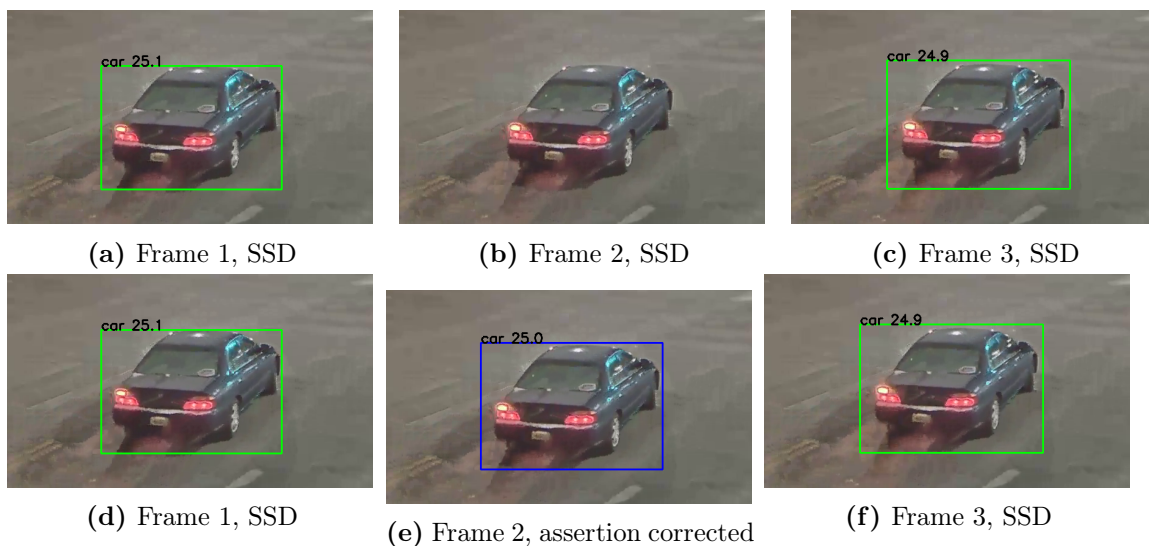


Figure 5.1: Top row: example of flickering in three consecutive frames of a video. The object detection method, SSD [120], failed to identify the car in the second frame. **Bottom row:** example of correcting the output of a model. The car bounding box in the second frame can be inferred using nearby frames based on a consistency assertion.

First, we show that model assertions can be used for **runtime monitoring**: they can be used to log unexpected behavior or automatically trigger corrective actions, e.g., shutting down an autopilot. Furthermore, model assertions can often find *high confidence* errors, where the model has high certainty in an erroneous output; these errors are problematic because prior uncertainty-based monitoring would not flag these errors. Additionally, and perhaps surprisingly, we have found that many groups are also interested in validating human-generated labels, which can be done using model assertions.

Second, we show that assertions can be used for **active learning**, in which data is continuously collected to improve ML models. 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 [42, 158]. 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? A data point can be 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 (Section 5.1.2). We show that our bandit algorithm can reduce labeling costs by up to 40% over traditional uncertainty-based methods.

Third, we show that assertions can be used for **weak supervision** [128, 147]. 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., a TV news host (identifier) should have consistent gender (attribute), or that certain predictions should (or should not) exist in temporally related outputs, e.g., cars in adjacent video frames (Figure 5.1). 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, `OMG`¹, 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 the remainder of this section, we describe `OMG`'s API, using `OMG` with active learning, using `OMG` with weak supervision, and our evaluation of `OMG`.

5.1.1 Model Assertions

We describe the model assertion interface, examples of model assertions, how model assertions can integrate into the ML development/deployment cycle, and its implementation.

¹`OMG` is a recursive acronym for `OMG Model Guardian`.

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 a *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 use 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, our library OMG can register arbitrary Python functions as model assertions.

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 and describe how one might look for assertions in other domains in an extended version of this work [101].

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 [162]. Second, complex ML deployments are developed by large teams, of which some developers may not have the ability to manage all parts of the application. As a result, it is critical to be able to do QA collaboratively to cover the application end-to-end.

Analyzing TV news. We spoke to a research lab studying bias in media via automatic analysis. This lab 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 [101]. We further describe how model assertions can be implemented via our consistency API for TV news in Section 5.1.3.

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 [46]. 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 can disagree. We implemented a model assertion that projects the 3D boxes onto the 2D camera plane 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 [30, 83, 86, 93, 94, 181] trained on MS-COCO [119], a corpus of still images. These still image object detection methods are deployed on video for detecting objects. None of these systems aim to detect errors, even though errors can affect analytics results.

In developing such systems, we noticed that objects flicker in and out of the video (Figure 5.1) and that vehicles overlap in unrealistic ways [101]. 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 [146]. Our researcher contacts mentioned that AF predictions from DNNs can rapidly oscillate. The European Society of Cardiology guidelines for detecting AF require at least 30 seconds of signal before calling a detection [53]. Thus, predictions should not rapidly switch between two states. A developer could specify this model assertion, which could be implemented to monitor ECG classification deployments.

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

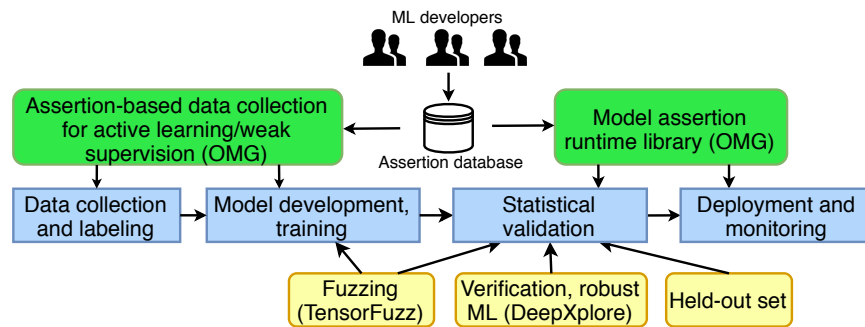


Figure 5.2: System diagram of how model assertions can integrate into the ML development/deployment pipeline. Users can collaboratively add to an assertion database. We also show how related work can be integrated into the pipeline.

techniques, including verification, fuzzing, and statistical techniques, as shown in Figure 5.2.

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 they be ML models or human labelers. Perhaps surprisingly, we have found several groups to also be interested in monitoring human label quality. Thus, 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 Section 5.1.2.

Third, model assertions can be used to generate weak labels to further train ML models without additional human labels. We describe how OMG accomplishes this via consistency assertions in Section 5.1.3. Users can also register their own weak supervision rules.

Implementing Model Assertions in OMG

We implement a prototype library for model assertions, OMG, that works with existing ML training and deployment frameworks. We briefly describe OMG’s implementation.

OMG logs user-defined assertions as callbacks. The simplest way to add an assertion is through `AddAssertion(func)`, where `func` is a function of the inputs and outputs (see below). OMG also provides an API to add consistency assertions as described in §5.1.3. Given this database, OMG requires a callback after model execution that takes the model’s input and output as input. Given the model’s input and output, OMG 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 5.1:

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

For active learning, OMG will take a batch of data and return indices for which data points to label. For weak supervision, OMG 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 OMG uses to improve model quality: BAL for active learning and consistency assertions for weak supervision.

5.1.2 Using Model Assertions for Active Learning with BAL

We introduce an algorithm, 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 [4] and our industrial contacts label data in bulk.

Given a set of data points that triggered model assertions, OMG must select which points to label. There are two key challenges which make 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 simplifying assumptions. We describe the statistical model we assume, the resource-unconstrained algorithm, our simplifying assumptions, and BAL. We note that, while the resource-unconstrained algorithm can produce statistical guarantees, BAL does not. We instead empirically verify its performance in Section 5.1.4.

Data selection as multi-armed bandits. We cast the data selection problem as a multi-armed bandit (MAB) problem [14, 21]. 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 settings [27, 123, 145], but we assume that the arms have context associated with them (i.e., severity scores from model assertions) and give submodular rewards (defined below). The rewards are possibly time-varying. We further assume there is an (unknown) smoothness parameter that determines the similarity between arms of similar contexts (formally, the α in the Hölder condition [55]). The following presentation is inspired by Chen et al. [33].

Concretely, we assume the data will be labeled in T rounds and denote the rounds $t = 1, \dots, T$. We refer to the set of n data points as $N = \{1, \dots, n\}$. 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. The user must select a set of arms $S^t = \{x_{s_1}, \dots, x_{s_{B^t}}\}$ such that $|S^t| \leq B^t$. We 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:

$$\Delta R(\{m\}, A) = R(A \cup \{m\}) - R(A). \quad (5.1)$$

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 $m \notin C$

$$\Delta R(\{m\}, A) \geq \Delta R(\{m\}, C). \quad (5.2)$$

Resource-unconstrained algorithm. Assuming an infinite labeling and computational budget, we describe an algorithm that selects data points to train on. If we assume that rewards for individual arms can be queried, then a recent bandit algorithm (CC-MAB [33]) can achieve a regret of $O(cT^{\frac{2\alpha d}{3\alpha d}} \log(T))$ for α to be the smoothness parameter. 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 [33].

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 simplifying assumptions and use these to modify CC-MAB for the resource-constrained setting. Our simplifying assumptions are that 1) data points with similar contexts (i.e., x_i^t) are interchangeable, 2) data points with higher severity scores have higher expected marginal gain, and 3) reducing the number of triggered assertions will increase accuracy.

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.

Leveraging these assumptions, we can simplify CC-MAB to require less computation for training models and to not require labels for all data points. 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 is inspired by ϵ -greedy algorithms [167]. 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.

5.1.3 Consistency Assertions and Weak Supervision

Although developers can write arbitrary Python functions as model assertions, we found that many assertions can be specified using an even simpler, high-level abstraction that we called *consistency assertions*. This interface allows OMG to generate multiple 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 Section 5.1.1). 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, 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. This could return the gender attribute.

Given these two functions, OMG 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, OMG can thus *automatically* generate weak labels for data points that do not satisfy the consistency assertions. Notably, OMG provides another way of producing labels for training that is complementary to human-generated labels and other sources of weak labels. OMG is especially suited for unstructured sources, e.g., video. We show in Section 5.1.4 that these weak labels can automatically increase model quality.

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 $y_{i,j}$:

- $\text{Id}(y_{i,j})$ returns an *identifier* for the output $y_{i,j}$, which is simply an opaque value.
- $\text{Attrs}(y_{i,j})$ 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 `AddConsistencyAssertion(Id, Attrs, T)`.

Examples. We briefly describe how one can use consistency assertions in several ML tasks motivated in Section 5.1.1:

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 a 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.

Generating Assertions and Labels from the API

Given the Id, Attrs, and T values, OMG 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.

OMG 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 , OMG will assert that at most one transition can occur within a T -second window; this can be overridden. For example, an identifier appearing is valid, but an identifier appearing, disappearing, then appearing is invalid. If a violation occurs, OMG will propose to remove, modify, or add predictions. In the latter case, OMG 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 5.1). OMG 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.

