conference

proceedings

2020 USENIX Conference on Operational Machine Learning (OpML '20)

July 28–August 7, 2020

Sponsored by

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

ISBN 978-1-939133-15-1

Proceedings of the 2020 USENIX Conference on Operational Machine Learning (OpML '20)

July 28–August 7, 2020

OpML '20 Sponsors

Bronze Sponsor

Open Access Sponsor

USENIX Supporters

USENIX Patrons

Bloomberg • Facebook • Google
Microsoft • NetApp

USENIX Benefactors

Amazon • Oracle • Thinkst Canary
Two Sigma • VMware

USENIX Partners

Top10VPN

Open Access Publishing Partner

PeerJ

USENIX Association

Proceedings of the 2020 USENIX Conference on Operational Machine Learning (OpML '20)

July 28–August 7, 2020

ISBN 978-1-939133-15-1

Conference Organizers

Program Co-Chairs
Nisha Talagala, *Pyxeda AI*
Joel Young, *LinkedIn*

Program Committee
Fei Chen, *LinkedIn*
Sindhu Ghanta, *ParallelM*
Kun Liu, *Amazon*
Prassana Padmanaban, *Netflix*
Neoklis Polyzotis, *Google*
Suresh Raman, *Intuit*
Bharath Ramsundar, *Stealth Mode*
Marius Seritan, *Argo AI*
Faisal Siddiqi, *Netflix*
Swaminathan Sundararaman, *Pyxeda*
Eno Thereska, *Amazon*
Boris Tvaroska, *Lenovo*
Todd Underwood, *Google*
Sandeep Uttamchandani, *Intuit*
Manasi Vartak, *Verta AI*
Jesse Ward, *LinkedIn*

Steering Committee
Nitin Agrawal, *Samsung*
Eli Collins, *Accel Partners/Samba Nova*
Casey Henderson, *USENIX Association*
Robert Ober, *Nvidia*
Bharath Ramsundar, *Stealth Mode*
Jairam Ranganathan, *Uber*
D. Sculley, *Google*
Tal Shaked, *Google*
Swaminathan Sundararaman
Nisha Talagala, *Pyxeda AI*
Sandeep Uttamchandani, *Intuit*
Joel Young, *LinkedIn*

Message from the OpML '20 Program Co-Chairs

Welcome to OpML 2020!

We are very excited to launch the second annual USENIX Conference on Operational Machine Learning (OpML).

Machine Learning is interlaced throughout our society bringing new challenges in deploying, managing, and optimizing these systems in production while maintaining fairness and privacy. We started OpML to provide a forum where practitioners, researchers, industry, and academia can gather to present, evaluate, and debate the problems, best practices, and latest cutting-edge technologies in this critical emerging field. Managing the ML production lifecycle is a necessity for wide-scale adoption and deployment of machine learning and deep learning across industries and for businesses to benefit from the core ML algorithms and research advances.

The conference received strong interest this year, with 54 submissions spanning both academia and industry. Thanks to the hard work of our Program Committee, we created an exciting program with 26 technical presentations with eight Ask-Me-Anything sessions with the authors. Each presentation and paper submission was evaluated by 3–5 PC members, with the final decisions made during a half-day online PC meeting.

This has been an exceptionally challenging time for our authors and speakers, for our program committee, and for the world. Given safety concerns and travel restrictions, this year we are experimenting with a new format with eight Ask-Me-Anything sessions hosted on Slack. Each presenter has prepared both short and long form videos and will be online to answer questions during their session. Furthermore, the conference is free to attend for all participants!

We would like to thank the many people whose hard work made this conference possible. First and foremost, we would like to thank the authors for their incredible work and the submissions to OpML '20. Thanks to the Program Committee for their hard work in reviews and spirited discussion (Bharath Ramsundar, Boris Tvaroska, Eno Thereska, Faisal Siddiqi, Fei Chen, Jesse Ward, Kun Liu, Manasi Vartak, Marius Seritan, Neoklis Polyzotis, Prasanna Padmanabhan, Sandeep Uttamchandani, Sindhu Ghanta, Suresh Raman, Swami Sundaraman, and Todd Underwood).

We would also like to thank the members of the steering committee for their guidance throughout the process (Bharath Ramsundar, Sandeep Uttamchandani, Swaminathan (Swami) Sundaraman, Casey Henderson, D. Sculley, Eli Collins, Jairam Ranganathan, Nitin Agrawal, Robert Ober, and Tal Shaked).

Finally, we would like to thank Casey Henderson and Kurt Andersen of USENIX for their tremendous help and insight as we worked on this new conference, and all of the USENIX staff for their extraordinary level of support throughout the process.

We hope you enjoy the conference and proceedings!

Best Regards,
Nisha Talagala, *Pyxeda AI*
Joel Young, *LinkedIn*

2020 USENIX Conference on Operational Machine Learning (OpML '20)

July 28–August 7, 2020

Tuesday, July 28

Session 1: Deep Learning

Wednesday, July 29

Session 2: Model Life Cycle

Thursday, July 30

Session 3: Features, Explainability, and Analytics

Friday, July 31

Session 4: Algorithms

Tuesday, August 4

Session 5: Model Deployment Strategies

Wednesday, August 5

Session 6: Applications and Experiences

DLSpec: A Deep Learning Task Exchange Specification

Abdul Dakkak[1*], Cheng Li[1*], Jinjun Xiong[2], Wen-mei Hwu[1]
[1]*University of Illinois Urbana-Champaign,* [2]*IBM T. J. Watson Research Center*
{dakkak, cli99, w-hwu}@illinois.edu, jinjun@us.ibm.com

Abstract

Deep Learning (DL) innovations are being introduced at a rapid pace. However, the current lack of standard specification of DL tasks makes sharing, running, reproducing, and comparing these innovations difficult. To address this problem, we propose DLSpec, a model-, dataset-, software-, and hardware-agnostic DL specification that captures the different aspects of DL tasks. DLSpec has been tested by specifying and running hundreds of DL tasks.

1 Introduction

The past few years have seen a fast growth of Deep Learning (DL) innovations such as datasets, models, frameworks, software, and hardware. The current practice of publishing these DL innovations involves developing ad-hoc scripts and writing textual documentation to describe the execution process of *DL tasks* (e.g., model training or inference). This requires a lot of effort and makes sharing and running DL tasks difficult. Moreover, it is often hard to reproduce the reported accuracy or performance results and have a consistent comparison across DL tasks. This is a known [7, 8] "pain point" within the DL community. Having an exchange specification to describe DL tasks would be a first step to remedy this and ease the adoption of DL innovations.

Previous work included curation of DL tasks in framework model zoos [3, 6, 12–14, 18], developing model catalogs that can be used through a cloud provider's API [1, 2, 5], or introducing MLOps specifications [4, 19, 20]. However, these work either use ad-hoc techniques for different DL tasks or are tied to a specific hardware or software stack.

We propose DLSpec, a DL artifact exchange specification with clearly defined model, data, software, and hardware aspects. DLSpec's design is based on a few key principles (Section 2). DLSpec is model-, dataset-, software-, and hardware-agnostic and aims to work with runtimes built using existing MLOp tools. We further develop a DLSpec runtime to support DLSpec's use for DL inference tasks in the context of benchmarking [9].

*The two authors contributed equally to this paper.

2 Design Principles

While the bottom line of a specification design is to ensure the usability and reproducibility of DL tasks, the following principles are considered to increase DLSpec's applicability:

Minimal — To increase the transparency and ease the creation, the specification should contain only the essential information to use a task and reproduce its reported outcome.

Program-/human-readable — To make it possible to develop a runtime that executes DLSpec, the specification should be readable by a program. To allow a user to understand what the task does and repurpose it (e.g., use a different HW/SW stack), the specification should be easy to introspect.

Maximum expressiveness — While DL training and inference tasks can share many common software and hardware setups, there are differences when specifying their resources, parameters, inputs/outputs, metrics, etc. The specification must be general to be used for both training and inference tasks.

Decoupling DL task description — A DL task is described from the aspects of model, data, software, and hardware through their respective manifest files. Such decoupling increases the reuse of manifests and enables the portability of DL tasks across datasets, software, and hardware stacks. This further enables one to easily compare different DL offerings by varying one of the four aspects.

Splitting the DL task pipeline stages — We demarcate the stages of a DL task into pre-processing, run, and post-processing stages. This enables consistent comparison and simplifies accuracy and performance debugging. For example, to debug accuracy, one can modify the pre- or post-processing step and observe the accuracy; and to debug performance, one can surround the run stage with the measurement code. This demarcation is consistent with existing best practices [11, 17].

Avoiding serializing intermediate data into files — A naive way to transfer data between stages of a DL task is to use files. In this way, each stage reads a file containing input data and writes to a file with its output data. This approach can be impractical since it introduces high serializing/deserializing overhead. Moreover, this technique would not be able to support DL tasks that use streaming data.

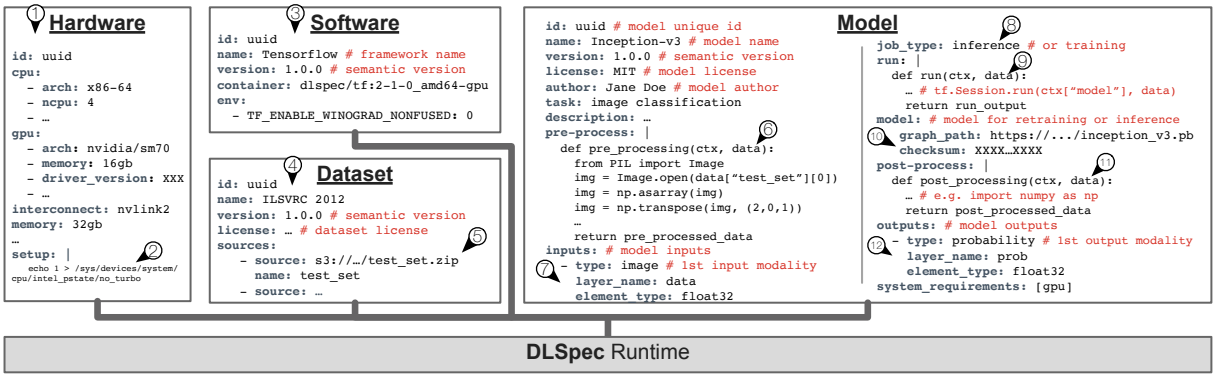


Figure 1: An example DLSpec that consists of a hardware, software, dataset and model manifest.

3 DLSpec Design

A DLSpec consists of four manifest files and a reference log file. All manifests are versioned [15] and have an ID (i.e., can be referenced). The four manifests (shown in Figure 1) are:

- **Hardware**: defines the hardware requirements for a DL task. The parameters form a series of hardware constraints that must be satisfied to run and reproduce a task.
- **Software**: defines the software environment for a DL task. All executions occur within the specified container.
- **Dataset**: defines the training, validation, or test dataset.
- **Model**: defines the logic (or code) to run a DL task and the required artifact sources.

The reference log is provided by the specification author for others to refer to. The reference log contains the IDs of the manifests used to create it, achieved accuracy/performance on DL task, expected outputs, and author-specified information (e.g. hyper-parameters used in training).

We highlight the key details of DLSpec:

Containers — A software manifest specifies the container to use for a DL task, as well as any configuration parameters that must be set (e.g., framework environment variables). The framework or other libraries information are listed for ease of inspection and management.

Hardware configuration — While containers provide a standard way of specifying the software stack, a user cannot specify some hardware settings within a container. E.g., it is not possible to turn off Intel's turbo-boosting (Figure 1②) within a container. Thus, DLSpec specifies hardware configurations in the hardware manifest to allow the runtime to set them outside the container environment.

Pre-processing, run, and post-processing stages — The pre-/post-processing and run stages are defined via Python functions embedded within the manifest. We do this because a DLSpec runtime can use the Python sub-interpreter [16] to execute the Python code within the process, thus avoiding using intermediate files (see Section 2). Using Python functions also allows for great flexibility; e.g., the Python function can download and run Bash and R scripts or download, compile, and some C++ code. The signature of the DLSpec Python functions is `fun(ctx, data)` where `ctx` is a hash table that includes manifest information (such as the types of inputs) accepted by the model. The second argument, `data`, is the output of the previous step in the dataset→pre-processing→run→post-processing pipeline. In Figure 1, for example, the pre-processing stage's ⑥ `data` is the list of file paths of the input dataset (ImageNet test set in this case).

Artifact resources — DL artifacts used in a DL task are specified as remote resources within DLSpec. The remote resource can be hosted on an FTP, HTTP, or file server (e.g., AWS S3 or Zenodo) and have a checksum which is used to verify the download.

4 DLSpec Runtime

While a DL practitioner can run a DL task by manually following the setup described in the manifests, here we describe how a runtime (i.e., an MLOps tool) can use the DLSpec manifests shown in Figure 1.

A DLSpec runtime consumes the four manifests and selects the ① hardware to use, and runs any ② setup code specified (outside the container). A ③ container is launched using the image specified, and the ④ dataset is downloaded into the container using the ⑤ URLs provided. The ⑥ dataset file paths are passed to the pre-processing function and its outputs are then processed to match the ⑦ model's input parameters. The ⑨ DL task is run. In the case of ⑧ inference, this causes the ⑩ model to be downloaded into the container. The results from the run are then ⑪ post-processed using the ⑫ data specified by the model outputs.