Task	Model	Assertions
TV news	Custom	Consistency (§5.1.3, <code>news</code>)
Object detection (video)	SSD [120]	Three vehicles should not highly overlap (<code>multibox</code>), identity consistency assertions (<code>flicker</code> and <code>appear</code>)
Vehicle detection (AVs)	Second [182], SSD	Agreement of Point cloud and image detections (<code>agree</code>), <code>multibox</code>
AF classification	ResNet [146]	Consistency assertion within a 30s time window (<code>ECG</code>)

Table 5.1: A summary of tasks, models, and assertions used in our evaluation.

5.1.4 Evaluation

Experimental Setup

We evaluated OMG 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 5.1.

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 Section 5.1.3. 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 [30, 83, 86, 93, 94, 181], 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 a ResNet-34 SSD [120] (henceforth SSD) model pretrained on MS-COCO [119]. We deployed SSD for detecting vehicles in the `night-street` (i.e., `jackson`) video that is commonly used [30, 83, 94, 181]. 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. The `flicker` and `appear` assertions are implemented with our consistency API as described in Section 5.1.3.

Autonomous vehicles. We studied the problem of object detection for autonomous

vehicles using the NuScenes dataset [29], 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 [114, 182] 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 [146]. Unfortunately, the full dataset used in [146] is not publicly available, so we used the CINC17 dataset [1]. 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 Section 5.1.3.

Model Assertions can be Written with High Precision, 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 assertions (see Table 5.1 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 5.2, 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 whether model assertions could be written with high precision. To test this, we randomly sampled 50 data points that triggered each assertion and manually checked

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

Table 5.2: Number of lines of code (LOC) for each assertion. All assertions could be written in under 60 LOC including helper functions. The assertion main body could be written in under 25 LOC in all cases.

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

Table 5.3: Precision of model assertions deployed on 50 randomly selected examples. Model assertions can be written with 88-100% precision across all domains.

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 5.3, model assertions achieve at least 88% precision in all cases.

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 [35, 83, 93, 94]. Furthermore, sampling solutions that are based on confidence would be unable to identify these errors.

To determine whether 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.

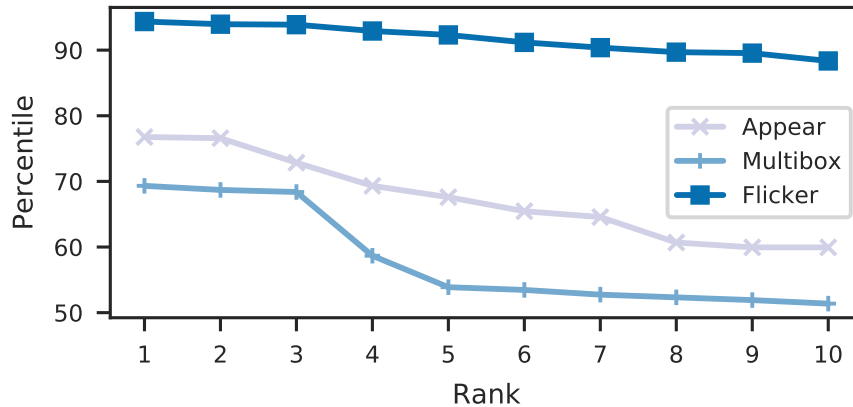


Figure 5.3: Percentile of confidence of the top-10 ranked errors by confidence found by OMG for video analytics. The x-axis is the rank of the errors caught by model assertions, ordered by rank. The y-axis is the percentile of confidence among all the boxes. Mdel assertions can find errors where the original model has high confidence (94th percentile).

As shown in Figure 5.3, 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.

We further show that model assertions can identify errors in human labels, which effectively have a confidence of 1. These results are shown in Kang et al. [101].

Model Assertions can Improve Model Quality via Active Learning

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

We asked whether multiple model assertions can 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” [158], uniform sampling assertions, and BAL for the active learning strategies. We used the mAP metric for both datasets, which is widely used for object detection [73, 119]. We defer hyperparameters to Kang et al. [101].

As we show in Figure 5.4, BAL outperforms both random sampling and uncertainty sampling on both datasets after the first round, which is required for calibration. BAL also outperforms uniform sampling from model assertions by the last round. For `night-street`, at a fixed accuracy threshold of 62%, BAL uses 40% fewer labels than random and uncertainty

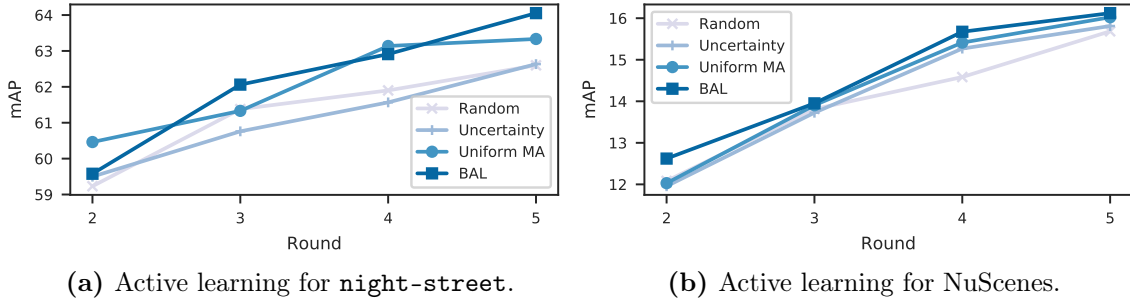


Figure 5.4: Performance of random sampling, uncertainty sampling, uniform sampling from model assertions, and BAL for active learning. BAL improves accuracy on unseen data and can achieve an accuracy target (62% mAP) with 40% fewer labels compared to random and uncertainty sampling for *night-street*. BAL also outperforms both baselines for the NuScenes dataset.

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) [60].

These results are expected, as prior work has shown that uncertainty sampling can be unsuited for deep networks [156].

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).

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 used 1,000 weak labels and the same training procedure as in active learning.

Table 5.4 shows that model assertion-based weak supervision 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.

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

Table 5.4: Accuracy of pretrained and weakly supervised models. Weak supervision can improve accuracy with no human-generated labels.

5.1.5 Discussion

In this Section, we introduced model assertions, a model-agnostic technique that allows domain experts to indicate errors in ML models. We showed that model assertions can be used at runtime to detect high-confidence errors, which prior methods would not detect. We proposed methods to use model assertions for active learning and weak supervision to improve model quality. We implemented model assertions in a novel library, OMG, and demonstrated that they can apply to a wide range of real-world ML tasks, improving monitoring, active learning, and weak supervision for ML models.

5.2 Learned Observation Assertions

Machine learning (ML) is increasingly being deployed in complex applications with real-world consequences. For example, ML models are being deployed to make predictions over perception data in autonomous vehicles (AVs) [102], with potentially fatal consequences for errors, such as striking pedestrians [171]. Thus, quality assurance and testing of ML pipelines are of paramount concern [10, 136, 179, 184].

A critical component of ML deployments is the curation of *high-quality* training data, in which crowd workers produce labels over data. Similar to how errors in tabular data results in downstream errors in query results, erroneous training data (e.g., Figure 5.5) can lead to subsequent safety repercussions for trained models. As such, finding these errors is critical, which we focus on in this work.

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

Recent work has proposed Model Assertions (MAs) that indicate when errors in ML

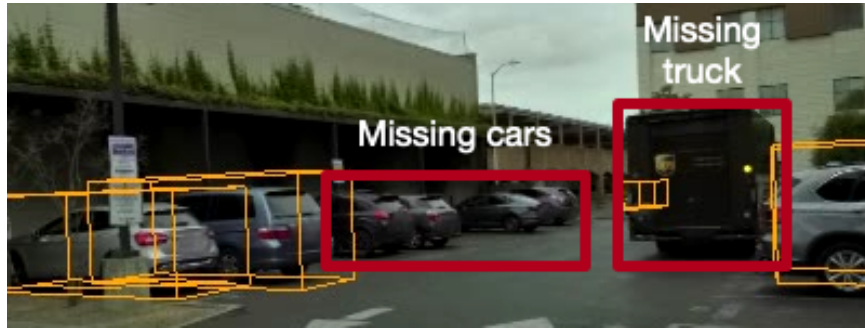


Figure 5.5: Example of human labels (orange) and missing labels (red) in the Lyft Perception dataset. The black truck highlighted is within 25m of the AV. Such errors can cause downstream issues with perception and planning systems.

model predictions or labels may be occurring [101]. MAs are black-box functions over model inputs and outputs that return a quantitative severity score indicating when an ML model or human-proposed label may have an error. For example, a MA may assert that a prediction of a box of a car should not appear and disappear in subsequent frames of a video. MAs can be used to monitor the ML models in deployment, and to flag problematic data to label and retrain the model.

However, in our experience deploying MAs in a real-world AV perception training pipeline, we have found several major challenges. First, users must manually specify MAs, which can be challenging for complex ML deployments. Second, calibrating severity scores so that higher severity scores indicate a higher chance of error is challenging. This is especially important as organizations have limited resources to evaluate potential errors in ML models or human labels. Third, ad-hoc methods of specifying severity scores ignores organizational resources [163] that are already present: large amounts of ground-truth labels and existing ML models.

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

Our first contribution, LOA, allows users to specify properties of interest for perception tasks. LOA contains three components: data associations, feature distributions, and application objective functions. LOA can be used to specify assertions without ad-hoc code or

severity scores by automatically transforming the specification into a probabilistic graphical model and scoring data components, producing statistically grounded severity scores.

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

Our second contribution is methods of leveraging *organizational resources* [163], i.e., existing labels and ML models, to automatically specify severity scores via LOA. Users specify features over data, which are used to automatically generate *feature distributions*, and *application objective functions* (AOFs) to guide the search for errors. Feature distributions take sets of observations and output a probability of seeing a feature of the input. For example, a feature distribution might take a 3D bounding box of a car and return the likelihood of encountering that box volume. AOFs transform feature distribution values for the application at hand. For example, if we wish to find likely tracks (e.g., a track missed by human labels), the AOF could simply return the feature distributions' value. If we wish to find unlikely tracks (e.g., a “ghost” track that an ML model erroneously predicts), the AOF could return one minus the feature distributions' value.

We evaluate FIXY on two real-world AV datasets, the publicly available Lyft Level 5 perception dataset [104] as well as an internally collected dataset. Both datasets were annotated by leading commercial labeling vendors. Despite best efforts from these vendors [34], we find a number of labeling errors via FIXY, some of which could cause safety violations (e.g., in Figures 5.5 and 5.12). Using LOA, we discovered errors in 70% of the scenes of the Lyft Level 5 autonomous vehicle dataset, a popular autonomous vehicle dataset. We further show that FIXY can achieve recalls of up to 75% when searching for errors in these datasets, while achieving 2× higher precision for finding label error than hand-crafted MAs. Furthermore, FIXY can find novel errors in ML models that the hand-crafted MAs in previous work are unable to find, and finds high-confidence errors that uncertainty sampling misses.

In the remainder of this section, we describe LOA's interface and how FIXY integrates in the broader context of ML deployments, how FIXY automatically learns feature distributions, how FIXY scores assertions, and our evaluation of LOA.

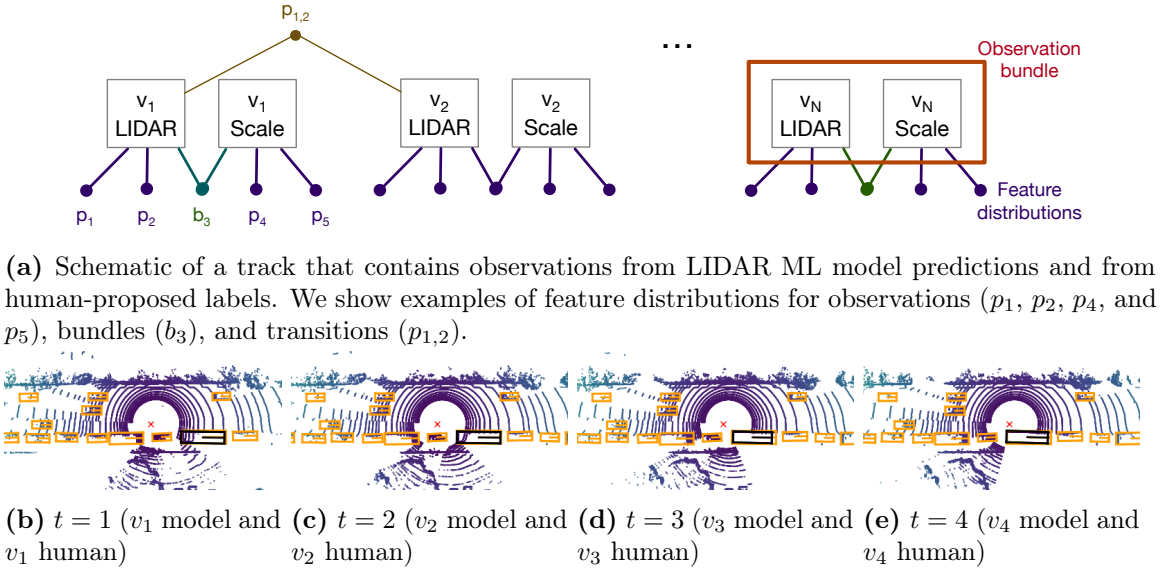


Figure 5.6: Example of the factor graph (top) and corresponding LIDAR point cloud data (bottom). The track is in black and other human-proposed labels are in orange for reference.

5.2.1 Example and Background

ML workflow. As an example, we describe the ML deployment pipeline for our AVs, focusing on labeling data for perception systems. Other organizations deploy similar pipelines, e.g., as documented by Kaparthy [102].

Our AV deployment pipeline is a continuous process, in which ML models are trained, tested, and deployed on vehicles. Because ML models are continuously exposed to new and different scenarios, we continuously collect and label data, which is subsequently used to develop and retrain ML models [17].

Label quality is of paramount concern: erroneous labels can lead to downstream errors, which in turn can lead to safety violations. Vendors that provide labels are not always accurate, which is in contrast to the large body of work that assumes datasets are “gold.” For our perception system, the most egregious errors are when objects are entirely missed in labeling. We show examples of missing labels in Figure 5.5, in which a truck and several cars were missed by the human labeler.

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

allows better utilization of auditing resources.

Model assertions. MAs are user-provided, black box functions over ML model inputs and outputs that indicate if the ML model has an error [101]. MAs can be deployed at test time to indicate possible errors so corrective actions can be taken. They can additionally be used to select data that produces errors for labeling, e.g., as studied by Kang et al. [101], as many organizations continuously collect data to label.

Unfortunately, MAs are specified in an ad-hoc manner. They require users to write programs to specify exactly what forms of errors occur and ad-hoc severity scores to indicate the likelihood of an error. We have found that these ad-hoc procedures can be challenging to use.

Factor graphs. FIXY generates graphical models from data and feature distributions. We specifically consider factor graphs due to the ease of representing data and distributions [111].

Given a set of random variables $X = \{X_1, \dots, X_n\}$, a factor graph represents a factorization of a joint distribution $g(X_1, \dots, X_n)$. Assume that the joint distribution can be factorized in terms of a set of functions f_j , which we will call *factors*, and $S_j \subseteq X$

$$g(X_1, \dots, X_n) = \prod_{j=1}^m f_j(S_j). \quad (5.3)$$

Formally, we can represent a factor graph as a graph $G = (X, F, E)$, where X and F are two disjoint sets of nodes. The graph is *bipartite*, meaning that each edge connects a node in X to a node in F , but no edge connects nodes in X among themselves nor nodes in F among themselves. For every factor $f_j \in F$, there is an edge that connects it to X_i if and only if $X_i \in S_j$ in the factorization of g .

We consider specific factor graphs that are automatically generated by FIXY, as described later in this chapter.

LIDAR. We extensively use and show LIDAR data and predictions over LIDAR data as examples of missing human labels or ML model predictions. LIDAR is generated by pulsing light and timing the returns of the pulsed light [173]. With accurate timings, LIDAR data gives accurate distance measurements and are represented as point clouds. In this chapter, we show birds-eye view of LIDAR data: concentric circles indicate same distances from the LIDAR sensor and we draw predicted boxes over the scenes. We show an example in Figure 5.6. LIDAR figures in white background are from the Lyft Level 5 perception

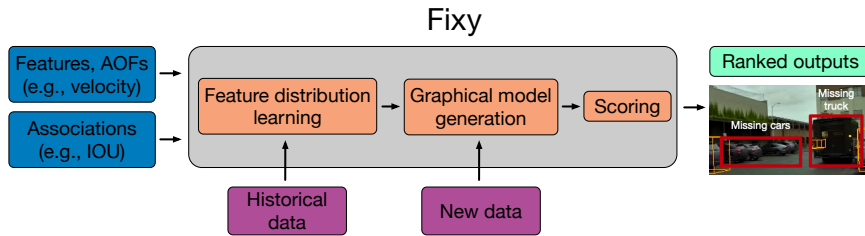


Figure 5.7: System diagram for FIXY. Users provide features over perception data (e.g., box volume) and associations between observations. Given these inputs, FIXY will learn feature distributions, generate graphical models, score new data, and output potential errors.

dataset [104] and figures in black background are from our internal dataset.

5.2.2 System Overview

Goals. FIXY aims to enable users to find errors in ML labeling pipelines and in ML models, primarily in the form of missing labels. In particular, FIXY aims to reduce manual effort by only requiring users to specify natural quantities (e.g., box volume, velocity) as opposed to specifying the exact form of errors as model assertions expect users to do.

Inputs and outputs. We first denote human-proposed labels and ML model outputs as “observations.” As input, FIXY takes a set of observations. As output, FIXY returns a ranked list of (potentially a subset of) observations, where higher ranked observations are ideally more likely to contain errors.

Offline, FIXY takes already-present labels to learn feature distributions over features of the observations. FIXY will then use this data to rank potential errors.

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

We show a system diagram in Figure 5.7. Users need only provide the features (and data to be ranked). Once the feature distributions are learned, FIXY will rank potential errors for auditing.

Workflow. FIXY contains an offline (distribution learning) and online (error ranking) phase. In the offline phase, FIXY will learn feature distributions from existing labels. In the online phase, FIXY will rank potential errors.

We have found that users of label checking tools are often non-experts in coding and ML tools, so we have opted for simplicity in LOA. Thus, a user of FIXY need only specify features and optionally AOFs. In particular, many features are *already computed* for use in other pipelines so can be reused (e.g., object volume, velocity, and distance from vehicle). Thus, the features can be specified in few lines of code, as we show below.

Worked example. Consider the use case of finding missing human labels of 3D bounding boxes over LIDAR point cloud data. For example, Figure 5.5 show several missing cars and a missing truck.

In this example, we have two sources of observations: predictions from an ML model and human labels. To find these errors, the user will: 1) associate observations and 2) specify features. Then, FIXY will automatically score and rank potential errors.

The analyst first associates observations within a time step (i.e., overlapping model predictions and human labels) and between adjacent timesteps (i.e., objects across time). To do so, the user can specify that observations with high box overlap are associated. While this is provided by default by FIXY, the user can also write a short amount of code using the intersection over union (IOU):

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

The analyst then specifies features. As a concrete example, the analyst may specify a feature that computes box volume. The user need only provide code to compute the box volume: FIXY will learn distribution of box volumes and use it to find anomalous boxes.

```
# KDEObsDistribution takes features and learns a
# KDE density estimator over the features
class VolumeDistribution(KDEObsDistribution):
    def feature(self, box):
        vol = box.width * box.height * box.length
        return vol
```

The user can also specify other features, such as object velocity. The two code snippets above (and another other features the user wishes to specify) are all that a user need to provide to FIXY.

Given the associations between observations and the features, FIXY will learn the likelihood of encountering specific feature values offline, using already-collected resources.

Once these feature distributions are learned, FIXY will score and rank new data, ideally with potential errors ranked higher. Concretely, consider Figure 5.5. Although not shown, an ML model highlighted the truck in a time-consistent way. Since the track is highly consistent, FIXY returns a high likelihood of an error. An expert auditor can then verify if the potential error is actually a missing label.

5.2.3 Learned Observation Assertions

LOA provides a simple means of specifying associations between observations and specifying associations between observations and feature distributions. Intuitively, applications that contain observations over time and over multiple modalities/models may have observations that are associated across time/modalities. Furthermore, feature distributions may operate over individual observations or groups of observations. We show an example of a compiled LOA graph and corresponding sensor data observations in Figure 5.6.

In this section, we provide a formal description of LOA. However, users interface with LOA via a Python library. In particular, users only need to specify features over which distributions are learned and methods of associating observations. Our implementation provides class interfaces where users can override the feature computation (for the feature distributions) and the association method (for associating observations). We show an example in Section 5.2.2 of the code the user needs to provide.

Overview

LOA contains elements for allowing users to specify how observations interact with each other and how feature distributions interact with observations. Our implementation of LOA is embedded in Python for ease of integration with standard ML packages. Since perception data often contains spatial and temporal components, we allow users to construct *observation bundles* within a single time step and *tracks* across time. We collectively refer to observations, bundles, and tracks as OBTs. LOA then allows features to be specified over any OBT. Finally, the user can specify application objective functions (AOFs) over any feature distribution.

Scene Syntax

Overview. We consider *scenes* of data, which consists of *observations* and *features* over these observations. Our syntax consists of specifying how observations relate to each other within a scene and how features relate to groups of observations.

Formalism. A scene consists of a set of tracks. Each track contains a set of observation bundles. An observation bundle contains observations from different modalities, such as LIDAR, vision, and for offline data, human proposals of labels. We summarize our syntax notation in Table 5.5.

Formally, we denote the scene (i.e., set of tracks) as $s = \{\tau\}$. Each track consists of an indexed sequence of observation bundles, $\tau = (\beta_0, \dots, \beta_n)$. Each observation bundle consists of a set of observations, $\beta = \{\omega_{\tau, \beta}\}$.