We tested DLSpec in the context of inference benchmarking and implemented a runtime for it [9, 10]. We collected over 300 popular models and created reusable manifests for each. We created software manifests for each major framework (Caffe, Caffe2, CNTK, MXNet, PyTorch, TensorFlow, TensorFlow Lite, and TensorRT), dataset manifest (ImageNet, COCO, Pascal, CIFAR, etc.), and then wrote hardware specs for X86, ARM, and PowerPC. We tested our design and showed that it enables consistent and reproducible evaluation of DL tasks at scale.

5 Conclusion

An exchange specification, such as DLSpec, enables a streamlined way to share, reproduce, and compare DL tasks. DLSpec takes the first step in defining a DL task for both training and inference and captures the different aspects of DL model reproducibility. We are actively working on refining the specifications as new DL tasks are introduced.

References

[1] Amazon SageMaker. https://aws.amazon.com/sagemaker, 2020. Accessed: 2020-02-20.

[2] MLOps: Model management, deployment and monitoring with Azure Machine Learning. https://docs.microsoft.com/en-us/azure/machine-learning/concept-model-management-and-deployment, 2020. Accessed: 2020-02-20.

[3] Caffe2 model zoo. https://caffe2.ai/docs/zoo.html, 2020. Accessed: 2020-02-20.

[4] Grigori Fursin, Herve Guillou, and Nicolas Essayan. CodeReef: an open platform for portable MLOps, reusable automation actions and reproducible benchmarking. *arXiv preprint arXiv:2001.07935*, 2020.

[5] Architecture for MLOps using TFX, Kubeflow Pipelines, and Cloud Build. https://bit.ly/39P6JFk, 2020. Accessed: 2020-02-20.

[6] GluonCV. https://gluon-cv.mxnet.io/, 2020. Accessed: 2020-02-20.

[7] Odd Erik Gundersen and Sigbjørn Kjensmo. State of the art: Reproducibility in artificial intelligence. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[8] Matthew Hutson. Artificial intelligence faces reproducibility crisis, 2018.

[9] Cheng Li, Abdul Dakkak, Jinjun Xiong, and Wen-mei Hwu. The Design and Implementation of a Scalable DL Benchmarking Platform. *arXiv preprint arXiv:1911.08031*, 2019.

[10] Cheng Li, Abdul Dakkak, Jinjun Xiong, Wei Wei, Lingjie Xu, and Wen-Mei Hwu. XSP: Across-Stack Profiling and Analysis of Machine Learning Models on GPUs. IEEE, May 2020. The 34th IEEE International Parallel & Distributed Processing Symposium (IPDPS'20).

[11] Peter Mattson, Christine Cheng, Cody Coleman, Greg Diamos, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, et al. Mlperf training benchmark. *arXiv preprint arXiv:1910.01500*, 2019.

[12] Modelhub. http://modelhub.ai, 2020. Accessed: 2020-02-20.

[13] ModelZoo. https://modelzoo.co, 2020. Accessed: 2020-02-20.

[14] ONNX Model Zoo. https://github.com/onnx/models, 2020. Accessed: 2020-02-20.

[15] Tom Preston-Werner. Semantic versioning 2.0.0. https://www.semver.org, 2019.

[16] Initialization, Finalization, and Threads. https://docs.python.org/3.6/c-api/init.html#sub-interpreter-support, 2019. Accessed: 2020-02-20.

[17] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. Mlperf inference benchmark. *arXiv preprint arXiv:1911.02549*, 2019.

[18] TensorFlow Hub. https://www.tensorflow.org/hub, 2020. Accessed: 2020-02-20.

[19] Manasi Vartak, Harihar Subramanyam, Wei-En Lee, Srinidhi Viswanathan, Saadiyah Husnoo, Samuel Madden, and Matei Zaharia. ModelDB: a system for machine learning model management. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, pages 1–3, 2016.

[20] Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, et al. Accelerating the machine learning lifecycle with mlflow. *Data Engineering*, page 39, 2018.

Finding Bottleneck in Machine Learning Model Life Cycle

Chandra Mohan Meena
WalmartLabs

Sarwesh Suman
WalmartLabs

Vijay Agneeswaran
WalmartLabs

Abstract

Our data scientists are adept in using machine learning algorithms and building model out of it, and they are at ease with their local machine to do them. But, when it comes to building the same model from the platform, they find it slightly challenging and need assistance from the platform team. Based on the survey results, the major challenge was platform complexity, but it is hard to deduce actionable items or accurate details to make the system simple. The complexity feedback was very generic, so we decided to break it down into two logical challenges: Education & Training and Simplicity-of-Platform. We have developed a system to find these two challenges in our platform, which we call an Analyzer. In this paper, we explain how it was built and it's impact on the evolution of our machine learning platform. Our work aims to address these challenges and provide guidelines on how to empower machine learning platform team to know the data scientist's bottleneck in building model.

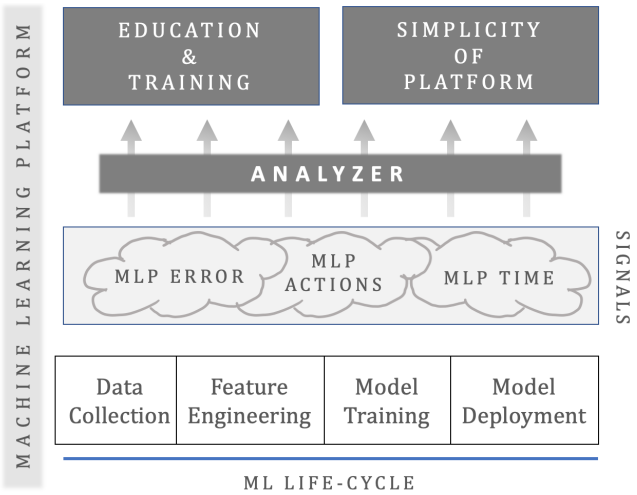


Figure 1: System Architecture

1 Introduction

We have a good strength of data scientists in Walmart. Our system has counted more than 393 frequent users and 998 infrequent users. The data scientists are from various teams in Walmart. We have seen various kinds of challenges in the evolution of our platform to support end to end Machine learning (ML) model life cycle. We have observed that our data scientists are not able to use the platform effectively. Less adoption of our platform is the key challenge for us to solve. It is not sufficient to be completely dependent on surveys to improve our system as people often lie in user surveys, possibly due to a phenomenon known as social desirability bias [2].

Analyzer approach: We have instrumented our machine learning platform (MLP) to collect data for analysing how users interact with the platform. We named the data collected or sent from a product or a feature as signals. Example of signals are clicks, interaction with MLP features, time spent on features, time to complete certain actions, and so on. We have created various classes of signals for our product: MLP-action signals (e.g., key-actions, clicks, and views), MLP-error signals (e.g., error in each component) and MLP-time signal (e.g., time on component, component load time, minutes per session) [3, 4]. We have defined four stages of the ML lifecycle, namely, "Data collection,cleaning,labeling", "Feature Engineering", "Model training" and "Model deployment" [1]. In each stage signals are generated for the Analyzer to capture particular challenges faced by user.. We present two dimensions of challenges: Education-&-Training (ET) and Simplicity-of-Platform (SoP). In ET, the focus is on a data scientist's capability and self-sufficiency to use the MLP and SoP focuses on the technical or design challenges in ML lifecycle. The Fig. 1 shows our system architecture.

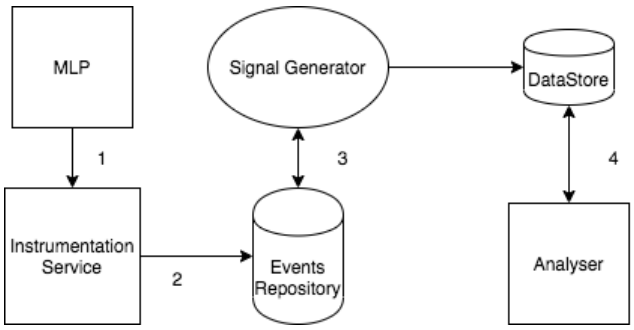


Figure 2: Collection of Signals

2 Signal Collection

Our instrumentation framework collect user actions from platform and sends it to our instrumentation service. We consider all actions as events and store them in our event repository (database). The signal generator reads the data from the repository and classifies it into multiple signals. We name the data collected or sent form product or feature as signals. Example of signals are clicks, interaction with the tools, time spent on the tools, time to complete certain actions. These signals are mapped to multiple classes: MLP-actions, MLP-error and MLP-time. The signal generator annotates these classes of signals with the user and platform related meta-data and stores it into a datastore. Our Analyzer runs the final show. It goes through each new signal and evaluates respective stage wise bottleneck for a user. Fig. 2 describes this flow.

3 Analyzer

In this section, we summarize the design of Analyzer, a system that uses a user-behaviour-segregation-model and set of rules to evaluate challenges of using MLP. The model uses K-means clustering technique, which groups the users based on action and time signals and later by looking at the centroid/clustor representative values. Users in the group are labelled as: Beginner, Intermediate and Expert. Our model has suggested 85 Expert, 118 Intermediate and 190 Beginner users of our platform. We have defined set of rules for each different type of signal. Analyzer applies these rules and smell the challenge from signal. We have defined following set of rules:

1. If more than 50% expert and 65% intermediate and 80% beginner users see the error signal then it is mapped to SoP challenge

2. If less than 10% expert and more than 65% intermediate and 80% beginner users see the error signal then it is mapped to ET challenge

3. if more than 50% expert does redundent actions more than 5 times in a week/month then it is mapped to SoP challenge

4. If less than 10% expert and more than 50% intermediate and 75% does redundent actions more than 5 times in a week/month then it is mapped to ET challenge

Challenge	Count
Education & Training	17
Simplicity-of-Platform	6

Table 1: Platform challenges

Above table depicts different instances of ET and SoP challenge at different stages. Some of critical instances are described below:

a) Model-training stage has a default cut-off time, beyond which jobs are terminated automatically. Analyzer has mapped these signals as ET challenge, based on rule-1, where we found users were unaware of this feature.

b) Data-collection stage provides connectors which enable users to retrieve data from varied data sources. Analyzer has mapped these signals as ET challenge, based on rule-4. We have discovered that users prefer creating their own connectors instead of reusing the existing ones, thus resulting in lot of duplicity.

c) Feature-engineering stage, user see a list of notebooks. Notebook IDE runs in a kubernetes container, if notebook is idle for longer than 6 hours then notebook becomes inactive. User need to activate notebook in order to open it. If there is no enough resource available then display error. Analyzer has mapped these signal as SoP challenge, based on rule-1.

Analyzer has been running in production for last 4 months and these above instances are the outcome of our data analysis. This has helped our product team to build better Education and Training (effective tutorial, documentation and videos) for model-training and data-connectors features. Above SoP instance: 'c' highlighted the complexity which user faces in creating notebook in Feature-engineering stage. This enabled product team to integrate live resource information on the notebook list page as part of UX improvement.

4 Conclusion

The Analyser has identified 17 and 6 instances respectively of Education-&-Training (ET) and Simplicity of Platform (SoP) challenges. ET instances have enabled product team to create effective set of documentation and tutorials. SoP instances helped in the evolution of ML Platform as a whole. Our

Analyzer helps us directly in making data driven decisions to improve user adoption. We believe that our Analyzer can be adopted by other Machine Learning Platform teams facing similar challenges. In future, we will add more challenges (ex: Scale, Performance, Hardware-Resources) and use Deep Learning for deriving more insights from these signals.

Acknowledgments

We would like to acknowledge our colleagues who built ML Platform, Senior Director: Ajay Sharma for the opportunity, Distinguised Architect: Bagavath Subramaniam, Product Manager: Rakshith Uj and Anish Mariyamparampil for helping us in instrumentation in UI and Saiyam Pathak for the design review.

References

[1] Saleema Amershi, Andrew Begel, Christian Bird, Rob DeLine, Harald Gall, Ece Kamar, Nachi Nagappan, Besmira Nushi, and Tom Zimmermann. Software engineering for machine learning: A case study. In *International Conference on Software Engineering (ICSE 2019) - Software Engineering in Practice track*. IEEE Computer Society, May 2019.

[2] Wikipedia contributors. Social desirability bias. https://en.wikipedia.org/wiki/Social_desirability_bias, Jan 2020. Accessed on 2020-01-21.

[3] Aleksander Fabijan, Pavel Dmitriev, Helena Holmstrom Olsson, and Jan Bosch. The evolution of continuous experimentation in software product development: From data to a data-driven organization at scale. *International Conference on Software Engineering*, pages 770–780, 2017.

[4] Kerry Rodden, Hilary Hutchinson, and Xin Fu. Measuring the user experience on a large scale: User-centered metrics for web applications. In *Proceedings of CHI 2010*, 2010.

Detecting Feature Eligibility Illusions in Enterprise AI Autopilots

Fabio Casati, Veeru Metha, Gopal Sarda, Sagar Davasam, Kannan Govindarajan
ServiceNow, Inc.