In order to reason about erroneous or unusual artifacts in the perception system, we define features over the elements of the scene. Users can assign features to any of the elements of the scene; these assignments are often done automatically (e.g., a volume feature would apply over every observation). Concretely, features can be over observations, observation bundles, tracks, or entire scenes.

Formally, π , the feature function, maps each element to its features. For example, $\pi(\omega_{\tau, \beta})$ are the features assigned to the observation in track τ in bundle β , which could be a feature on the volume of the object detected. Similarly, $\pi(\tau)$ assigns track τ its features, which could be the total number of observations within that track.

In addition to features over discrete groups of observations, we provide syntax for features over adjacent observations within a track (“transition features”), i.e., $\pi(\beta_i, \beta_{i+1})$. As a concrete example, we have implemented a transition feature for the estimated instantaneous velocity. We note that this syntax is for convenience, as it could be implemented via track features. Nonetheless, we have found it useful in our applications to allow for transition features.

Finally, AOFs can be specified over any feature distribution. These AOFs are numeric transformations of the returned feature distribution score, e.g., the identity function, the zero function, or $f(x) = 1 - x$.

Syntactic element	Meaning
s	Scene
τ	Track
β	Observation bundle
ω	Observation
π	Feature mapping function

Table 5.5: Table of syntactic elements in FIXY’s DSL.

Generating Graphical Models

FIXY will compile the scene, feature distributions, and AOFs to a graphical model, which is used to score groups of observations. FIXY uses these scores to flag potential errors.

To compile a scene, FIXY will create nodes for each observation and feature distribution. Then, FIXY will create edges between each feature distribution and the observation it applies over. If a feature distribution applies to a single observation, FIXY will create a single edge. If a feature distribution applies to a group of observations (e.g., an observation bundle or track), FIXY will create one edge between each observation in the group and the feature distribution.

Once the graphical model is constructed, FIXY can then score any OBT. FIXY will score an observation by the negative log-likelihood implied by its feature distributions. The score of a group of observations is the sum of the scores of the observations, normalized by the number of feature distributions. We defer the full discussion of scoring and a worked example to Section 5.2.5.

5.2.4 Feature Distributions

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

To fit these feature distributions, FIXY takes as input scalar or vector valued features over OBTs. For example, a feature over an observation may take a bounding box and return the volume of the box as described in Section 5.2.2. The user may also manually specify feature distributions to rank severity (e.g., distance of an object to the AV) or to filter

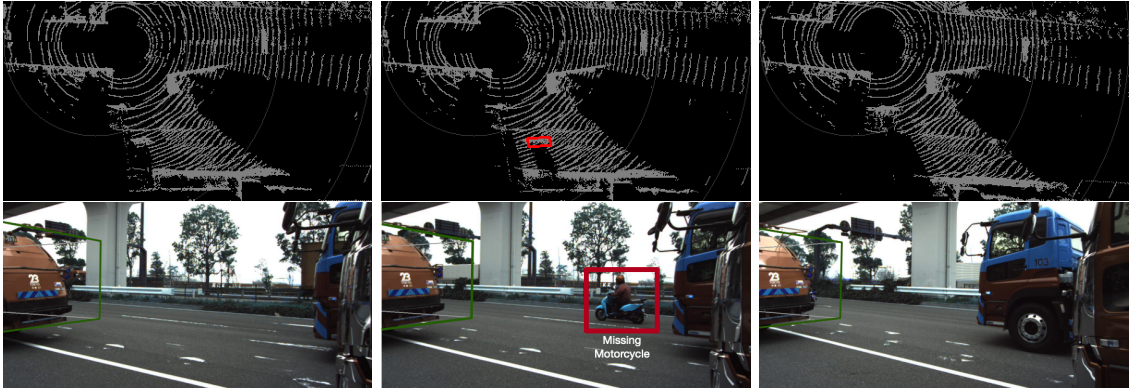


Figure 5.8: Example of a motorcycle (highlighted in red) missed by human proposals. We show both the LIDAR point cloud data (top) and the camera view (bottom).

certain instances (e.g., only search for errors in detecting pedestrians). Finally, FIXY takes an optional AOF, which can be applied per feature or over the resulting score.

We describe the feature types, their specification, and how FIXY fits them below.

Feature Types

FIXY contains features over OBTs and transitions. While transition features can be implemented as track features, we provide a syntactic element for ease of use.

FIXY’s first feature type are features over single observations. Each feature is associated with a specific observation type (e.g., a feature over a LIDAR model prediction). These features are typically used to specify time-independent information over the predictions. For example, a feature may take a 3D box prediction from a LIDAR model and return the box volume. The observation feature would be over box volumes in this case.

FIXY’s second feature type are features over observation bundles. These features are typically used to specify consistency between observations of the same object in a single time step. For example, consider the intuition that observations within bundles should agree on object class. To specify this, a user could provide a feature that returns 0 if all the classes agree and 1 otherwise. The feature would then learn the Bernoulli probability of the class agreement between observation types.

FIXY’s third feature type are features between observations or bundles in adjacent time

steps within a track. These features are typically used to specify information over time-dependent quantities or consistency. For example, a feature could specify the velocity estimated by box center offset.

FIXY's fourth feature type are features over entire tracks. Although rare, these features can be used to normalize scores over entire tracks.

Learning Feature Distributions

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

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

The density estimators have hyperparameters. We have found that default hyperparameters work in all cases we tried, so we defer exploring hyperparameters to future work.

Application Objective Functions

AOFs wrap data feature distributions to transform them into an application-specific probabilities to guide the search for labeling errors. As such, they take scalar values and return scalar values. The most common operations are taking the inverse and setting the probability to 0/1 under certain conditions. For example, when searching for likely tracks, the application objective function may be the identity. In contrast, when searching for unlikely tracks, the application objective function may invert the probability.

5.2.5 Scoring Relative Plausibility

Given the compiled factor graph, FIXY can score any OBT. FIXY will score the observations via the sum of log likelihood of the feature distributions transformed by the application

objective functions. Formally, given the AOFs f_i , the score for an observation ω is

$$\sum_{\pi_i \in \pi(\omega)} \ln(f_i(\pi_i(\omega))). \quad (5.4)$$

The score of any component in the graph is the sum of the scores of the observations, normalized by the total number of features that connect to the component. We normalize by the number of features so that components of different sizes are comparable (e.g., a track with 10 observations compared to a track with 100 observations).

Consider the missing truck in Figure 5.5 and suppose the ML model predicted it in two adjacent time steps ($t = 1, 2$) for simplicity. Suppose the predicted box volumes are 44.8 m^3 and 45.9 m^3 , and the predicted velocity was 2 m/s . In this case, the scores of the volumes may be 0.37 and 0.39 respectively, and the score for the velocity may be 0.21. The score for the track would be $(\ln(0.37) + \ln(0.39) + \ln(0.21))/3 = -1.17$. Since FIXY only uses the score to rank, this would be compared to other scores.

5.2.6 Applications

We provide examples of applying FIXY to finding different kinds of errors in ML applications. For all applications, we assume that the predictions are 3D bounding boxes over LIDAR point cloud data. We further assume access to two features: an observation feature over box volume and a transition feature over estimated velocity.

Finding missing tracks. In this application, we are interested in finding tracks that human proposals missed entirely. For example, Figure 5.8 shows a motorcycle close to the AV but is only visible for a short period of time due to occlusion. It is important to find such errors as this may cause ML models to misclassify motorcycles at deployment time.

To find such errors, we additionally execute a 3D bounding box prediction model over the data. Given the ML model predictions, we associate ML model predictions and human proposal in the same frame if they have high box overlap.

The AOF zeros out any track that contains any human proposals. The remaining tracks contain only model predictions and are scored as usual, with the intuition that consistent predictions from the model are likely to be correct. We show an example of a high probability track (Figure 5.8) and low probability track (Figure 5.9).

Finding missing labels within tracks. We are interested in finding errors in labels proposed by humans that should belong to an existing track. For example, Figure 5.10 shows a car trailing the AV, where the first frame is missing the car box.

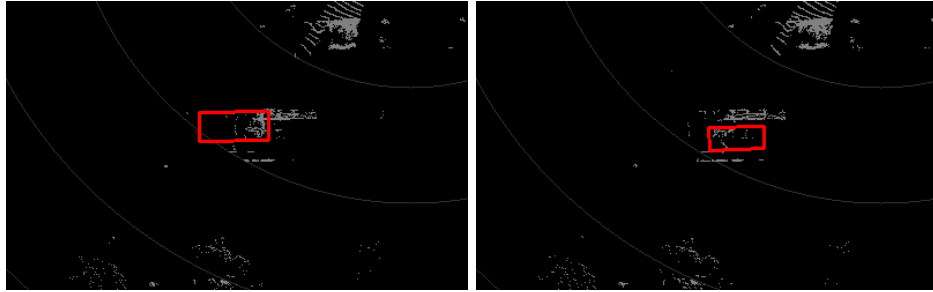


Figure 5.9: Example of an unlikely track. Predictions are inconsistent within a track, suggesting that they are spurious.

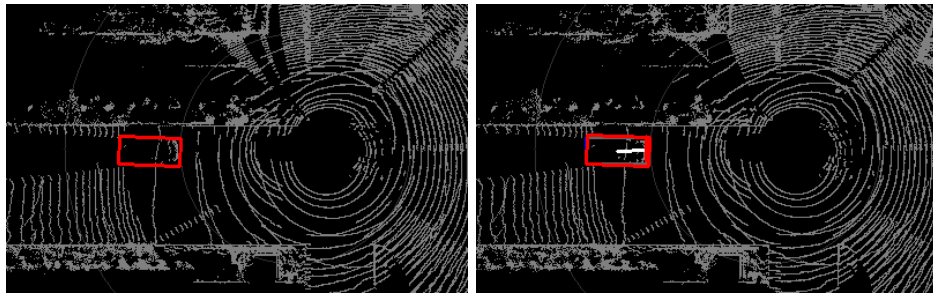


Figure 5.10: Example of missing human label within a track that FIXY can find. The left panel only contains an ML model prediction while the right contains both a human label and an ML model prediction.

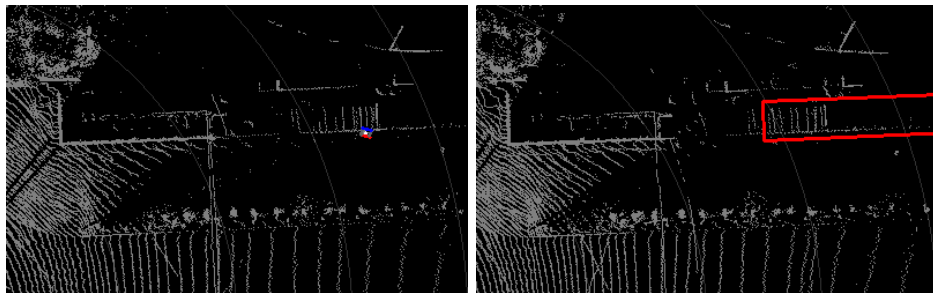


Figure 5.11: Example of a low probability bundle. The box of the person and truck highly overlap, but are strongly inconsistent in box volume.