1 Problem and Motivation

SaaS Enterprise workflow companies, such as Salesforce and ServiceNow, facilitate AI adoption by making it easy for customers to train AI models on top of workflow data, *once they know the problem they want to solve and how to formulate it*. However, as we experience over and over, it is very hard for customers to have this kind of knowledge for their processes, as it requires an awareness of the business and operational side of the process, as well as of what AI could do on each with the specific data available. The challenge we address is how to take customers to that stage, and in this paper we focus on a specific aspect of such challenge: the identification of which "useful inferences" AI could make and which process attributes can (or cannot) be leveraged as predictors, based on the data available for that customer. This is one of the steps we see as central to improve what today is a very limited adoption of AI in enterprises, even in the most "digitized" organization [8,9]. For simplicity in the following we assume that business process data is stored in a DB table, and define the problem semi-formally as follows: Given a table T with a set of fields $f \in F$, and given a point $e \in E$ in the process where inference is desired, identify i) the set of fields $LF \subseteq F$ on which AI could make "useful predictions" (potential labels) and, for each such field $l \in LF$, ii) which sets of fields $FS_l \subseteq \mathcal{P}(F)$ are "usable predictors" (potential featuresets)[1]. This formulation, besides being simple and fairly general (many problems can be mapped to it) corresponds to the question our customers typically seek answer to.

2 What Makes the Problem Hard

The number of models we may want to train to explore our problem space is $|F| \cdot |E| \cdot 2^{|F|}$ (possible labels $\times$ point in the process for which we want the label to be predicted, $\times$ possible featureset). The database tables supporting a typical process have hundreds of fields, which makes the number of models larger than the number of atoms in the universe. Even if we assume that AI and state-of-the-art autoML [1, 3, 6] will do model/parameter search, data processing and transformation, feature engineering, and feature selection [2, 7] for us at scale, the space of possibilities for each process and customer is still in the thousands. Furthermore, fields in a table are often foreign keys to other tables (e.g., *user*, or *company*), and very often the predictive value are in these linked tables.

State	Assigned person	Assignment group	Duration	SLA Breached	Priority	...
Completed	Mary	HW support	3 days 4 hrs	YES	1	
Active	John	UI Forms		NO	2	
...						

Figure 1: Example Table Supporting a CSM Process

Scale, however, is not the only challenge. Consider a DB table supporting a Customer Service process (Figure 1), typically consisting of hundreds of fields. A snapshot of the table at any point in time would include a large proportion of completed cases and only a few in progress, at different points in their lifecycle - often not enough to build any kind of classifier per lifecycle state, except for the final state. However, if we use a snapshot for determining which fields are potentially "good" features and labels and if we then train models based on snapshots, our predictions would suffer from the following *illusions* about the ability to make useful predictions:

i) **Inverse causality**: Looking at a data snapshot we would believe we can infer the *Assignment Group* from the *Assigned Person*, but in reality the group is chosen first. The problem of causality is widely studied (e.g., [4, 5]), although mostly among snapshot of continuous variables, while the problem here is with categorical variables that evolve over time, where the temporal evolution is hard to capture.

ii) **Observability illusion**: While a causal dependency $f1 \rightarrow f2$ may exist and can be derived from a snapshot, $f1$ may not be observable at the time of prediction. For example, the

[1] $\mathcal{P}(F)$ denotes the powerset of F

case *duration* may depend on the assignment group, but we cannot predict duration at the start of the case because the independent variables are undefined (null) at that point.
iii) **End state bias**: Field values change during execution. For example, *Priority* is often *high* at the start, but then a workaround is found and priority drops to *low*. A snapshot would get many closed cases, so that the *priority* in a training dataset is biased towards the final value. Because we mostly have data from completed cases, an end state bias means that we will not be able to train a model with those features and label unless we take specific actions to correct the problem.

Even with infinite training capacity, the presence of any of these illusions will render a recommended model useless in production. Notice that while these problems may seem obvious once we read them, our experience is that they are not. This is true even when specific, targeted models are built by data scientists, often because scientists do not have a full grasp on the process, while process owners may not be familiar with ML. Indeed, we see calling out these problems as the main contributors here. We next discuss the strategies we adopt to tackle them, and for brevity we assume the ML model is required to make inferences at the start of a new case.

3 Uncovering Illusions

Audit-based process reconstruction. All enterprise systems allow some form of logging for audit purposes, but this is typically enabled only on some tables and fields. ServiceNow, for example, enables auditing and allows users to walk back in history on any (logged) record. However, audit tables are optimized for writing, not reading, and querying them is not a recommended practice as it will cause a heavy load on the system. The approach we take here is to obtain *samples* of the records, and walk back to the starting values. In a nutshell, i) observability can be estimated by looking at the field density at start, ii) causal inference can be heuristically identified by looking at which field in a pair $f1, f2$ gets filled first, and iii) end state bias is determined based on percentage of cases in which a field changes from first time it is filled to end.

Fig. 2, top half, shows experimental results for four processes from our customers' data, related to various aspects of customer service, from incident management to task execution for problem resolution (details omitted for privacy). The figure shows the mean (sd) number of fields that are *ineligible* as they are often null at the start or change more than a defined threshold. Here, "often" means > 20% of the times, but the numbers are not very sensitive to this threshold in our experience. 20% is also approximately the point where end bias affects trained models to a point that our customers find unusable. Results are averaged over 20 runs, each with 10 samples of 5 records each, for records created in different days and months. The figure shows that over 50% of the potential features are ineligible based on being null at time of prediction, 10% for end bias, and over 30% of the remaining potential feature-label pairs are also ineligible due to reverse causality. For example, this analysis would capture the reverse causality between assignment group and person. *p4* is empty because we did not have access to the history of such process.

Since cases are not *iid* (similar problems often happen in burst), we need to take repeated samples of small sets of records, in different days (in our experiments, going over 10 samples of 5 records does not lead to improved measures).
Starting or Delayed Snapshot. If (some) fields are not audited - as it often happens, we have to resort to snapshot-based heuristics. How to do so depends on what the SaaS platform allows, but common methods include i) looking for records where the last update and creation datetime is the same (*updated_time==created_time*), ii) leveraging an "update count" field (*update_count==0*), or iii) a state field (e.g., "*state==new case*"). If arrival of cases and time to first action on it are both *Poisson* with rates λ and τ per time unit, then we can expect $\tau \cdot \lambda$ new cases per time unit [10]. Notice that this approach does not lead to *iid* sampling, unless we operate on a live system and repeat the process in different days.

Audit-based reconstruction	number of fields	fields null at start	fields with end bias	total ineligible fields	ineligible feature-label pairs (inverse causality among eligible fields only)
Process P1	**99**	62.6 (0.9)	10.5 (1.1)	68.5 (1.3)	1908 (90)
Process P2	**173**	100.5 (3.9)	15.5(3.1)	105.7 (4.5)	6230 (216)
Process P3	**181**	119.3 (0.9)	29.8 (5.2)	131.5 (4.0)	4834 (583)
Process P4	-	-	-	-	-
Delayed snapshots					
Process P1	**99**	60.5(2.2)	-	60.5(2.2)	2303 (126)
Process P2	**173**	91.6 (0.9)	-	91.6 (0.9)	8011 (335)
Process P3	**181**	98.2 (2.5)	-	98.2 (2.5)	8659 (168)
Process P4	**241**	161.7 (4.0)	-	161.7 (4.0)	13193 (684)

Figure 2: Experimental results.

In many processes from our sample the number of useful "starting records" was less than a handful, either because agents pick up the work quickly or because business rules/chron jobs edit the record for whatever reason. Therefore we tweaked this approach to perform a *delayed sampling* (e.g., *update_count==1* or *update_time < created_time + 00:05:00*)). Fig. 2 shows results on the same processes with the same thresholds, with the exception of end bias which is harder to compute as we do not have the end state (statistical methods are possible but are beyond the scope of this short paper). While limited, delayed snapshot- based approaches still have the merit of having high precision in observability, and can also capture inverse causality as we extend the time window progressively.

Conclusion. The main take-home message is that i) applying ML to systems of record requires change and cause-effect analyses, ii) these analyses are challenging due to the scale of the problem in any realistic settings, and iii) a specific set of sampling methods can allow to filter out fields which are likely not useful for our model to concentrate the heavy-lifting analysis on fewer fields, something that is essential in resource constrained environments.

References

[1] AWSLabs. Autogluon: AutoML toolkit for deep learning. Available at: https://autogluon.mxnet.io/ (last accessed Feb 21, 2020).

[2] M. Dash and H. Liu. Feature selection for classification. *Intelligent Data Analysis*, 1(1):131 – 156, 1997.

[3] Xin He, Kaiyong Zhao, and Xiaowen Chu. AutoML: A survey of the state-of-the-art. Available at: https://arxiv.org/abs/1908.00709, Feb 2019.

[4] Patrik O. Hoyer, Dominik Janzing, Joris M Mooij, Jonas Peters, and Bernhard Schölkopf. Nonlinear causal discovery with additive noise models. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 689–696. Curran Associates, Inc., 2009.

[5] David Lopez-Paz, Krikamol Muandet, and Benjamin Recht. The randomized causation coefficient. *J. Mach. Learn. Res.*, 16(1):2901–2907, January 2015.

[6] Salesforce. TransmogrifAI. Technical report, Sept 2019. Available at: https://readthedocs.org/projects/transmogrifai/downloads/pdf/stable/.

[7] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Auto-weka: combined selection and hyperparameter optimization of classification algorithms. In *KDD '13*, 2013.

[8] Tamim Saleh Tim Fountaine, Brian McCarthy. Building the ai-powered organization. *Harvard Business Review*, 2019.

[9] Neil Webb. Notes from the AI frontier: AI adoption advances, but foundational barriers remain. *McKinsey & Company Report*, 2018.

[10] Moshe Zukerman. Introduction to queueing theory and stochastic teletraffic models. 07 2013.

Time Travel and Provenance for Machine Learning Pipelines

Alexandru A. Ormenisan
KTH - Royal Institute of Technology
Logical Clocks AB

Moritz Meister
Logical Clocks AB

Fabio Buso
Logical Clocks AB

Robin Andersson
Logical Clocks AB

Seif Haridi
KTH - Royal Institute of Technology

Jim Dowling
KTH - Royal Institute of Technology
Logical Clocks AB

Abstract

Machine learning pipelines have become the defacto paradigm for productionizing machine learning applications as they clearly abstract the processing steps involved in transforming raw data into engineered features that are then used to train models. In this paper, we use a bottom-up method for capturing provenance information regarding the processing steps and artifacts produced in ML pipelines. Our approach is based on replacing traditional intrusive hooks in application code (to capture ML pipeline events) with standardized change-data-capture support in the systems involved in ML pipelines: the distributed file system, feature store, resource manager, and applications themselves. In particular, we leverage data versioning and time-travel capabilities in our feature store to show how provenance can enable model reproducibility and debugging.

1 From Data Parallel to Stateful ML Pipelines

Bulk synchronous parallel processing frameworks, such as Apache Spark [4], are used to build data processing pipelines that use idempotence to enable transparently handling of failures by re-executing failed tasks. As data pipelines are typically stateless, they need lineage support to identify those stages that need to be recomputed when a failure occurs. Caching temporary results at stages means recovery can be optimized to only re-run pipelines from the most recent cached stage. In contrast, database technology uses stateful protocols (like 2-phase commit and agreement protocols like Paxos [10]) to provide ACID properties to build reliable data processing systems. Recently, new data parallel processing frameworks have been extended with the ability to make atomic updates to tabular data stored in columnar file formats (like Parquet [3]) while providing isolation guarantees for concurrent clients. Examples of such frameworks are Delta Lake [8], Apache Hudi [1], and Apache Iceberg [2]. These ACID data lake platforms are important for ML pipelines as they provide the ability to query the value of rows at specific points in time in the past (time-travel queries).

The Hopsworks Feature Store is built on the Hudi framework, where data files are stored in HopsFS [11] as parquet files and available as external tables in a modified version of Apache Hive [6] that shares the same metadata layer as HopsFS. HopsFS and Hive have a unified metadata layer, where Hive tables and feature store metadata are extended metadata for HopsFS directories. Foreign keys and transactions in our metadata layer ensure the consistency of extended metadata through Change-Data-Capture(CDC) events.

Just like data pipelines, ML pipelines should be able to handle partial failures, but they should also be able to reproducibly train a model even if there are updates to the data lake. The Hopsworks feature store with Hudi enables this, by storing both the features used to train a model and the Hudi commits (updates) for the feature data, see figure 1.

In contrast to data pipelines, ML pipelines are stateful. This state is maintained in the metadata store through a series of CDC events as can be seen in figure 1. For example, after a model has been trained and validated, we need state (from the metadata store) to check if the new model has better performance than an existing model running in production. Other systems like TFX [5] and MLFlow [14] also provide a metadata store to enable ML pipelines to make stateful decisions. However, they do so in an obtrusive way - they make developers re-write the code at each of the stages with their specific component models. In Hopsworks [9], however, we provide an unobtrusive metadata model based on implicit provenance [12], where change capture APIs in the platform enable metadata about artifacts and executions to be implicitly saved in a metadata store with minimal changes needed to user code that makes up the ML pipeline stages.

2 Versioning Code, Data, and Infrastructure

The defacto approach for versioning code is git, and many solutions are trying to apply the same process to the versioning of data. Tools, such as DVC [7] and Pachyderm [13] version data with git-like semantics and track immutable files, instead of changes to files. An alternative to git-like versioning that we chose is to use an *ACID Data-Lake* with time-travel query

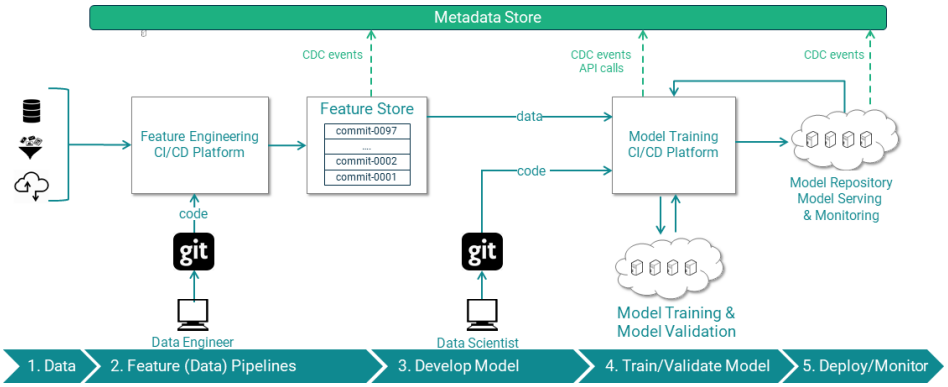


Figure 1: Hopsworks ML Pipelines with Implicit Metadata.

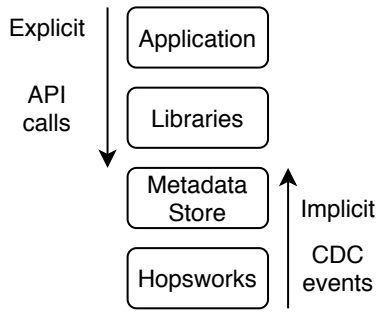


Figure 2: Implicit vs Explicit Provenance.

capabilities. Recent platforms such as Delta-Lake, Apache Hudi, Apache Iceberg extend data lakes with ACID guarantees using Apache Spark to perform updates and queries. These platforms store version and temporal information about updates to tables in a commit log (containing the row-level changes to tables). As such, you can issue time-travel queries such as "what was the value of this feature on this date-time", or "select this feature data for the time range 2017-2018". These type of queries are traditionally not possible in data warehouses that typically only store the latest values for rows in tables. Efficient time travel queries are enabled by temporal indexes (bloom filters in Hudi) over the parquet files that store the commits (updates to tables). Aside from code and data versioning, we also consider the versioning of infrastructure. With much of the ML work being done in Python, it is crucial to know the Python libraries and their versions when you ran a pipeline to facilitate importing/exporting and re-executing the pipeline, reproducibly.

3 Provenance in Hopsworks

TFX and MLFLow track provenance, by asking the user to explicitly mark the operation on a ML artifact that should be saved in their separate metadata store. We call this approach explicit, as the user has to explicitly say what would be tracked though calls to dedicated APIs. In Hopsworks, we instrument our distributed file systems - HopsFS, our resource manager - HopsYarn and our feature store to implicitly track the file operations, including the tagging of files with additional information. Our metadata store is also tightly coupled with the file system and thus we can capture metadata store operations in the same style. As we can see in figure 2, implicit provenance [12] relies on bottom-up propagation of Change-Data-Capture(CDC) events, while explicit provenance relies on user invocation of specific API and has a up-down propagation. With implicit provenance we can track which application used or created a specific file and who is the owner of the application. This together with file naming conventions and tagging of files allows us to automatically create relations between files on disk representing machine learning artifacts.

4 Reproducible ML Pipelines

With versioned code, data and environments, as well as provenance information to mark the usages of the particular version, it is easy to reproduce executions of the same pipeline. With the help of provenance information we augment this reproducibility scenario with additional functionality such as determining whether your current setup will provide similar results to the original and warn the user in case the input datasets, the environment or the code differ to the original. We also provide an environment that encourages exploration and learning through the ability to search through full text elasticsearch queries for most used datasets, most recent pipeline using a specific dataset or the persons who ran the latest version of a particular pipeline successfully. Provenance can also be used to warn a user not to expect a similar result to previous runs due to changes in the code, data or environment.

5 Provenance for Debugging ML Pipelines

Implicit provenance provides us with links between ML artifacts used and created in each of the stages of the pipeline. We know at what time they were used, by whom and in what application. This allows us to determine the footprint of each stage of the pipeline, as the files that were read, modified, created or changed in any way during the execution of the pipeline stages. From the footprint of each of the pipeline stages we can also determine the impact of previous stages. We defined the impact of a pipeline stage as the union of footprints for pipeline stages that use the output of the current stage as input. Since artifacts can be shared across pipelines, and since pipelines can be partially re-executed and forked into new pipelines, the impact of a pipeline can be quite large.

6 Summary

In this paper, we discussed how the implicit model for provenance can be used next to a feature store with versioned data to build reproducible and more easily debugged ML pipelines. Our future work will provide development tools and visualization support that can help developers more easily navigate and re-run pipelines based on the architecture we have built.

Acknowledgments

This work was funded by the EU Horizon 2020 project Human Exposome Assessment Platform (HEAP) under Grant Agreement no. 874662.

References

[1] Apache Hudi. https://hudi.apache.org/. [Online; accessed 25-February-2020].

[2] Apache Iceberg. https://iceberg.apache.org/. [Online; accessed 25-February-2020].

[3] Apache Parquet. https://parquet.apache.org/. [Online; accessed 25-February-2020].

[4] Apache Spark. https://spark.apache.org/. [Online; accessed 25-February-2020].

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

[6] Fabio Buso. Sql on hops. Master's thesis, KTH, School of Information and Communication Technology (ICT), 2017.

[7] Data Version Control. https://dvc.org/. [Online; accessed 25-February-2020].

[8] Delta Lake. https://delta.io/. [Online; accessed 25-February-2020].

[9] Mahmoud Ismail, Ermias Gebremeskel, Theofilos Kakantousis, Gautier Berthou, and Jim Dowling. Hopsworks: Improving user experience and development on hadoop with scalable, strongly consistent metadata. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2525–2528. IEEE, 2017.

[10] Leslie Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[11] Salman Niazi, Mahmoud Ismail, Seif Haridi, Jim Dowling, Steffen Grohsschmiedt, and Mikael Ronström. Hopsfs: Scaling hierarchical file system metadata using newsql databases. In *15th {USENIX} Conference on File and Storage Technologies ({FAST} 17)*, pages 89–104, 2017.

[12] Alexandru A. Ormenisan, Mahmoud Ismail, Seif Haridi, and Jim Dowling. Implicit provenance for machine learning artifacts. In *Proceedings of MLSys'20: Third Conference on Machine Learning and Systems.*, 2020.

[13] Pachyderm. https://www.pachyderm.com/. [Online; accessed 25-February-2020].

[14] Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, et al. Accelerating the machine learning lifecycle with mlflow. *IEEE Data Eng. Bull.*, 41(4):39–45, 2018.

An Experimentation and Analytics Framework for Large-Scale AI Operations Platforms

Thomas Rausch[1,2], Waldemar Hummer[2], Vinod Muthusamy[2]
[1] *TU Wien,* [2] *IBM Research AI*

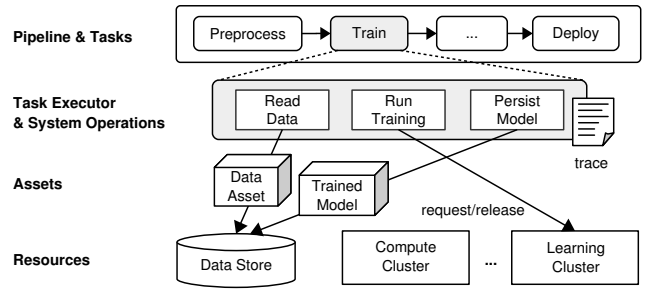
Abstract

This paper presents a trace-driven experimentation and analytics framework that allows researchers and engineers to devise and evaluate operational strategies for large-scale AI workflow systems. Analytics data from a production-grade AI platform developed at IBM are used to build a comprehensive system and simulation model. Synthetic traces are made available for ad-hoc exploration as well as statistical analysis of experiments to test and examine pipeline scheduling, cluster resource allocation, and similar operational mechanisms.

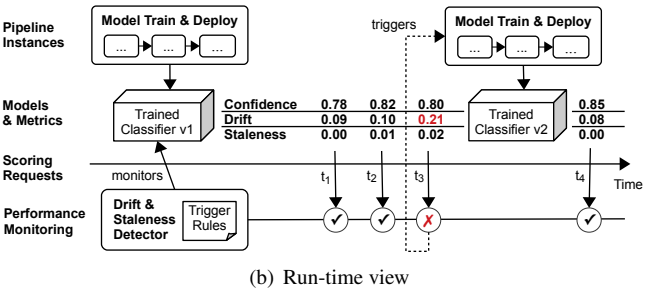
1 Introduction

Operationalizing AI has become a major endeavor in both research and industry. Automated, operationalized pipelines that manage the AI application lifecycle will form a significant part of infrastructure workloads [6]. AI workflow platforms [1, 6] orchestrate the heterogeneous infrastructure required to operate a large number of customer-specific AI pipelines. It is challenging to fine-tune operational strategies that achieve application-specific cost-benefit tradeoffs while catering to the specific domain characteristics of ML models, such as accuracy, or robustness. A key challenge is to determine the cost trade-offs associated with executing a pipeline, and the potential model performance improvement [5–7].

We present a trace-driven experimentation and analytics environment that allows researchers and engineers to devise and evaluate such operational strategies for large-scale AI workflow systems. Traces from a production-grade AI platform developed at IBM, recorded from several thousand pipeline executions over the course of a year are used to build a comprehensive simulation model. Our simulation model describes the interaction between pipelines and system infrastructure, and how pipeline tasks affect different ML model metrics. We implement the model in a standalone, stochastic, discrete event simulator, and provide a toolkit for running experiments. By integrating a time-series database and analytics front-end, we allow for ad-hoc exploration as well as statistical analysis of experiments.



(a) Build-time view



(b) Run-time view

Figure 1: Conceptual system model of AI ops platforms.

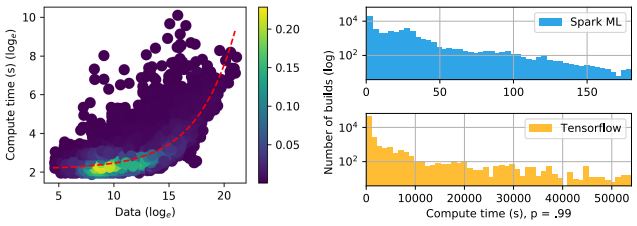
2 System Description

Conceptual Model Automated AI ops pipelines integrate the entire lifecycle of an AI model, from training, to deployment, to runtime monitoring [1, 6, 10]. To manage risks and prevent models from becoming stale, pipelines are triggered automatically by monitoring runtime performance indicators of deployed models. It is therefore essential to model both build-time and run-time aspects of the system. In general, we say that a model has a set of *static* and *dynamic* properties. Static properties are assigned by the pipeline at build-time, such as the prediction type (e.g., classification, or regression) or the model type (e.g., random forests or DNN). Dynamic properties change during runtime, such as model performance or robustness scores [13].

Build-time view: AI *pipelines* are workflows, i.e., graph-structured compositions of *tasks*, that create or operate on machine learning models [6]. At build time, an AI pipeline generates or augments a *trained model* by operating on *data assets* and using underlying infrastructure *resources* (e.g., data store, cluster compute or GPU nodes).

Run-time view: The outcome of a successful pipeline execution is usually a deployed model that is being served by the platform and used by applications for scoring. At runtime, the deployed model has associated performance indicators that change over time. Some indicators can be measured directly, by inspecting the scoring inputs and outputs (e.g., confidence), whereas other metrics (e.g., bias, or drift [4, 11]) require continuous evaluation of the runtime data against the statistical properties of the historical scorings and training data.

Synthesizing & Simulating Pipelines We synthesize plausible pipelines from three common pipeline structures we have identified by analyzing both commercial and research use cases [2, 3, 6, 9]. The generated pipelines vary in parameters such as the type of model they generate, the ML frameworks they use, or the number of tasks. The parameters are sampled from distributions we have fitted over analytics data. In the same way, we sample the metadata of data assets (such as the number of dimensions and instances or the size in bytes) as input for pipelines. We currently model the following pipeline steps: (a) data pre-processing (performing manipulation operations on the training data), (b) model training, (c) model validation, (d) model compression (e.g., removing layers from DNNs [12], changing the model size).



(a) Data preprocessing task (b) Training task

Figure 2: Observations of compute time for data preprocessing and training tasks based on other known properties.

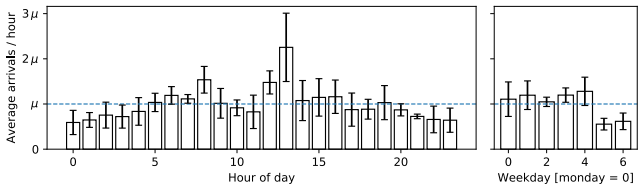


Figure 3: Average arrivals per hour stratified by hour of day and weekday ($n = 210824$). μ shows the average arrivals per hour overall. Error bars show one standard deviation.