To find such errors, we use the 3D bounding box prediction model's predictions. The association of observations into bundles is done as above. The AOF zeros out the probability of any bundle that contains a human proposal and any track that does not contain any human proposals. Thus, the remaining bundles only contain ML model predictions and are in tracks that contain at least one human proposal.

The remaining bundles are scored as usual, with the intuition that predicted boxes that produce high probability bundles are likely to be correct predictions. We show an example of a high probability bundle (Figure 5.10) compared to a low probability bundle (Figure 5.11).

Finding erroneous ML model predictions. We are interested in finding erroneous ML model predictions. As such, we assume there are no human proposals. We show an example of an erroneous track in Figure 5.13, where the truck is inconsistently predicted.

The AOF inverts the probability of each feature, with the goal of inverting the ranking of the tracks that are likely to be correct and the tracks that are likely to be incorrect.

5.2.7 Evaluation

We evaluated FIXY on whether it can find errors in ML pipelines. We show that FIXY can find errors in human-proposed labels that are difficult to specify with ad-hoc MAs and novel errors in ML model predictions that prior work cannot find.

Experimental Setup

Datasets. We evaluated FIXY on two AV perception datasets: an internal dataset from our research organization and the publicly available Lyft Level 5 perception dataset [104]. The Lyft dataset has been used to develop models [189] and host competitions [160]. Both datasets consists of many scenes of LIDAR and camera data that were densely labeled with 3D bounding boxes by leading external vendors for human labels (“human-proposed labels”).

We executed FIXY on 46 scenes from the Lyft dataset (the entire validation set, i.e., not seen at training time) and 13 scenes from our internal dataset. Additionally, we asses the recall of FIXY on a scene from our internal dataset that we vetted carefully.

The class labels, sampling rate, and physical sensor layout differ between the two datasets, showing that FIXY can apply across dataset characteristics.

Observation sources. We used three sources of observations: human-proposed labels, LIDAR model predictions [114, 189], and expert auditor labels. All sources predict 3D bounding boxes. We focus on the common classes of car, truck, pedestrian, and motorcycle.

Features. We used the features shown in Table 5.6. These features consist of features that were automatically learned from data (volume, velocity, count) in addition to features for selecting more egregious errors (distance, model only).

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

Table 5.6: Description of features we used in this evaluation. Model only and count were manually specified features.

Baselines. We compared against manually designed ad-hoc MAs developed by Kang et al. [101] and uncertainty sampling. The ad-hoc MAs were designed to find errors in similar settings to ours, across both human-proposed labels and ML model predictions. Uncertainty sampling is commonly used in active learning [158].

Users of FIXY are typically non-experts in coding and ML tools (Section 5.2.2). As such, we focus on simple ad-hoc MAs (e.g., ones developed by Kang et al. [101]) and low-code features in FIXY. Each feature required fewer than 6 lines of code to implement.

Both datasets were vetted by leading vendors for human labels. Thus, we find errors that were not found in an external audit.

FIXY can Find Missing Tracks

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

Experimental setup. To find tracks entirely missed by human labelers, we associated LIDAR observations and human observations by box overlap within the same frame and associated observations within a track by box overlap across time. We further deployed the features described above. For the ad-hoc MA baseline, we used the “consistency” assertion as described by Kang et al. [101]. For comparison purposes, we ordered the ML model predictions randomly and by model confidence.

We manually checked the top 10 potential errors as proposed by FIXY and ad-hoc model assertions (in some cases, fewer than 10 potential errors were flagged; we use the maximum number in these cases). We measured the precision among these potential errors, where a higher precision indicates that there are more errors within the top 10 proposals. For the Lyft dataset, we measured the precision across every scene in the validation set (i.e., data that was not seen during model training) that we discovered errors. For our internal dataset,

Method	Dataset	Precision at top 10	Precision at top 5	Precision at top 1
FIXY	Lyft	69%	70%	67%
Ad-hoc MA (rand)	Lyft	32%	30%	24%
Ad-hoc MA (conf)	Lyft	39%	40%	39%
FIXY	Internal	76%	100%	100%
Ad-hoc MA (rand)	Internal	49%	64%	66%
Ad-hoc MA (conf)	Internal	71%	86%	66%

Table 5.7: Precision at top 10, 5, and 1 of FIXY and ad-hoc MA baselines for finding tracks missed by humans. FIXY outperforms baselines by up to 2 \times .



Figure 5.12: Examples of labeling errors in the Lyft dataset. The missing objects in these examples can be within 20 meters the autonomous vehicle and several are in motion: vehicles in motion are the most important to detect.

we focused on the scene that failed audit.

Results: recall. To assess the recall of FIXY, we exhaustively audited a 15 second scene from our internal dataset. It contained 24 missing tracks. In this scene, FIXY achieved a recall of 75%, finding 18 of the missing tracks in the top 10 ranked errors per-class. We believe this result is reflective of the larger dataset.

We further manually searched for errors in the Lyft dataset and found errors in 32 of the 46 scenes (70% of scenes). Unfortunately, due to the sheer number of errors in the Lyft dataset, we were unable to perform recall experiments on the level of boxes. However, LOA found errors in *100% of the scenes with errors* in the top 10 ranked errors. This dataset may be used in a different manner internally, but we were unable to find public guidelines for dataset use.

Results: precision. FIXY outperforms on finding errors on precision in both datasets (Table 5.7) by aggregating information across observations in tracks, which is difficult to do with ad-hoc MAs.

We show three examples of errors FIXY found in the Lyft dataset in Figure 5.12. Many of these errors are close to the AV and are clearly visible. These errors are problematic

because they can confuse ML models and could potentially cause downstream issues.

Discussion. To further contextualize our results, we note that FIXY uncovered an error that was missed by an internal audit. Specifically, the motorcycle track described in Section 5.2.6 (Figure 5.8) was not found in our initial internal audit. Given the short time period the motorcycle was visible, it can be difficult to find for both crowd workers and auditors. Nonetheless, it is critical to be accurately labeled for two key reasons. First, clean training data is critical for liability purposes should an accident occur. Second, the motorcycle is close to the autonomous vehicle, which is especially problematic for downstream planning.

Our internal model does better than the public model. This is primarily because the Lyft dataset is very noisy: our internal model was trained on *already audited* data, which is of higher quality and results in more calibrated model predictions. These results highlight the need for high quality data: noisy data results in lower performing models.

Furthermore, the open-sourced Lyft perception dataset has a number of vehicles that were not labeled. We plan to open source the errors we have found to aid in the development of consistent labeling for the Lyft dataset.

FIXY can Find Missing Observations

As a case study, we searched for missing observations in human-proposed tracks. We applied the following AOFs. We set the probability of an observation in a bundle with a human proposal to 0. We set the probability of any track without a human proposal to 0. For FIXY, we then ranked the bundles by likelihood. For the ad-hoc MA baseline, we random ordered bundles.

We were only able to find a single example of such a missing observation. For this example, FIXY ranked the missing observation at the top. We show the missing observation in Figure 5.10 and examples of low probability missing observations in Figure 5.11. The feature distributions correctly identify consistent predictions within tracks and correctly downweights inconsistent predictions.

FIXY can Find Novel ML Prediction Errors

We further investigated whether or not FIXY can find errors in LIDAR model predictions. For this use-case, we did not assume access to human-proposed labels.

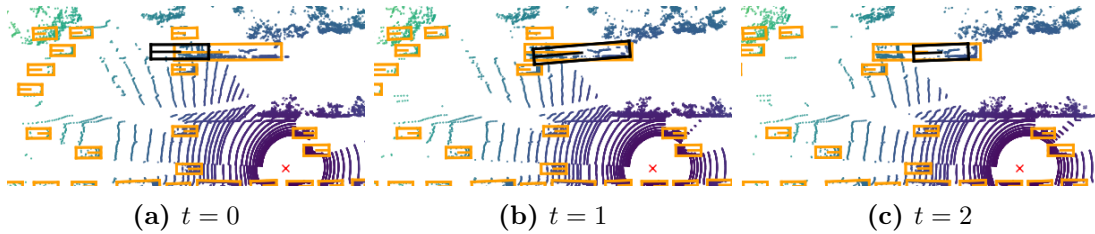


Figure 5.13: Example of a model error (in black) in the Lyft dataset not found by ad-hoc model assertions. We show ground-truth boxes from human labels in orange for reference. The erroneous prediction overlaps across frames, but is not consistent. FIXY can find such errors as they produce unlikely values under learned feature distributions.

Experimental setup. Unlike for finding errors in human-proposed labels, ad-hoc MAs can achieve high precision for errors in ML model predictions. As such, we deployed three ad-hoc MAs as used in Kang et al. [101] (`appear`, `flicker`, and `multibox`). Briefly, the `appear` assertion asserts that an observation should have observations in nearby timestamps, the `flicker` assertion asserts that an observation should not appear and disappear rapidly, and the `multibox` assertion asserts that 3 boxes should not overlap. These assertions can be reproduced in FIXY with hand-tuned features.

In addition to ad-hoc MAs, we additionally compared to uncertainty sampling, in which we sampled predictions around a confidence threshold.

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

Results and discussion. Across these scenes, FIXY achieves a precision at 10 of 82% while uncertainty sampling achieved a precision of 42%. We note that errors we found with FIXY were not found by ad-hoc MAs. Many of these errors have tracks without missing time stamps (so will not trigger the `flicker` assertion) and are longer than two observations (so will not trigger the `appear` assertion). We show an example of such a track in Figure 5.13, in which the predictions overlap across frames, but in an unlikely way.

Furthermore, in contrast to uncertainty sampling, FIXY uncovers errors with high model confidence. FIXY discovered errors in ML model predictions with confidences as high as 95%, which uncertainty sampling would miss.

5.2.8 Discussion

To address the problem of finding errors in labels and ML models, we propose Learned Observation Assertions (LOA) and implement FIXY. LOA allows users to specify data-driven feature distributions to indicate which data points are potentially erroneous. Our prototype implementation of LOA, FIXY, leverages existing organizational resources (trained ML models and existing labeled data) to find labeling errors. We show that FIXY can find errors in human labels up to $2\times$ more effectively than prior research work.

Chapter 6

Discussion

In the past few years, the computational complexity of ML methods (particularly DNNs) has increased dramatically. For example, GPT-3, a state-of-the-art language model, takes 358 Tflops to execute. The trend towards larger models does not seem to be abating. Furthermore, the scale of these models appears to be growing faster than advances in hardware. As a parallel trend, more unstructured data is being generated than ever before, including user-generated video, images, and text on social media, scientific video footage, etc.

Querying this data using the most accurate methods is infeasible today and will likely become even more expensive. As a result, queries over large-scale unstructured data will continue to be far too expensive for exhaustive execution. This is particularly the case for resource-limited applications, including scientific applications. However, this is also the case for applications that require human intervention, such as high-stakes labeling.

In my experience in deploying the systems I have built, several organizations have already experienced these constraints. For example, the ecologists I have collaborated with have limited computational resources: even executing state-of-the-art models on their dataset is difficult. Autonomous vehicle companies collect too much data for expert annotation.

Given these trends, users of these ML methods in analytics and beyond will need principled ways of trading off between accuracy and computational/human resources.