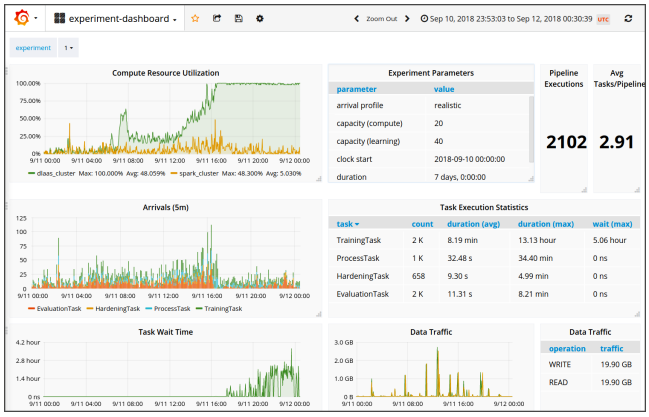


Figure 4: Experiment analytics dashboard showing infrastructure and pipeline execution metrics of an experiment run

For simulating pipeline executions, we have developed a stochastic, discrete-event simulator called PipeSim, which implements the conceptual system model and data synthesizers. We simulate task execution time and resource use based on our traces. Figure 2 shows examples of ways to simulate the compute time of data preprocessing and training tasks. For example, we correlate the size of a data asset with the preprocessing time Figure 2(a). For a training task, we stratify the observations into the frameworks they used, and the data asset size they processed. Figure 2(b) shows a distribution of compute times for Tensorflow and SparkML tasks.

To generate random workload we model the interarrivals of pipeline triggers in seconds as a random variable and sequentially draw from a fitted exponentiated Weibull distribution, which we found to produce a good fit. We variate means based on arrival patterns from our observations. Figure 3 shows pipeline triggers per hour averaged over several hundred thousand pipeline executions.

Experiment Runner & Explorer PipeSim is based on SimPy [8] and persists synthetic traces into InfluxDB. Resource allocation and scheduling algorithms are integrated as Python code into the simulator, which can then be evaluated by running PipeSim. The analytics frontend shown in Figure 4 allows exploratory analysis of experiment results. It displays the experiment parameters, general statistics about individual task executions and wait time. The graphs show the resource utilization of compute resources, individual tasks arrivals, network traffic, and overall wait time of pipelines, which allows us to quickly observe the impact of resource utilization on pipeline wait times.

We modeled the production system with PipeSim and analyzed the active scheduling policies. Experiments allowed us to approximate the increased execution times of pipelines given the projected user growth for the next year, and identify GPU cluster resource bottlenecks. We are now working to simulate and visualize aggregated model metrics to examine the effect of pipeline scheduling on overall model performance.

References

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

[2] IBM Corporation. Medtronic builds a cognitive mobile personal assistant app to assist with daily diabetes management, 2017. IBM Case Studies.

[3] IBM Corporation. An AI-powered assistant for your field technician. Technical report, 2018.

[4] Jaka Demšar and Zoran Bosnić. Detecting concept drift in data streams using model explanation. *Expert Systems with Applications*, 92:546–559, 2018.

[5] Sindhu Ghanta, Sriram Subramanian, Lior Khermosh, Harshil Shah, Yakov Goldberg, Swaminathan Sundararaman, Drew Roselli, and Nisha Talagala. MPP: Model performance predictor. In *2019 USENIX Conference on Operational Machine Learning*, OpML 19, pages 23–25, 2019.

[6] Waldemar Hummer, Vinod Muthusamy, Thomas Rausch, Parijat Dube, and Kaoutar El Maghraoui. Modelops: Cloud-based lifecycle management for reliable and trusted ai. In *2019 IEEE International Conference on Cloud Engineering (IC2E'19)*, Jun 2019.

[7] Tian Li, Jie Zhong, Ji Liu, Wentao Wu, and Ce Zhang. Ease.ml: Towards multi-tenant resource sharing for machine learning workloads. *Proc. VLDB Endow.*, 11(5):607–620, January 2018.

[8] Norm Matloff. Introduction to discrete-event simulation and the simpy language. *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August*, 2(2009), 2008.

[9] Thomas Rausch, Waldemar Hummer, Vinod Muthusamy, Alexander Rashed, and Schahram Dustdar. Towards a serverless platform for edge AI. In *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*, 2019.

[10] Evan R Sparks, Shivaram Venkataraman, Tomer Kaftan, Michael J Franklin, and Benjamin Recht. Keystoneml: Optimizing pipelines for large-scale advanced analytics. In *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*, pages 535–546. IEEE, 2017.

[11] Yange Sun, Zhihai Wang, Haiyang Liu, Chao Du, and Jidong Yuan. Online ensemble using adaptive windowing for data streams with concept drift. *International Journal of Distributed Sensor Networks*, 12(5):4218973, 2016.

[12] Dharma Teja Vooturi, Saurabh Goyal, Anamitra R. Choudhury, Yogish Sabharwal, and Ashish Verma. Efficient inferencing of compressed deep neural networks. *CoRR*, abs/1711.00244, 2017.

[13] Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. Evaluating the robustness of neural networks: An extreme value theory approach. *arXiv preprint arXiv:1801.10578*, 2018.

Challenges Towards Production-Ready Explainable Machine Learning

Lisa Veiber
SnT – Univ. of Luxembourg

Kevin Allix
SnT – Univ. of Luxembourg

Yusuf Arslan
SnT – Univ. of Luxembourg

Tegawendé F. Bissyandé
SnT – Univ. of Luxembourg

Jacques Klein
SnT – Univ. of Luxembourg

Abstract

Machine Learning (ML) is increasingly prominent in organizations. While those algorithms can provide near perfect accuracy, their decision-making process remains opaque. In a context of accelerating regulation in Artificial Intelligence (AI) and deepening user awareness, explainability has become a priority notably in critical healthcare and financial environments. The various frameworks developed often overlook their integration into operational applications as discovered with our industrial partner. In this paper, explainability in ML and its relevance to our industrial partner is presented. We then dis- cuss the main challenges to the integration of explainability frameworks in production we have faced. Finally, we provide recommendations given those challenges.

1 Introduction

The increasing availability of data has made automated techniques for extracting information especially relevant to businesses. Indeed, AI overall contribution to the economy has been approximated to $15.7tr by 2030 [8]. State-of-the-art frameworks have now outmatched human accuracy in complex pattern recognition tasks [6]. However, many accurate ML models are—in practice—black boxes as their reasoning is not interpretable by users [1]. This trade-off between accuracy and explainability is certainly a great challenge [6] when critical operations must be based on a justifiable reasoning.

Explainability is henceforth a clear requirement as regulations scrutinises AI. In Europe, our industrial partner faces GDPR, the General Data Protection Regulation, which includes the *right to explanation*, whereby an individual can request explanations on the workings of an algorithmic decision produced based on their data [5]. Additionally, user distrust, from their lack of algorithmic understanding, can cause a reluctance in applying complex ML techniques [1, 2]. Explainability can come at the cost of accuracy [10]. When faced with a trade-off between explainability and accuracy, industrial operators may, because of regulation reasons, have to resort to less accurate models for production systems. Finally, without explanations, business experts produce by themselves justifications for the model behaviour. This can lead to a plurality of explanations as they devise contradicting insights [3].

Integrating explainability in production is a crucial but difficult task. We have faced some challenges with our industrial partner. Theoretical frameworks are rarely tested on operational data, overlooking those challenges during the design process. Overcoming them becomes even more complex afterwards.

2 Explainability

Explainability is rather ill-defined in the literature. It is often given discordant meaning [7] and its definition is dependent on the task to be explained [4]. Nonetheless, we use explainability interchangeably with interpretability [7] and define it as aiming to respond to the opacity of the inner workings of the model while maintaining the learning performance [6].

From this definition, explainability can be modelled in different ways. First, interpretability can be either *global* or *local*. Whereas the former explains the inner workings of the model at the model level [4], the latter reaches interpretability at the instance level. Furthermore, there is the distinction between inherent and post-hoc interpretability. Inherent explainability refers to the model being explainable [1], while post-hoc explainability entails that once trained, the model has to further undergo a process which will make its reasoning explainable [9, 10]. This results in four different types of explainability frameworks. The effectiveness of the frameworks depends on the type chosen with respect to the task of the model we want to explain.

Moreover, explainability is usually achieved through visualization-based frameworks, which produce graphical representations of predictions, or text explanation of the decision [1]. Effective visualizations or textual description with a decision can be sufficient to reach explainability [3]. Still, those supposedly-interpretable outputs are rarely validated through user studies. In our case, the frameworks, which were not validated in such a way, yielded models that have proven to be just as non-interpretable as the original ML model for our industrial partner.

Various reasons for explainability were previously mentioned. Our industrial partner was further interested in explainability for audit purposes as they must provide justifications for the automated system decisions. Besides, they were concerned with potential biases within training datasets. Some features, such as gender, although perhaps appearing as effective discriminators in ML models, cannot be legally used in business operations analytics [12]. Yet, other less explicit attributes may be direct proxies to such restricted features, eventually creating biases in the models. Adding explainability to existing models can uncover existing biases arising from proxies included in the model. This allows the operator to change models when bias is identified.

3 Challenges to Explainability

3.1 Data Quality

Our industrial partner was implementing ML model on tabular data. One of the first challenge identified was that most frameworks are designed for Natural Language Processing and Computer Vision tasks. Thus, there are fewer frameworks focusing on tabular data. With this omission, complications relating to tabular data quality are not properly addressed. This resulted in limitations of the framework explainability. For instance, visualizations designed for continuous features became inefficient for categorical variables. Furthermore, frameworks tested on tabular data often rely on datasets which are optimal for visualization purposes. Our data was not optimal as it contained missing values, and clusters between classes were not clearly separated. For example, while the several approaches we tested reported impressive results on a selection of domain, it is often impossible to know beforehand whether or not it can be applied to a specific domain. Indeed, they proved to be far less valuable when applied to the real datasets of our partner. For one specific problem, we obtained visualizations such as shown in Figure 1. In this case, the display of the gradient projection of data points [10], which relies on comparing the different classes according to pairs of variables, was cluttered. There was no clear distinction between the classes, hence the framework did not provide interpretability.

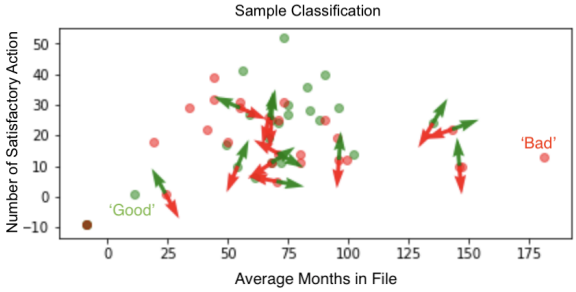


Figure 1: Example of visualization-based interpretability framework [10]

3.2 Task and Model Dependencies

Our industrial partner was implementing Random Forests. However, some frameworks are designed for specific model types, more particularly for Neural Networks and Support Vector Machines. This has been addressed by model agnostic approaches [9, 10] and surrogate models. Still, the latter loses accuracy when approximating the unexplainable model, only providing explanations when both models reach the same predictions, but not when they differ. Moreover, explainability is also task-dependent. Our industrial partner needed different explanations for different audiences. Yet, we detected through our experiments an emphasis on theoretical review of task dependency rather than empirical research. This insufficiency of practical consideration limits frameworks deployment in production, as a lengthy experimental process is required to inspect which explanations best fit the task.

3.3 Security

Another challenge from explainability in production is security. This was a significant concern for our partner. Indeed, if clients can have access to the reasoning of the model decision-making, they could apply adversarial behaviours by incrementally changing their behaviour to influence the decision-making process of the model. Thus, explainability raises robustness concerns preventing its immediate deployment in production. Furthermore, in a recent paper [11], it was shown that under strong assumptions, an individual having access to model explanations could recover part of the initial dataset used for training the ML algorithm. Given the strict data privacy regulations, this remains a case to investigate.

Implementing explainability frameworks in production can therefore significantly slow and complicate the project.

4 Conclusion

Data is crucial to derive information on which to base operational decisions. However, complex models achieving high accuracy are often opaque to the user. Explainable ML aims to make those models interpretable to users. The lack of research on operational data makes it challenging to integrate explainability frameworks in production stages. Several challenges to explainability in production we have faced include data quality, task-dependent modelling, and security. Given those challenges we recommend industrials to (1) clearly define their needs to avoid obstacles defined are task and model dependencies in this paper, (2) give more consideration to possible industrial applications when frameworks are designed, (3) undertake systematic user validation of frameworks to evaluate the explainability potential of those frameworks, (4) regarding the security challenges, we suggest to simulate adversarial behavior and observe the model behaviour to raise any robustness issues, (5) industrials could also undertake the exercise of recovering the entire training datasets given explanations and a small part of the original data.

References

[1] Or Biran and Courtenay Cotton. Explanation and justification in machine learning: A survey. In *IJCAI-17 workshop on explainable AI (XAI)*, volume 8, 2017.

[2] Chris Brinton. A framework for explanation of machine learning decisions. In *IJCAI-17 workshop on explainable AI (XAI)*, pages 14–18, 2017.

[3] Derek Doran, Sarah Schulz, and Tarek R Besold. What does explainable AI really mean? a new conceptualization of perspectives. *arXiv preprint arXiv:1710.00794*, 2017.

[4] Finale Doshi-Velez and Been Kim. A roadmap for a rigorous science of interpretability. *arXiv preprint arXiv:1702.08608*, 2, 2017.

[5] Bryce Goodman and Seth Flaxman. European union regulations on algorithmic decision-making and a "right to explanation". *AI magazine*, 38(3):50–57, 2017.

[6] David Gunning. Explainable artificial intelligence (xai). *Defense Advanced Research Projects Agency (DARPA), nd Web*, 2, 2017.

[7] Zachary C. Lipton. The mythos of model interpretability. *Queue*, 16(3):31–57, June 2018.

[8] PwC. Pwc's global artificial intelligence study: Sizing the prize. https://www.pwc.com/gx/en/issues/data-and-analytics/publications/artificial-intelligence-study.html, 2017. Accessed Feb 2020.

[9] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery.

[10] Andrew Slavin Ross, Michael C Hughes, and Finale Doshi-Velez. Right for the right reasons: Training differentiable models by constraining their explanations. 2017.

[11] Reza Shokri, Martin Strobel, and Yair Zick. Privacy risks of explaining machine learning models. *arXiv preprint arXiv:1907.00164*, 2019.

[12] Naeem Siddiqi. Intelligent credit scoring: Building and implementing better credit risk scorecards. 2017.

RIANN: Real-time Incremental Learning with Approximate Nearest Neighbor on Mobile Devices

Jiawen Liu[†], *Zhen Xie*[†], *Dimitrios Nikolopoulos*[‡], *Dong Li*[†]
[†]University of California, Merced [‡]Virginia Tech

Abstract

Approximate nearest neighbor (ANN) algorithms are the foundation for many applications on mobile devices. Real-time incremental learning with ANN on mobile devices is emerging. However, incremental learning with current ANN algorithms on mobile devices is hard, because data is dynamically and incrementally generated and as a result, it is difficult to reach high timing and recall requirements on indexing and search. Meeting the high timing requirements is critical on mobile devices because of the requirement of short user response time and because battery lifetime is limited.

We introduce an indexing and search system for graph-based ANN on mobile devices called RIANN. By constructing ANN with dynamic ANN construction properties, RIANN enables high flexibility for ANN construction to meet the strict timing and recall requirements in incremental learning. To select an optimal ANN construction property, RIANN incorporates a statistical prediction model. RIANN further offers a novel analytical performance model to avoid runtime overhead and interaction with the device. In our experiments, RIANN significantly outperforms the state-of-the-art ANN (2.42× speedup) on Samsung S9 mobile phone without compromising search time or recall. Also, for incrementally indexing 100 batches of data, the state-of-the-art ANN satisfies 55.33% batches on average while RIANN can satisfy 96.67% with minimum impact on recall.

1 Introduction

Approximate nearest neighbor (ANN) is an essential algorithm for many applications, e.g., recommendation systems, data mining and information retrieval [2, 4, 7, 10, 15–17] on mobile devices. For example, applications on mobile devices often provide recommendation functionalities to help users quickly identify interesting content (e.g., videos from YouTube [6], images from Flickr [14], or content from Taobao [9]). To meet user requirements, it is important to incrementally construct ANN on mobile devices in real-time. For example, it is essential to recommend to users new content that is of interest while the content is still fresh, as there is a clear tendency for users to prefer newer content. This necessitates real-time incremental learning for ANN on mobile devices.

However, real-time incremental learning for ANN on mobile devices imposes two challenges. First, current ANN models cannot meet the real-time requirement of high recall for incremental learning due to static graph construction. Specifically, with different size of batches in incremental learning, current ANN algorithms either index batches of data with high recall without meeting the real-time requirement or index data in real-time with low recall.

Second, current ANN algorithms perform end-to-end indexing hence indexing time, query time and recall are unknown to users prior to or during ANN indexing, while these results are required to reach high recall in real-time incremental learning.

To address the above challenges, we propose RIANN, a system to enabling real-time ANN incremental learning on mobile devices. To achieve our goals, we propose a dynamic ANN indexing structure based on HNSW [13]. With the dynamic ANN graph construction, we can target different indexing times, query times and recall on the fly. Next, we propose a statistical performance model to guide dynamic ANN construction over millions of possible properties. We further propose an analytical performance model to avoid interaction with mobile devices and runtime overhead.

2 Framework Design

Dynamic ANN graph construction. Currently, most graph-based ANN algorithms work in the following manner: during graph construction, the algorithms build a graph $G = (P, E)$ based on the geometric properties of points in dataset P with connecting edges E between the points. At search time, for a query point $p \in P$, ANN search employs a natural greedy traversal on G. Starting at some designated point $d \in P$, the algorithms traverse the graph to get progressively closer to p. The state-of-the-art ANN algorithm HNSW exploits the above procedure with building a multi-layer structure consisting of hierarchical set of proximity graphs (layers) for nested subsets of the stored elements.

However, since current graph-based ANN algorithms including HNSW are designed using static graph construction properties, there is little flexibility in controlling graph construction properties for real-time ANN incremental learning.

To address this problem, we propose RIANN to construct graphs in a dynamic manner. The dynamic graph construction depends on user requirements and a batch size of data

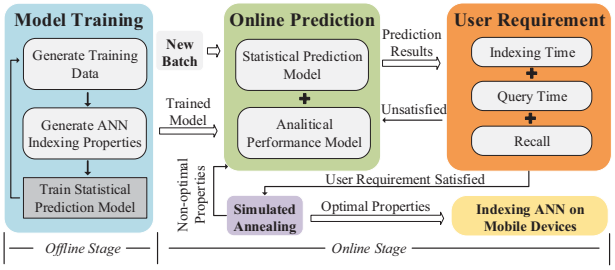


Figure 1: Overview of RIANN.

points. To achieve this goal, we build dynamic construction graph properties (e.g., out-degree edges of each point and candidates to build those edges). The advantage of dynamic group construction properties is that we can meet the real-time requirement while maintaining high recall.

Domain-specific statistical prediction model. The traditional approach to obtain the optimal indexing properties is to examine different indexing properties [3, 9, 11, 13]. However, to obtain the optimal indexing properties, this approach 1) requires an excessive amount of time (days and even weeks) [3] to obtain the optimum and 2) requests the exact indexing data size which is impractical in ANN incremental learning.

We propose a statistical prediction model to solve the problem. Figure 1 presents the overview of our design. The statistical prediction model is to estimate the recall and indexing or query time of each construction property for indexing a batch of data. The model is based on gradient boosted trees [8](GBTs) with simulated annealing [12]. We use XGBoost [5] as the GBTs model for training and implement a light-weighted XGBoost inference engine for mobile devices.

Analytical performance model. Though the prediction model is promising to predict ANN recall, the model has two issues to predict indexing/query time: 1) it interacts with mobile devices frequently to collect training data and 2) it incurs nonnegligible runtime overhead.

To address those problems, we propose an analytical performance model to predict indexing/query time at runtime. Equation 1 is used to estimate the time of querying data point $p \in P$ in one layer, where P refers to the set of points in one batch. The metric is defined as:

$$T_{lyr}(cand, deg, N) = T_d * h(cand, D_{avg}(deg, N)) * D_{avg}(deg, N) \quad (1)$$

Where *cand* represents candidate points, *deg* is the out-degree, N is the set of all points in one layer and T_d represents the time to calculate the distance. $h(cand, D_{avg}(deg, N))$ is a function to obtain the average number of hops from the entry point to the target point. The function can be formulated with small sample profiling offline. $D_{avg}(deg, N) = \frac{1}{|N|+1}(\sum_{i=1}^{|N|} N_i + deg)$ calculates the average out-degree after inserting p.

With the number of candidates and out-degree of p, the query time T_{qry} and indexing time T_{idx} are defined as follows:

$$T_{qry}(cand_q, deg) = \sum_{\ell=2}^{\ell_{max}} T_{lyr}(1, deg, N_\ell) + T_{lyr}(cand_q, deg * 2, N_1) \quad (2)$$

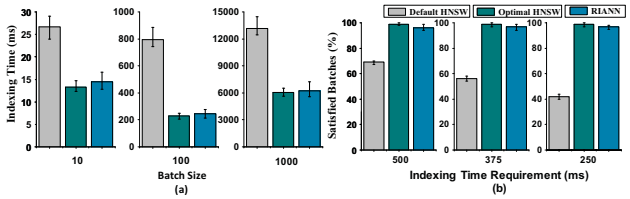


Figure 2: (a) Indexing time of RIANN and HNSW with different batch size. (b) Percentage of satisfied batches of RIANN and HNSW with different indexing time requirement.

$$T_{idx}(cand_i, deg) = \sum_{\ell=\ell_p}^{\ell_{max}} T_{lyr}(1, deg, N_\ell) + \sum_{\ell=2}^{\ell_p} (T_{lyr}(cand_i, deg, N_\ell) + f(deg)) + T_{lyr}(cand_i, dre * 2, N_1) + f(dre) \quad (3)$$

Where ℓ_{max} is the maximum number of layers, ℓ_p represents the indexing layer for p and $cand_q$ and $cand_i$ represent the number of candidates for query and indexing. The time of querying p from ℓ_{max} to ℓ can be calculated by $\sum_{\ell}^{\ell_{max}} T_{lyr}(cand, deg, N_\ell)$. The time of updating out-degree of p is calculated by $f(deg)$ that depends on the implementation and linear to the number of out-degree.

3 Evaluation

Comparison of RIANN and HNSW. We compare RIANN with HNSW which is the state-of-the-art ANN algorithm [3]. We employ SIFT [1] dataset on a Samsung S9 with Android 9. We use the graph construction properties listed in the paper [13] denoted as default HNSW. The optimal HNSW represents hypothetical results in which 1) users know the data size of ANN indexing which is impractical; 2) users spend a large amount of time (days and even weeks) to obtain the optimal HNSW. We experiment different batch size (10, 100 and 1000) in batch increments of 10 for incremental learning.

In Figure 2, we observe that RIANN shows significant performance improvement (2.42 times speedup on average) than the default HNSW and 8.67% performance less than the optimal HNSW while RIANN maintains the same recall and query time compared to the optimal and default HNSW. The runtime overhead of RIANN is included in Figure 2.

Evaluation with User Requirement. We incrementally index 100 batches and the batch size starts from 10 to 1000. In Figure 2, we observe that 55.33% batches are satisfied using default HNSW while RIANN can satisfy 96.67% with only 2.43% loss in recall.

4 Conclusion

We present RIANN, a real-time graph-based ANN indexing and search system. It constructs graphs in a dynamic manner with a statistical prediction model and an analytical performance model to incrementally index and search data in real-time on mobile devices. RIANN significantly outperforms the state-of-the-art ANN (2.42× speedup) without compromising query time or recall in incremental learning. Also, for incrementally indexing 100 batches of data, the state-of-the-art ANN satisfies 55.33% batches on average while RIANN can satisfy 96.67% with compromising 2.43% recall.

References

[1] Laurent Amsaleg and Hervé Jegou. Datasets for approximate nearest neighbor search. http://corpus-texmex.irisa.fr.

[2] Akhil Arora, Sakshi Sinha, Piyush Kumar, and Arnab Bhattacharya. Hd-index: Pushing the scalability-accuracy boundary for approximate knn search in high-dimensional spaces. *Proceedings of the VLDB Endowment*, 11(8):906–919, 2018.

[3] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 87:101374, 2020.

[4] Lei Chen, M Tamer Özsu, and Vincent Oria. Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 491–502, 2005.

[5] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[6] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, et al. The youtube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 293–296, 2010.

[7] Arjen P de Vries, Nikos Mamoulis, Niels Nes, and Martin Kersten. Efficient k-nn search on vertically decomposed data. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 322–333, 2002.

[8] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[9] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proceedings of the VLDB Endowment*, 12(5):461–474, 2019.

[10] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *Proceedings of the VLDB Endowment*, 9(1):1–12, 2015.

[11] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017.

[12] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[13] Yury A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[14] Börkur Sigurbjörnsson and Roelof Van Zwol. Flickr tag recommendation based on collective knowledge. In *Proceedings of the 17th international conference on World Wide Web*, pages 327–336, 2008.

[15] George Teodoro, Eduardo Valle, Nathan Mariano, Ricardo Torres, Wagner Meira, and Joel H Saltz. Approximate similarity search for online multimedia services on distributed cpu–gpu platforms. *The VLDB Journal*, 23(3):427–448, 2014.

[16] Chong Yang, Xiaohui Yu, and Yang Liu. Continuous knn join processing for real-time recommendation. In *2014 IEEE International Conference on Data Mining*, pages 640–649. IEEE, 2014.

[17] Yuxin Zheng, Qi Guo, Anthony KH Tung, and Sai Wu. Lazylsh: Approximate nearest neighbor search for multiple distance functions with a single index. In *Proceedings of the 2016 International Conference on Management of Data*, pages 2023–2037, 2016.