In this dissertation, we have introduced novel systems and techniques to navigate these trade-offs. First, we have introduced end-to-end, proxy-based query processing methods for ML-based queries. These methods leverage cheap approximations to expensive methods in a principled manner. We have further introduced abstractions for finding errors in ML-based deployments, which can find errors with high precision. These methods use existing

resources to guide limited human resources. As our results show, navigating trade-offs between accuracy and computational resources is possible and can give orders of magnitude speedups with limited error.

6.1 Future Directions

Although we have shown the promise of expressing and accelerating queries over unstructured data, much work remains for the data systems community. In particular, we have found that many scientists in the social and life sciences have workloads that can be accelerated using ML methods. However, in our discussions with practitioners, we have found several repeated key requirements, which existing work today does not handle.

Usability. The most critical requirement is that solutions that require dedicated engineering support are challenging for non-experts. To draw an analogy, `pandas` and `R` are the frameworks of choice for analyzing structured data amongst this group of practitioners. In contrast, teams of engineers can deploy higher effort, but also more scalable solutions, such as `Spark`. Even setting up a `Postgres` database can be challenging for non-experts.

In this dissertation, we have described systems and algorithms at the level of `Postgres` or `Spark`, in which experts can leverage state-of-the-art methods. However, much work remains to enable non-experts to deploy such methods.

Training new models. Furthermore, in many scientific investigations, the investigators are searching for novel phenomena. The novel phenomena are either unclear to the investigators at the beginning of the search or not in pretrained models. As a result, using off-the-shelf models is insufficient for these applications. Coupled with the difficulty in deploying state-of-the-art ML methods, leveraging ML methods for studying these novel phenomena becomes increasingly challenging.

Much of the effort in data systems has focused on building solutions for the best-funded engineering efforts. However, some of the most important work for our society happens in low-resource settings, where deploying large-scale systems is difficult. As this dissertation shows, carefully designed techniques and abstractions can aid in both low- and high-resource settings by directly connecting end use cases with system/algorithm design. We hope that the data systems community continues such work to enable users in a wide range of settings.

Bibliography

- [1] 2017. AF Classification from a short single lead ECG recording: the PhysioNet/Computing in Cardiology Challenge 2017. <https://physionet.org/challenge/2017/>
- [2] 2018. MLPerf. <https://mlperf.org/>.
- [3] 2019. NVIDIA TensorRT. <https://developer.nvidia.com/tensorrt>
- [4] 2020. Scale API: The API For Training Data. <https://scale.ai/>
- [5] Swarup Acharya, Phillip B Gibbons, and Viswanath Poosala. 1999. Aqua: A fast decision support systems using approximate query answers. In *PVLDB*. 754–757.
- [6] Sameer Agarwal, Henry Milner, Ariel Kleiner, Ameet Talwalkar, Michael Jordan, Samuel Madden, Barzan Mozafari, and Ion Stoica. 2014. Knowing when you’re wrong: building fast and reliable approximate query processing systems. In *SIGMOD*. ACM, 481–492.
- [7] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. 2013. BlinkDB: queries with bounded errors and bounded response times on very large data. In *EuroSys*. ACM, 29–42.
- [8] Alan Akbik, Duncan Blythe, and Roland Vollgraf. 2018. Contextual String Embeddings for Sequence Labeling. In *COLING*. 1638–1649.
- [9] Corrado Alessio. 2019. Animals-10. <https://www.kaggle.com/alessiocorrado99/animals10>
- [10] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software engineering for machine learning: A case study. In *ICSE-SEIP*. IEEE, 291–300.
- [11] Michael R Anderson, Michael Cafarella, Thomas F Wenisch, and German Ros. 2019. Predicate Optimization for a Visual Analytics Database. *ICDE (2019)*.

- [12] Rosana Ardila, Megan Branson, Kelly Davis, Michael Henretty, Michael Kohler, Josh Meyer, Reuben Morais, Lindsay Saunders, Francis M Tyers, and Gregor Weber. 2019. Common voice: A massively-multilingual speech corpus. *arXiv preprint arXiv:1912.06670* (2019).
- [13] Elizabeth Arens. 2019. Always Up-to-Date Guide to Social Media Image Sizes. <https://sproutsocial.com/insights/social-media-image-sizes-guide/>
- [14] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multi-armed bandit problem. *Machine learning* 47, 2-3 (2002), 235–256.
- [15] Favyen Bastani and Samuel Madden. 2022. OTIF: Efficient Tracker Pre-processing over Large Video Datasets. (2022).
- [16] Favyen Bastani, Oscar Moll, and Sam Madden. 2020. Vaas: video analytics at scale. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2877–2880.
- [17] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, et al. 2017. Tfx: A tensorflow-based production-scale machine learning platform. In *SIGKDD*. ACM.
- [18] Sara Beery, Dan Morris, and Siyu Yang. 2019. Efficient Pipeline for Camera Trap Image Review. *arXiv preprint arXiv:1907.06772* (2019).
- [19] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. 2010. A theory of learning from different domains. *Machine learning* 79, 1 (2010), 151–175.
- [20] V. Bentkus and F. Gotze. 1996. The Berry-Esseen Bound for Student’s Statistic. *The Annals of Probability* 24, 1 (1996), 491–503.
- [21] Donald A Berry and Bert Fristedt. 1985. Bandit problems: sequential allocation of experiments (Monographs on statistics and applied probability). *London: Chapman and Hall* 5 (1985), 71–87.
- [22] Christopher M Bishop. 2006. *Pattern recognition and machine learning*. springer.
- [23] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *TACL* 5 (2017), 135–146.
- [24] Vladimir Braverman and Rafail Ostrovsky. 2013. Generalizing the layering method of indyk and woodruff: Recursive sketches for frequency-based vectors on streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 58–70.

- [25] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *NeurIPS* 33 (2020), 1877–1901.
- [26] Tom B Brown, Nicholas Carlini, Chiyuan Zhang, Catherine Olsson, Paul Christiano, and Ian Goodfellow. 2018. Unrestricted adversarial examples. *arXiv preprint arXiv:1809.08352* (2018).
- [27] Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. 2009. Pure exploration in multi-armed bandits problems. In *International conference on Algorithmic learning theory*. Springer, 23–37.
- [28] Michael Buckland and Fredric Gey. 1994. The relationship between recall and precision. *Journal of the American society for information science* 45, 1 (1994), 12–19.
- [29] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. 2019. nuScenes: A multi-modal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027* (2019).
- [30] Christopher Canel, Thomas Kim, Giulio Zhou, Conglong Li, Hyeontaek Lim, David Andersen, Michael Kaminsky, and Subramanya Dooloor. 2019. Scaling Video Analytics on Constrained Edge Nodes. *SysML* (2019).
- [31] Alexandra Carpentier, Remi Munos, and Andrés Antos. 2015. Adaptive strategy for stratified Monte Carlo sampling. *J. Mach. Learn. Res.* 16 (2015), 2231–2271.
- [32] Callie R Chappell and Tadashi Fukami. 2018. Nectar yeasts: a natural microcosm for ecology. *Yeast* 35, 6 (2018), 417–423.
- [33] Lixing Chen, Jie Xu, and Zhuo Lu. 2018. Contextual Combinatorial Multi-armed Bandits with Volatile Arms and Submodular Reward. In *NeurIPS*. 3247–3256.
- [34] Chiao-Lun Cheng. 2019. Training Data - Quantity is no Panacea. (2019). <https://scale.com/blog/training-data-quantity-is-no-panacea>
- [35] Sandeep Chinchali, Apoorva Sharma, James Harrison, Amine Elhafsi, Daniel Kang, Evgenya Pergament, Eyal Cidon, Sachin Katti, and Marco Pavone. 2019. Network offloading policies for cloud robotics: a learning-based approach. *arXiv preprint arXiv:1902.05703* (2019).
- [36] Eunsol Choi, Omer Levy, Yejin Choi, and Luke Zettlemoyer. 2018. Ultra-fine entity typing. *ACL* (2018).
- [37] François Chollet et al. 2015. Keras.

- [38] Pramod Chunduri, Jaeho Bang, Yao Lu, and Joy Arulraj. 2021. Zeus: Efficiently localizing actions in videos using reinforcement learning. *arXiv preprint arXiv:2104.06142* (2021).
- [39] Devin Coldewey. 2018. Uber in fatal crash detected pedestrian but had emergency braking disabled. <https://techcrunch.com/2018/05/24/uber-in-fatal-crash-detected-pedestrian-but-had-emergency-braking-disabled/>
- [40] Cody Coleman, Daniel Kang, Deepak Narayanan, Luigi Nardi, Tian Zhao, Jian Zhang, Peter Bailis, Kunle Olukotun, Chris Re, and Matei Zaharia. 2018. Analysis of DAWN Bench, a Time-to-Accuracy Machine Learning Performance Benchmark. *arXiv preprint arXiv:1806.01427* (2018).
- [41] Cody Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. 2017. DAWN Bench: An End-to-End Deep Learning Benchmark and Competition. *Training* 100, 101 (2017), 102.
- [42] Cody Coleman, Christopher Yeh, Stephen Mussmann, Baharan Mirzasoleiman, Peter Bailis, Percy Liang, Jure Leskovec, and Matei Zaharia. 2020. Selection via Proxy: Efficient Data Selection for Deep Learning. In *ICLR*.
- [43] Gordon V Cormack and Thomas R Lynam. 2005. TREC 2005 Spam Track Overview.. In *TREC*. 500–274.
- [44] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2009. *Introduction to algorithms*. MIT press.
- [45] Graham Cormode, Antonios Deligiannakis, Minos Garofalakis, and Andrew McGregor. 2009. Probabilistic histograms for probabilistic data. *PVLDB* 2, 1 (2009), 526–537.
- [46] Alex Davies. 2018. How do self-driving cars see? (And how do they see me?). <https://www.wired.com/story/the-know-it-alls-how-do-self-driving-cars-see/>
- [47] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *CVPR*. Ieee, 248–255.
- [48] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
- [49] Mehmet Emin Dönderler, Ediz Şaykol, Umut Arslan, Özgür Ulusoy, and Uğur Güdükbay. 2005. BilVideo: Design and implementation of a video database management system. *Multimedia Tools and Applications* (2005).

- [50] Brad Dwyer. 2020. A popular self-driving car dataset is missing labels for hundreds of pedestrians. <https://blog.roboflow.ai/self-driving-car-dataset-missing-pedestrians/>.
- [51] Bradley Efron and Robert J Tibshirani. 1994. *An introduction to the bootstrap*. CRC press.
- [52] Miles Efron. 2011. Information search and retrieval in microblogs. *Journal of the American Society for Information Science and Technology* 62, 6 (2011), 996–1008.
- [53] EHRA. 2010. 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 (2010), 2369–2429.
- [54] Steven Eliuk, Cameron Upright, Hars Vardhan, Stephen Walsh, and Trevor Gale. 2016. dMath: Distributed Linear Algebra for DL. *arXiv preprint arXiv:1611.07819* (2016).
- [55] Lawrence C Evans. 1998. Graduate studies in mathematics. In *Partial differential equations*. Am. Math. Soc.
- [56] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, et al. 2018. A configurable cloud-scale DNN processor for real-time AI. In *ISCA*. IEEE Press, 1–14.
- [57] T Gale, S Eliuk, and C Upright. 2017. High-Performance Data Loading and Augmentation for Deep Neural Network Training. In *GPU technology conference 2017*.
- [58] Edward Gan, Peter Bailis, and Moses Charikar. 2020. Coopstore: Optimizing precomputed summaries for aggregation. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2174–2187.
- [59] Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. 2002. Querying and mining data streams: you only get one look a tutorial. In *SIGMOD*. 635–635.
- [60] Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. 2018. Detectron. <https://github.com/facebookresearch/detectron>.
- [61] Herman Heine Goldstine, John Von Neumann, and John Von Neumann. 1947. Planning and coding of problems for an electronic computing instrument. (1947).
- [62] Teofilo F Gonzalez. 1985. Clustering to minimize the maximum intercluster distance. *Theoretical computer science* 38 (1985), 293–306.
- [63] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

- [64] Michael Gordon and Manfred Kochen. 1989. Recall-precision trade-off: A derivation. *Journal of the American Society for Information Science* 40, 3 (1989), 145–151.
- [65] Ed Greengrass. 2000. Information retrieval: A survey. (2000).
- [66] Sudipto Guha and Boulos Harb. 2005. Wavelet synopsis for data streams: minimizing non-euclidean error. In *KDD*. 88–97.
- [67] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. 2017. On calibration of modern neural networks. In *International conference on machine learning*. PMLR, 1321–1330.
- [68] Peter J Haas and Joseph M Hellerstein. 1999. Ripple joins for online aggregation. *ACM SIGMOD Record* 28, 2 (1999), 287–298.
- [69] John Michael Hammersley and David Christopher Handscomb. 1964. General principles of the Monte Carlo method. In *Monte Carlo Methods*. Springer, 50–75.
- [70] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *arXiv preprint arXiv:1510.00149* (2015).
- [71] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. 2016. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *MobiSys*. ACM, 123–136.
- [72] Brandon Haynes, Maureen Daum, Dong He, Amrita Mazumdar, Magdalena Balazinska, Alvin Cheung, and Luis Ceze. 2021. VSS: A storage system for video analytics. In *SIGMOD*. 685–696.
- [73] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask r-cnn. In *ICCV*. IEEE, 2980–2988.
- [74] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*. 770–778.
- [75] Joseph M Hellerstein, Peter J Haas, and Helen J Wang. 1997. Online aggregation. In *Acm Sigmod Record*, Vol. 26. ACM, 171–182.
- [76] Dan Hendrycks and Thomas G Dietterich. 2018. Benchmarking neural network robustness to common corruptions and surface variations. *arXiv preprint arXiv:1807.01697* (2018).
- [77] Samitha Herath, Mehrtash Harandi, and Fatih Porikli. 2017. Going deeper into action recognition: A survey. *Image and vision computing* 60 (2017), 4–21.

- [78] Geoffrey Hinton and Tijmen Tieleman. 2012. *Lecture 6.5 - RMSProp*. Technical Report.
- [79] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).
- [80] James Hong, Will Crichton, Haotian Zhang, Daniel Y Fu, Jacob Ritchie, Jeremy Barenholtz, Ben Hannel, Xinwei Yao, Michaela Murray, Geraldine Moriba, et al. 2020. Analyzing Who and What Appears in a Decade of US Cable TV News. *arXiv preprint arXiv:2008.06007* (2020).
- [81] Eduard Hovy, Mitch Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. 2006. OntoNotes: the 90% solution. In *NAACL*. 57–60.
- [82] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).
- [83] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Paramvir Bahl, Matthai Philipose, Phillip B Gibbons, and Onur Mutlu. 2018. Focus: Querying Large Video Datasets with Low Latency and Low Cost. *OSDI* (2018).
- [84] Clayton Hutto and Eric Gilbert. 2014. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Proceedings of the International AAAI Conference on Web and Social Media*, Vol. 8.
- [85] Stratos Idreos, Martin L Kersten, Stefan Manegold, et al. 2007. Database Cracking.. In *CIDR*, Vol. 7. 68–78.
- [86] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: scalable adaptation of video analytics. In *SIGCOMM*. ACM, 253–266.
- [87] Rie Johnson and Tong Zhang. 2013. Accelerating stochastic gradient descent using predictive variance reduction. In *NIPS*. 315–323.
- [88] Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. 2020. Spanbert: Improving pre-training by representing and predicting spans. *TACL* 8 (2020), 64–77.
- [89] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *ISCA*. IEEE, 1–12.
- [90] Sandeep Juneja and Perwez Shahabuddin. 2006. Rare-event simulation techniques: an introduction and recent advances. *Handbooks in operations research and management science* 13 (2006), 291–350.

- [91] Daniel Justus, John Brennan, Stephen Bonner, and Andrew Stephen McGough. 2018. Predicting the computational cost of deep learning models. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 3873–3882.
- [92] Daniel Kang, Nikos Arechiga, Sudeep Pillai, Peter Bailis, and Matei Zaharia. 2022. Finding Label and Model Errors in Perception Data with Learned Observation Assertions. *SIGMOD* (2022).
- [93] Daniel Kang, Peter Bailis, and Matei Zaharia. 2019. BlazeIt: Optimizing Declarative Aggregation and Limit Queries for Neural Network-Based Video Analytics. *PVLDB* (2019).
- [94] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: optimizing neural network queries over video at scale. *PVLDB* 10, 11 (2017), 1586–1597.
- [95] Daniel Kang, Edward Gan, Peter Bailis, Tatsunori Hashimoto, and Matei Zaharia. 2020. Approximate Selection with Guarantees using Proxies. *PVLDB* (2020).
- [96] Daniel Kang, John Guibas, Peter Bailis, Tatsunori Hashimoto, Yi Sun, and Matei Zaharia. 2021. Accelerating Approximate Aggregation Queries with Expensive Predicates. *PVLDB* (2021).
- [97] Daniel Kang, John Guibas, Peter Bailis, Tatsunori Hashimoto, and Matei Zaharia. 2022. Task-agnostic Indexes for Deep Learning-based Queries over. *SIGMOD* (2022).
- [98] Daniel Kang, John Guibas, Peter Bailis, Yi Sun, Tatsunori Hashimoto, and Matei Zaharia. 2021. Proof: Accelerating Approximate Aggregation Queries with Expensive Predicates. *arXiv preprint arXiv:2107.12525* (2021).
- [99] Daniel Kang, Ankit Mathur, Teja Veeramacheneni, Peter Bailis, and Matei Zaharia. 2021. Jointly Optimizing Preprocessing and Inference for DNN-based Visual Analytics. *PVLDB* (2021).
- [100] Daniel Kang, Ankit Mathur, Teja Veeramacheneni, Peter Bailis, and Matei Zaharia. 2021. Jointly optimizing preprocessing and inference for DNN-based visual analytics. *PVLDB* (2021).
- [101] Daniel Kang, Deepti Raghavan, Peter Bailis, and Matei Zaharia. 2020. Model Assertions for Monitoring and Improving ML Model. *MLSys* (2020).
- [102] Andrej Kaparthy. 2018. Building the Software 2.0 Stack. (2018).
- [103] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 97–117.

- [104] R. Kesten, M. Usman, J. Houston, T. Pandya, K. Nadhamuni, A. Ferreira, M. Yuan, B. Low, A. Jain, P. Ondruska, S. Omari, S. Shah, A. Kulkarni, A. Kazakova, C. Tao, L. Platinsky, W. Jiang, and V. Shet. 2019. Lyft Level 5 Perception Dataset 2020. <https://level5.lyft.com/dataset/>.
- [105] Omar Khattab, Christopher Potts, and Matei Zaharia. 2021. Relevance-guided supervision for openqa with colbert. *TACL* 9 (2021), 929–944.
- [106] Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *SIGIR*. 39–48.
- [107] Leslie Kish. 1965. *Survey sampling*. Number 04; HN29, K5.
- [108] Qiuqiang Kong, Yin Cao, Turab Iqbal, Yuxuan Wang, Wenwu Wang, and Mark D Plumbley. 2020. Panns: Large-scale pretrained audio neural networks for audio pattern recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 28 (2020), 2880–2894.
- [109] P. Kraft, Daniel Kang, D. Narayanan, S. Palkar, P. Bailis, and M. Zaharia. 2019. Willump: A statistically-aware end-to-end optimizer for machine learning inference. *MLSys* (2019).
- [110] Alex Krizhevsky et al. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*.
- [111] Frank R Kschischang, Brendan J Frey, and H-A Loeliger. 2001. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory* 47, 2 (2001), 498–519.
- [112] Tony CT Kuo and Arbee LP Chen. 1996. A content-based query language for video databases. In *ICMCS*. IEEE, 209–214.
- [113] Tony CT Kuo and Arbee LP Chen. 2000. Content-based query processing for video databases. *IJDTA* 2, 1 (2000), 1–13.
- [114] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. 2019. PointPillars: Fast encoders for object detection from point clouds. In *CVPR*. 12697–12705.
- [115] Thi-Lan Le, Monique Thonnat, Alain Boucher, and François Brémont. 2008. A query language combining object features and semantic events for surveillance video retrieval. In *MMM*. Springer.
- [116] Timothy Lee. 2018. 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/>.

- [117] John Z Li, M Tamer Ozsü, Duane Szafron, and Vincent Oria. 1997. MOQL: A multimedia object query language. In *MIPR*. 19–28.
- [118] Kaiyu Li and Guoliang Li. 2018. Approximate query processing: What is new and where to go? *Data Science and Engineering* 3, 4 (2018), 379–397.
- [119] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *ECCV*. Springer, 740–755.
- [120] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. Ssd: Single shot multibox detector. In *ECCV*. Springer, 21–37.
- [121] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. 2015. Deep Learning Face Attributes in the Wild. In *Proceedings of International Conference on Computer Vision (ICCV)*.
- [122] Chenglang Lu, Mingyong Liu, and Zongda Wu. 2015. SVQL: A sql extended query language for video databases. *IJDTA* (2015).
- [123] Tyler Lu, Dávid Pál, and Martin Pál. 2010. Contextual multi-armed bandits. In *Proceedings of the Thirteenth international conference on Artificial Intelligence and Statistics*. 485–492.
- [124] Yao Lu, Aakanksha Chowdhery, Srikanth Kandula, and Surajit Chaudhuri. 2018. Accelerating Machine Learning Inference with Probabilistic Predicates. In *SIGMOD*. ACM, 1493–1508.
- [125] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press, USA.
- [126] Valerie N Martin, Robert N Schaeffer, and Tadashi Fukami. 2022. Potential effects of nectar microbes on pollinator health. *Philosophical Transactions of the Royal Society B* 377, 1853 (2022), 20210155.
- [127] Peter Mattson, Christine Cheng, Cody Coleman, Greg Diamos, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, et al. 2019. Mlperf training benchmark. *arXiv preprint arXiv:1910.01500* (2019).
- [128] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In *AFNLP*. Association for Computational Linguistics, 1003–1011.
- [129] Volodymyr Mnih, Csaba Szepesvári, and Jean-Yves Audibert. 2008. Empirical bernstein stopping. In *Proceedings of the 25th international conference on Machine learning*. ACM, 672–679.