FlexServe: Deployment of PyTorch Models as Flexible REST Endpoints

Edward Verenich[§‡], Alvaro Velasquez[‡], M. G. Sarwar Murshed[§], and Faraz Hussain[§]

[§]Clarkson University, Potsdam, NY
{verenie,murshem,fhussain}@clarkson.edu

[‡]Air Force Research Laboratory, Rome, NY
{edward.verenich.2,alvaro.velasquez.1}@us.af.mil

Abstract

The integration of artificial intelligence capabilities into modern software systems is increasingly being simplified through the use of cloud-based machine learning services and representational state transfer architecture design. However, insufficient information regarding underlying model provenance and the lack of control over model evolution serve as an impediment to more widespread adoption of these services in operational environments which have strict security requirements. Furthermore, although tools such as TensorFlow Serving allow models to be deployed as RESTful endpoints, they require the error-prone process of converting the PyTorch models into static computational graphs needed by TensorFlow. To enable rapid deployments of PyTorch models without the need for intermediate transformations, we have developed FlexServe, a simple library to deploy multi-model ensembles with flexible batching.

1 Introduction

The use of machine learning (ML) capabilities, such as visual inference and classification, in operational software systems continues to increase. A common approach for incorporating ML functionality into a larger operational system is to isolate it behind a microservice accessible through a REpresentational State Transfer (REST) protocol [1]. This architecture separates the complexity of ML components from the rest of the application while making the capabilities more accessible to software developers through well-defined interfaces.

Multiple commercial vendors offer such capabilities as cloud services as described by Cummaudo et al. in their assessment of using such services in larger software systems [2]. They found a lack of consistency across services for the same data points, and also the unpredictable evolution of models, resulting in temporal inconsistency of a given service for identical data points. This behavior is due to the underlying classification models and their evolution as they are trained on different and additional data points by their vendors, providing insufficient information to the consuming system developer regarding the provenance of the model. Lack of control over the underlying models prevents many operational systems from consuming inference output from these services.

A better approach for preserving the benefits of a REST architecture while maintaining control of all aspects of model behavior is to deploy them as RESTful endpoints, thereby exposing them to the rest of the system. A popular approach for serving ML models as REST services is TensorFlow Serving [3]. However, serving PyTorch's dynamic graph through Tensor Flow Serving requires transforming PyTorch models to an intermediate representation such as Open Neural Network Exchange (ONNX) [4] which in turn is converted to a static graph compatible with TensorFlow Serving. This two-step conversion often fails and is difficult to debug because not all PyTorch features are supported by ONNX, making the train-test-deploy pipeline through TensorFlow Serving at best slow and at worst impossible for some PyTorch models. Another solution is to use the KFServing Kubernetes library [5][1], but that requires the deployment of Kubernetes, as KFServing is a serverless library and depends on a separate ingress gateway deployed on Kubernetes. Although this is a promising good solution, its deployment options are not lightweight when compared to TensorFlow Serving and Kubernetes itself can be complex to configure and manage [6].

To enable PyTorch model deployments in a manner similar to TensorFlow model deployments with TensorFlow Serving, *we have developed FlexServe, a simple REST service deployment module* that provides the following additional capabilities which are commonly needed in operational environments: (i) the deployment of multiple models behind a single endpoint, (ii) the ability to share a single GPU memory across multiple models, and (iii) the capacity to perform inference using flexible batch sizes.

In the remainder of the paper, we give an overview of FlexServe and demonstrate its advantages in scenarios such as those outlined above.

[1] KFServing is currently in beta status.

Figure 1: FlexServe architecture consists of an ensemble module which loads N models into a shared memory space. Inference output of each model is combined in a single response and returned to the requesting client as a JSON response object. The Flask application invokes the *fmodels* module, which encompasses the ensemble of models, and exposes RESTful endpoints through the Web Service Gateway Interface. Variable batch sizes provide for maximum efficiency and flexibility as clients are not restricted to a fixed batch size of samples to send to the inference service. Additional efficiency is achieved through the use of the shared memory, better utilizing available GPUs and requiring only one data transformation for all models in the ensemble.

2 Approach and Use Cases

FlexServe is implemented as a Python module encompassed in a Flask [7] application. We chose the lightweight Gunicorn [8] application server as the Web-Server Gateway Interface (WSGI) HTTP server to be used within the Flask application. This WSGI enables us to forward requests to web applications using the Python programming language. Figure 1 shows the high-level FlexServe architecture and its interaction with consuming applications.

2.1 Multiple models, single endpoint

Running ensembles of models is a common way to improve classification accuracy [9], but it can also be used to adjust the number of false negatives of the ensemble dynamically. Consider an ensemble of n models trained to recognize the presence of a specific object. By using different architectures, the ensemble model takes advantage of different inductive biases that perform better at different geometric variations of the target object. Then, combining its inference outputs according to the sensitivity policy of the consuming application, ensemble sensitivity can be adjusted dynamically. For example, let $y \in \{0,1\}$ be a binary output (0=absent, 1=present) of a classifier and let y' be the combined output. Then for maximum sensitivity the policy is $y' = y_1 \parallel y_2 \parallel \ldots \parallel y_n$, meaning that when a single model detects the target, the final ensemble output is positive identification. Different sensitivity policies can be employed by the client as needed.

2.2 Share a single device

Deployed models vary in size, but are often significantly smaller than the memory available on hardware accelerators such as GPUs. Loading multiple models in the same device memory brings down the cost and provides for more efficient inference. FlexServe allows multiple models to be loaded as part of the ensemble and performs multi-model inference on a single *forward* call of *nn.Module*, thereby removing the additional data transformation calls associated with competing methods. Scaling horizontally to multiple CPU cores is also possible through the use of Gunicorn workers.

2.3 Varying batch size

FlexServe accepts varying batch sizes of image samples and returns a combined result of the form *'model y_i': ['class','class', ...,'class']* for every model y_i. This functionality can be used in many applications. For example, to perform time series tracking from conventional image sensors or inexpensive web cameras by taking images at various time intervals and sending these chronological batches of images to FlexServe. As an object moves through the field of view of the sensor, a series of images is produced that can be used to infer movement of an object through the surveillance sector when more sophisticated object trackers are not available and video feeds are too costly to transmit. This places the computation and power burden on the Flask server as opposed to the potentially energy-constrained consumer which is only interested in the inference results of the ensemble model.

3 Conclusion

Commercial cloud services offer a convenient way of introducing ML capabilities to software systems through the use of a REST architecture. However, lack of control over underlying models and insufficient information regarding model provenance & evolution limit their use in operational environments. Existing solutions for deploying models as RESTful services are not robust enough to work with PyTorch models in many environments. FlexServe is a lightweight solution that provides deployment functionality similar to TensorFlow Serving without intermediate conversions to a static computational graph.

Availability

The FlexServe deployment module is available in a public repository (https://github.com/verenie/flexserve) which provides details showing how FlexServe can be used to deploy an ensemble model and also describes its limitations with respect to the REST interface.

Acknowledgments

The authors acknowledge support from the StreamlinedML program (Notice ID FA875017S7006) of the Air Force Research Laboratory Information Directorate (EV) and startup funds from Clarkson University (FH).

References

[1] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. Hidden Technical Debt in Machine Learning Systems. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, page 2503–2511, Cambridge, MA, USA, 2015. MIT Press.

[2] Alex Cummaudo, Rajesh Vasa, John C. Grundy, Mohamed Abdelrazek, and Andrew Cain. Losing Confidence in Quality: Unspoken Evolution of Computer Vision Services. *CoRR*, abs/1906.07328, 2019.

[3] Christopher Olston, Noah Fiedel, Kiril Gorovoy, Jeremiah Harmsen, Li Lao, Fangwei Li, Vinu Rajashekhar, Sukriti Ramesh, and Jordan Soyke. Tensorflow-serving: Flexible, high-performance ML Serving. *arXiv preprint arXiv:1712.06139*, 2017.

[4] Junjie Bai, Fang Lu, Ke Zhang, et al. Onnx: Open neural network exchange. https://github.com/onnx/onnx, 2019.

[5] KFServing. https://github.com/kubeflow/kfserving. Accessed: 2020-02-25.

[6] Kelsey Hightower, Brendan Burns, and Joe Beda. *Kubernetes: up and running: dive into the future of infrastructure*. " O'Reilly Media, Inc.", 2017.

[7] Flask. https://github.com/pallets/flask. Accessed: 2020-02-25.

[8] Gunicorn. https://gunicorn.org/. Accessed: 2020-02-25.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016.

Auto Content Moderation in C2C e-Commerce

Shunya Ueta, Suganprabu Nagarajan, Mizuki Sango
Mercari Inc.

Abstract

Consumer-to-consumer (C2C) e-Commerce is a large and growing industry with millions of monthly active users. In this paper, we propose auto content moderation for C2C e-Commerce to moderate items using Machine Learning (ML). We will also discuss practical knowledge gained from our auto content moderation system. The system has been deployed to production at Mercari since late 2017 and has significantly reduced the operation cost in detecting items violating our policies. This system has increased coverage by 554.8 % over a rule-based approach.

1 Introduction

Mercari[1], a marketplace app is a C2C e-commerce service. In C2C e-commerce, trading occurs between consumers who can be both sellers and buyers. Unlike standard business-to-consumer (B2C) e-commerce, C2C buyers can buy items that are listed without strict guidelines. However, C2C e-commerce has the risk of sellers selling items like weapons, money and counterfeit items that violate our policies, intentionally or unintentionally. Our company needs to prevent the negative impact such items have on buyers, and we also need to comply with the law regarding items that can be sold in our marketplace. In order to keep our marketplace safe and protect our customers, we need a moderation system to monitor all the items being listed on it. Content moderation has been adopted throughout industry by Microsoft [3], Google [2] and Pinterest [1] in their products. Rule based systems are easy to develop and can be quickly applied to production. However the logic of rule based system is hard to manage and it is difficult to cover the inconsistencies in (Japanese) spellings. It is also infeasible for human moderators to review all the items at such a large scale.

Machine Learning (ML) systems can overcome the limitations of rule based systems by automatically learning the features of items deleted by moderators and adapting to spelling inconsistencies. The moderators review the items predicted as positive by our system and we continuously re-train our models with the latest annotated data. Figure 1 shows an overview of our content moderation system. The moderator functions as a Human In The Loop to review the results of ML inference, rule based logic and reported items from customers, and helps regulate the items that are deleted.

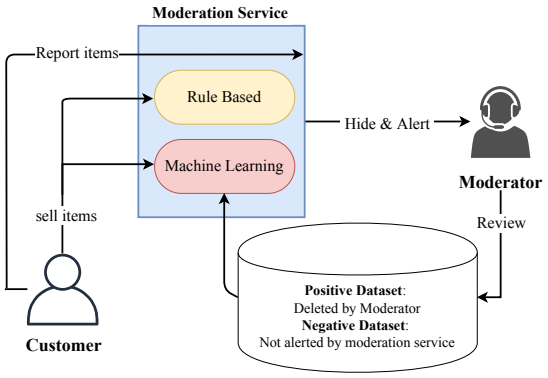


Figure 1: C2C e-Commerce content moderation system overview

The main contributions of this paper are (1) implementing multi-modal classifier models for imbalanced data in the wild (2) introducing and updating models in production (3) and preventing concept drift. In this paper, we also discuss more specifics about how we moderate items in our marketplace using ML.

2 Method

2.1 Model Training

Item listings in our marketplace consist of multi-modal data (e.g. text, image, brand, price), so we use multi-modal models to improve model performance. All models are trained in a one-vs-all setup since alerts from different models can over-

[1] Mercari. https://about.mercari.com/en/

lap. One model corresponds to one violated topic. One-vs-all models can also be easily re-trained and deployed independently. We made a pipeline using Docker [9] container-based workloads on a Kubernetes [6] cluster for model training workloads. We write manifest files containing requirements like CPU, GPU and Storage which are deployed using this pipeline.

For ML algorithms, we used Gated Multimodal Units (GMU) [5] and Gradient Boosted Decision Trees (GBDT) [7]. GMU potentially provides the most accuracy using multimodal data. GBDT is efficient to train and use for prediction when training dataset size is not large. We train the GMU models using PyTorch [10] and deploy them using PyTorch to ONNX [4] to Caffe2 [8] conversion.

The system then automatically evaluates the new model against the current model in production using offline evaluation (Sec. 2.2).

2.2 Evaluation

We propose offline and online evaluation to avoid concept drift [11].

Offline evaluation. We use the precision@K of the model as our evaluation metric since it directly contributes to the moderator's productivity, where K is the bound on the number of alerts in each violated topic and is defined by the moderation team. We evaluate the new model based on back-tests (the current model's output on test data is known). The back-tests guarantee that the new model is not worse than the current model which prevents concept drift. However, this test is biased towards the current model because the labels were created based on it. Thus, we also evaluate online for a predetermined number of days by using both models for predictions on all items.

Online evaluation. In our scenario, A/B testing is slow for decision making because the number of violations is much lower than valid item listings. As a result, A/B testing can take several months. This results in concept drift occurring and does not meet our business requirements. For faster decision making, we deploy the current and new models in production, and both of them accept all traffic. We set the thresholds of current and new models to alert half the target number each in that violated topic. The current and new model send $\frac{K}{2}$ alerts each to the moderator. If the new model has better precision@$\frac{K}{2}$ during online evaluation, we deprecate the old one and expand the new model to the target number of alerts. Table 1 shows the relative performance gain of the model based on precision@K. It shows our back-test reflecting the performance in production.