- [130] Guido Moerkotte, Thomas Neumann, and Gabriele Steidl. 2009. Preventing bad plans by bounding the impact of cardinality estimation errors. *PVLDB* 2, 1 (2009), 982–993.
- [131] Ben Munson. 2018. Video will account for 82% of all internet traffic by 2022, Cisco says. (2018). <https://www.fiercevideo.com/video/video-will-account-for-82-all-internet-traffic-by-2022-cisco-says>
- [132] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *EMNLP-IJCNLP*. 188–197.
- [133] NTSB. 2019. Vehicle Automation Report, HWY18MH010. <https://dms.nts.gov/public/62500-62999/62978/629713.pdf>
- [134] NVIDIA. 2020. NVIDIA T4 Tensor Core GPU for AI Inference. <https://www.nvidia.com/en-us/data-center/tesla-t4/>
- [135] Harry Nyquist. 1928. Certain topics in telegraph transmission theory. *Transactions of the American Institute of Electrical Engineers* 47, 2 (1928), 617–644.
- [136] Augustus Odena, Catherine Olsson, David Andersen, and Ian Goodfellow. 2019. Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. In *ICML*. 4901–4911.
- [137] Art Owen and Yi Zhou. 2000. Safe and Effective Importance Sampling. *J. Amer. Statist. Assoc.* 95, 449 (2000), 135–143.
- [138] Shoumik Palkar, James J Thomas, Anil Shanbhag, Deepak Narayanan, Holger Pirk, Malte Schwarzkopf, Saman Amarasinghe, Matei Zaharia, and Stanford InfoLab. 2017. Weld: A common runtime for high performance data analytics. In *CIDR*.
- [139] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. (2017).
- [140] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated white-box testing of deep learning systems. In *OSDI*. ACM, 1–18.
- [141] William B Pennebaker and Joan L Mitchell. 1992. *JPEG: Still image data compression standard*. Springer Science & Business Media.
- [142] Gregory Piatetsky-Shapiro and Charles Connell. 1984. Accurate estimation of the number of tuples satisfying a condition. *SIGMOD* (1984).
- [143] Neoklis Polyzotis, Martin Zinkevich, Sudip Roy, Eric Breck, and Steven Whang. 2019. Data Validation for Machine Learning. *SysML* (2019).

- [144] Viswanath Poosala, Peter J Haas, Yannis E Ioannidis, and Eugene J Shekita. 1996. Improved histograms for selectivity estimation of range predicates. *ACM Sigmod Record* 25, 2 (1996), 294–305.
- [145] Filip Radlinski, Robert Kleinberg, and Thorsten Joachims. 2008. Learning diverse rankings with multi-armed bandits. In *ICML*. ACM, 784–791.
- [146] Pranav Rajpurkar, Awni Y Hannun, Masoumeh Haghpanahi, Codie Bourn, and Andrew Y Ng. 2019. Cardiologist-level arrhythmia detection with convolutional neural networks. *Nature Medicine* (2019).
- [147] Alex Ratner, Stephen Bach, Paroma Varma, and Chris Ré. 2017. Weak Supervision: The New Programming Paradigm for Machine Learning. <https://dawn.cs.stanford.edu/2017/07/16/weak-supervision/>
- [148] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. 2019. Mlperf inference benchmark. *arXiv preprint arXiv:1911.02549* (2019).
- [149] Joseph Redmon and Ali Farhadi. 2017. YOLO9000: better, faster, stronger. *arXiv preprint* (2017).
- [150] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*.
- [151] Cedric Renggli, Bojan Karlaš, Bolin Ding, Feng Liu, Kevin Schawinski, Wentao Wu, and Ce Zhang. 2019. Continuous Integration of Machine Learning Models with ease.ml/ci: Towards a Rigorous Yet Practical Treatment. *SysML* (2019).
- [152] Christian P Robert. 2004. *Monte carlo methods*. Wiley Online Library.
- [153] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision* 115, 3 (2015), 211–252.
- [154] Priscilla A San Juan, J Nicholas Hendershot, Gretchen C Daily, and Tadashi Fukami. 2020. Land-use change has host-specific influences on avian gut microbiomes. *The ISME Journal* 14, 1 (2020), 318–321.
- [155] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*. 4510–4520.

- [156] Ozan Sener and Silvio Savarese. 2017. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489* (2017).
- [157] Sefik Ilkin Serengil and Alper Ozpinar. 2020. LightFace: A Hybrid Deep Face Recognition Framework. In *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*. IEEE, 23–27. <https://doi.org/10.1109/ASYU50717.2020.9259802>
- [158] Burr Settles. 2009. *Active learning literature survey*. Technical Report. University of Wisconsin-Madison Department of Computer Sciences.
- [159] Haichen Shen, Seungyeop Han, Matthai Philipose, and Arvind Krishnamurthy. 2017. Fast video classification via adaptive cascading of deep models. In *CVPR*. 3646–3654.
- [160] Vinay Shet. 2019. Lyft Level 5 Self-Driving Perception Dataset Competition Now Open. <https://medium.com/wovenplanetlevel5/lyft-level-5-self-driving-dataset-competition-now-open-97493e9f154a>. (2019).
- [161] Gary J Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. 2012. Overview of the high efficiency video coding (HEVC) standard. *IEEE Transactions on circuits and systems for video technology* 22, 12 (2012), 1649–1668.
- [162] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. 2017. Revisiting unreasonable effectiveness of data in deep learning era. In *ICCV*. 843–852.
- [163] Sahaana Suri, Raghuveer Chanda, Neslihan Bulut, Pradyumna Narayana, Yemao Zeng, Peter Bailis, Sugato Basu, Girija Narlikar, Christopher Ré, and Abishek Sethi. 2020. Leveraging organizational resources to adapt models to new data modalities. *arXiv preprint arXiv:2008.09983* (2020).
- [164] Mingxing Tan and Quoc V Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946* (2019).
- [165] Mingxing Tan, Ruoming Pang, and Quoc V Le. 2019. Efficientdet: Scalable and efficient object detection. *arXiv preprint arXiv:1911.09070* (2019).
- [166] David Taubman and Michael Marcellin. 2012. *JPEG2000 image compression fundamentals, standards and practice: image compression fundamentals, standards and practice*. Vol. 642. Springer Science & Business Media.
- [167] Michel Tokic and Günther Palm. 2011. Value-difference based exploration: adaptive control between epsilon-greedy and softmax. In *Annual Conference on Artificial Intelligence*. Springer, 335–346.

- [168] Alan Turing. 1949. Checking a large routine. In *Report on a Conference on High Speed Automatic Calculating machines*. Cambridge University Mathematics Lab, 67–69.
- [169] Paul Viola and Michael Jones. 2001. Rapid object detection using a boosted cascade of simple features. In *CVPR*, Vol. 1. IEEE, I–I.
- [170] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. 2011. The caltech-ucsd birds-200-2011 dataset. (2011).
- [171] Daisuke Wakabayashi. 2018. Self-Driving Uber Car Kills Pedestrian in Arizona, Where Robots Roam. <https://www.nytimes.com/2018/03/19/technology/uber-driverless-fatality.html>.
- [172] Gregory K Wallace. 1992. The JPEG still picture compression standard. *IEEE transactions on consumer electronics* 38, 1 (1992), xviii–xxxiv.
- [173] Ulla Wandinger. 2005. Introduction to lidar. In *Lidar*. Springer, 1–18.
- [174] Larry Wasserman. 2013. *All of statistics: a concise course in statistical inference*. Springer Science & Business Media.
- [175] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682* (2022).
- [176] Kilian Q Weinberger and Lawrence K Saul. 2009. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research* 10, 2 (2009).
- [177] Thomas Wiegand, Gary J Sullivan, Gisle Bjontegaard, and Ajay Luthra. 2003. Overview of the H. 264/AVC video coding standard. *IEEE Transactions on circuits and systems for video technology* 13, 7 (2003), 560–576.
- [178] Hao Wu. 2019. Low Precision Inference on GPU. <https://developer.download.nvidia.com/video/gputechconf/gtc/2019/presentation/s9659-inference-at-reduced-precision-on-gpus.pdf>
- [179] Weiming Xiang, Patrick Musau, Ayana A Wild, Diego Manzananas Lopez, Nathaniel Hamilton, Xiaodong Yang, Joel Rosenfeld, and Taylor T Johnson. 2018. Verification for machine learning, autonomy, and neural networks survey. *arXiv preprint arXiv:1810.01989* (2018).
- [180] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. 2017. Aggregated residual transformations for deep neural networks. In *CVPR*. 1492–1500.

- [181] Tiantu Xu, Luis Materon Botelho, and Felix Xiaozhu Lin. 2019. VStore: A Data Store for Analytics on Large Videos. In *EuroSys*. ACM, 16.
- [182] Yan Yan, Yuxing Mao, and Bo Li. 2018. Second: Sparsely embedded convolutional detection. *Sensors* 18, 10 (2018), 3337.
- [183] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. 2017. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In *NSDI*, Vol. 9. 1.
- [184] Jie M Zhang, Mark Harman, Lei Ma, and Yang Liu. 2020. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering* (2020).
- [185] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. 2016. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters* 23, 10 (2016), 1499–1503.
- [186] Lei Zhang, Shuai Wang, and Bing Liu. 2018. Deep learning for sentiment analysis: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8, 4 (2018), e1253.
- [187] Yuhao Zhang, Victor Zhong, Danqi Chen, Gabor Angeli, and Christopher D. Manning. 2017. Position-aware Attention and Supervised Data Improve Slot Filling. In *EMNLP 2017*. 35–45.
- [188] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *CoRR* abs/1709.00103 (2017).
- [189] Benjin Zhu, Zhengkai Jiang, Xiangxin Zhou, Zeming Li, and Gang Yu. 2019. Class-balanced Grouping and Sampling for Point Cloud 3D Object Detection. *arXiv preprint arXiv:1908.09492* (2019).
- [190] Xizhou Zhu, Yujie Wang, Jifeng Dai, Lu Yuan, and Yichen Wei. 2017. Flow-guided feature aggregation for video object detection. *arXiv preprint arXiv:1703.10025* (2017).

ProQuest Number: 30306172

INFORMATION TO ALL USERS

The quality and completeness of this reproduction is dependent on the quality and completeness of the copy made available to ProQuest.



Distributed by ProQuest LLC (2023).

Copyright of the Dissertation is held by the Author unless otherwise noted.

This work may be used in accordance with the terms of the Creative Commons license or other rights statement, as indicated in the copyright statement or in the metadata associated with this work. Unless otherwise specified in the copyright statement or the metadata, all rights are reserved by the copyright holder.

This work is protected against unauthorized copying under Title 17, United States Code and other applicable copyright laws.

Microform Edition where available © ProQuest LLC. No reproduction or digitization of the Microform Edition is authorized without permission of ProQuest LLC.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346 USA