3 System Design

Figure 2 describes the system architecture. Our deployments are managed using Horizontal Pod Autoscaler which helps to maintain high availability and cut down production costs. The system has a proxy layer which gets messages from a queue and makes REST calls to the prediction layer. The prediction layer is responsible for preprocessing and inference, and returns a prediction result to the proxy layer. The proxy layer aggregates the responses from all the models and publishes messages for those items predicted as positive by at least one model, to a different queue where these messages are then picked up by a worker, and sent to the moderators for manual review of items. In online and offline evaluation, the proxy layer logs the predictions from all models and these logs are exported to a Data Lake.

Table 1: Percentage gains of GBDT and GMU compared to Logistic Regression in offline and online evaluation on one violated topic.

Algorithms	Offline	Online
GBDT	+18.2 %	Not Released
GMU	**+21.2%**	**+23.2%**

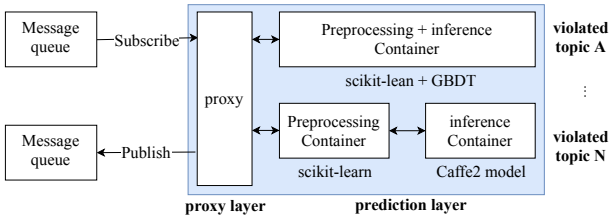


Figure 2: Auto content moderation system architecture. **GBDT**: One container contains the preprocessing and inferences. **GMU**:GMU has two containers. i) preprocessing. ii) inference using Caffe2.

4 Conclusion

Content moderation in C2C e-Commerce is a very challenging and interesting problem. It is also an essential part of services providing content to customers. In this paper, we discussed some of the challenges like new ML model introduction into production and how to efficiently prevent concept drift based on our experience. Our Auto Content Moderation system successfully increased moderation coverage by 554.8 % over a rule-based approach

Acknowledgments

The authors would like to express their gratitude to Abhishek Vilas Munagekar and Yusuke Shido for their contribution to this system and Dr. Antony Lam for his valuable feedback about the paper.

References

[1] Getting better at helping people feel better. https://newsroom.pinterest.com/en/post/getting-better-at-helping-people-feel-betterl. Accessed on 2020.04.14.

[2] Google maps 101: how contributed content makes a more helpful map. https://www.blog.google/products/maps/google-maps-101-how-contributed-content-makes-maps-helpful/. Accessed on 2020.04.14.

[3] New machine-assisted text classification on content moderator now in public preview. https://azure.microsoft.com/es-es/blog/machine-assisted-text-classification-on-content-moderator-public-preview/. Accessed on 2020.04.14.

[4] Open neural network exchange format (onnx). https://github.com/onnx/onnxl. Accessed on 2020.02.24.

[5] John Arevalo, Thamar Solorio, Manuel Montes-y Gómez, and Fabio A González. Gated multimodal units for information fusion. *arXiv preprint arXiv:1702.01992*, 2017. https://arxiv.org/abs/1702.01992.

[6] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, omega, and kubernetes. *Queue*, 14(1):70–93, 2016.

[7] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[8] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678, 2014.

[9] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[10] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[11] Indrė Žliobaitė, Mykola Pechenizkiy, and Joao Gama. An overview of concept drift applications. In *Big data analysis: new algorithms for a new society*, pages 91–114. Springer, 2016.

Challenges and Experiences with MLOps for Performance Diagnostics in Hybrid-Cloud Enterprise Software Deployments

Amitabha Banerjee, Chien-Chia Chen, Chien-Chun Hung, Xiaobo Huang, Yifan Wang, Razvan Chevesaran
VMware Inc., Palo Alto, CA, USA

Abstract

This paper presents how VMware addressed the following challenges in operationalizing our ML-based performance diagnostics solution in enterprise hybrid-cloud environments: data governance, model serving and deployment, dealing with system performance drifts, selecting model features, centralized model training pipeline, setting the appropriate alarm threshold, and explainability. We also share the lessons and experiences we learned over the past four years in deploying ML operations at scale for enterprise customers.

1. Introduction

Operationalizing ML solutions is challenging across the industry as described in [1-2]. In addition, VMware faces several unique challenges to productize our ML-based performance diagnostics solution. As one of the top hybrid-cloud providers, VMware has the most large-enterprise customers who deploy our Software-Defined Datacenter (SDDC) stack in both their on-premises datacenters as well as VMware-managed public clouds. This unique position gives VMware an opportunity to collect telemetry data from a wide variety of hardware/software combinations all over the world. However, it also results in two challenges: **(1) Data governance**. VMware must comply with local data regulations where the SDDC deployments reside, private cloud or public cloud. **(2) Model deployment**. Enterprise customers often have a long and inconsistent cycle to update the software, typically once every year or two. It is impractical to couple ML model deployment with customer's inconsistent lengthy update cycles.

Moreover, developing ML-based solutions for performance diagnostics needs to address the following issues: **(1) Handling Performance drifts**. System performance changes across the time due to software/hardware updates or normal hardware degradation. **(2) Feature engineering**. The system performance depends on numerous factors, which constitutes a multivariate ML problem. Consequently, the selection of the appropriate features plays an important role in model performance, and often requires human intervention. **(3) Model training**. Our experience has found that models perform better when being trained with global data from all deployments sharing a common hardware type. Thus, a centralized ML pipeline that complies with data regulations is required. **(4) Sensitivity to alarm threshold:** Flooded false alarms lead to either alarm disabling or overwhelming product support. On the other hand, conservative alarm threshold may miss performance issues and lead to silent failures and hence costly manual investigation. It is therefore critical to identify the appropriate alarm threshold. **(5) Explainability**. Once an issue is detected and an alarm is fired, it is important to provide insights and recommendations as to how to remediate the issue. Otherwise, it can still lead to unnecessary product support overhead.

2. Solution Design and Deployment

We have been deploying ML solutions to detect and root-cause performance issues in two offerings: (1) VMware vSAN Performance Diagnostics, a product feature allowing enterprise customers to self-diagnose performance issues in their deployments [3], and (2) VMware Support Insight, a support facing feature that allows VMware support teams to proactively monitor performance issues across all product deployments in a dashboard [4]. Figure 1 describes the ML/Ops lifecycle by which these solutions work from an operational perspective.

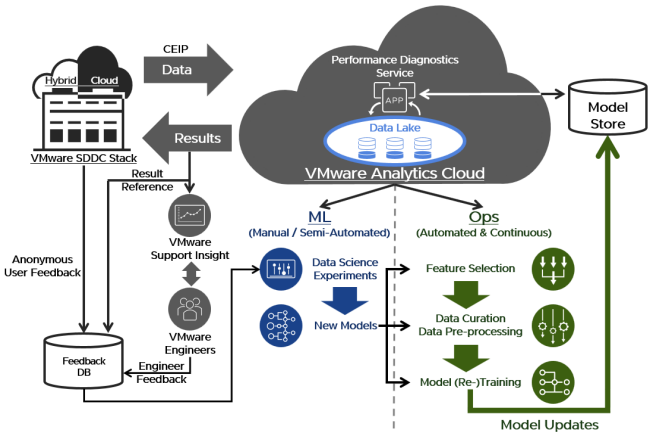


Figure 1. ML/Ops Lifecycle of Our Solution

2.1 ML/Ops Lifecycle

Data collection is done via VMware vCenter [5], the appliance managing a cluster of servers. For those customers who opted-in VMware Customer Experience Improvement Program (CEIP) [6], vCenter periodically transmits performance counters hourly to the VMware's centralized data platform, VMware Analytics Cloud (VAC) [7]. VAC stores and governs all the collected telemetry data and performs necessary placement, anonymization and encryption to comply with local data regulations. VAC also provides a run-time platform to deploy and run our ML-based performance diagnostics services. The results of the performance diagnostics are stored

in a key-value database in VAC from which they can be consumed either by the source vCenter, where the analysis in the form of charts, alarms, or recommendations are presented to the customers, or by support tools such as VMware Support Insight, where the support engineers monitor the results of multiple deployments simultaneously. Since the service runs within VMware's cloud infrastructure, its updates are decoupled from the customer's software update cycles, which allows a continuous deployment of new models [8]. VAC deployment addresses our challenges of data governance and model deployment.

2.2 ML/Ops Pipeline

We develop an ML/Ops pipeline to tackle the five challenges in our performance diagnostics use case. We develop a fully automated pipeline to continuously train new models based on the newly arrived batch of data. These models are maintained and version-controlled in the model store. The pipeline evaluates a large number of feature sets combining with various data pre-processing techniques and different ensembles of ML algorithms. The continuous training and deployment ensure our production model always learns the latest performance behaviors. Different models are trained for different hardware configurations. As an example, the model for a server with all-flash NVMe storage is different from that of a server with mixed magnetic disks and flash storage.

When deployed for performance diagnostics, the newest stable model specific to the hardware configuration of the system is used. If a rollback is desired, performance diagnostic would revert to the last-known-good model or any earlier model specified with certain version name.

The feature set candidates are first defined by product experts based on the specific performance issues to detect. Note that these feature sets are merely means to guide the pipeline to explore the feature space more efficiently, since there are easily thousands of thousands of performance metrics in a production SDDC. The pipeline also exploits several standard feature selection techniques such as PCA and autoencoder for each given set in addition to the human chosen ones, to achieve the best results [9-10].

Before feeding data to ML algorithms, the pipeline applies various data pre-processing techniques such as Z-transformation and Box-Cox transformation. We learned that ML algorithms might perform very differently with different data pre-processing techniques. For example, certain models prefer all dimensions to be in normal distribution, so Z-transformation works the best for this type of models. To provide explainability on top of the models' results, our pipeline employs a rule-based root-cause analysis engine to pinpoint the source of performance issue.

The rest of the pipeline includes the common ML training steps such as optimizing against a training set, applying boosting algorithms, and validating against a validation set to find the best performing model ensemble. Once a new model ensemble is trained and deployed in all three solutions outlined in Section 2.1, product performance experts, site reliability engineers, and solution engineers start consuming the output. These experts continuously monitor the latest model performance, review feedback given by service consumers, and examine new performance issues. This requires several manual or at best semi-automated steps to understand scenarios where the ML models are under-performing. To make this process more efficient, we developed a framework that enables performance perturbations in carefully orchestrated experiments and examine the behaviors from our models. We also leverage generic monitoring solution such as VMware Wavefront to monitor the performance of deployed models in conjunction with the feedback provided by the users [11]. As an example, we identified that our initial models were having high false positives in situations where very large IO size requests were being handled by vSAN because the ML models were overfitted towards the more common scenarios of small IO size requests. In such situations, we push new datasets to the Ops pipeline and evaluate if the changes improve the performance of the diagnostic service.

3. Lessons Learned

This paper describes our efforts spanning over four years in operationalizing an ML solution to detect and diagnose performance issues in our production deployed enterprise solutions. We learned several valuable lessons as follows:

(1) Continuous training and serving. Prior to building a pipeline, majority of the development time was spent on manual steps that are error-prone and time consuming. The ML/Ops pipeline automates our workflow and helps us churn models in hours, steps which earlier took us close to six months.

(2) Importance of a monitoring solution. Having a monitoring solution with great visualization helps us immensely in understanding why performance anomalies were being identified by an ML model and getting an intuitive understanding of how ML models work.

(3) Importance of automation: Continuous delivery of new models, noting improvements, and rollbacks when necessary are immensely valuable for production use cases. Delivering an ML framework for performance diagnostics in production necessitates a robust alarm threshold in a dynamically changing environment. It is hard to achieve the same without an efficient ML Ops lifecycle.

(4) Orchestrated experiment environment for model behavior validation. Having a framework that enables injection of the perturbed configuration allows us to quickly examine and verify whether the models react appropriately to a specific scenario. This mitigates the overhead with investigating the model issues in production environment.

4. References

[1] Z. Li, Y. Dang, “AIOps: Challenges and Experiences in Azure”, OpML 2019

[2] MLOps: Continuous delivery and automation pipelines in machine learning. https://cloud.google.com/solutions/machine-learning/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning.

[3] vSAN Performance Diagnostics. https://kb.vmware.com/s/article/2148770.

[4] VMware Support Insight. https://storagehub.vmware.com/t/vsan-support-insight/.

[5] VMware vCenter. https://www.vmware.com/products/vcenter-server.html.

[6] VMware Customer Experience Improvement Program (CEIP). https://www.vmware.com/solutions/trustvmware/ceip.html.

[7] VMware Analytics Cloud (VAC). https://blogs.vmware.com/vsphere/2019/01/understanding-vsphere-health.html.’

[8] A. Banerjee and A. Srivastava, “A Cloud Performance Analytics Framework to Support Online Performance Diagnosis and Monitoring Tools”, in Proc. of the 2019 ACM/SPEC International Conference on Performance Engineering, pp. 151–158, Mumbai, India, April 2019.

[9] I.T. Jolliffe, "Principal Component Analysis", Second Edition, Springer (2002).

[10] D. H. Ballard, "Modular learning in neural networks," Proceedings of the sixth National conference on Artificial intelligence, July, 1987, Seattle, WA, U.S.A.

[11] VMware Wavefront. https://cloud.vmware.com/wavefront.