GradeML: Towards Holistic Performance Analysis for Machine Learning Workflows

Tim Hegeman
T.M.Hegeman@vu.nl
Vrije Universiteit Amsterdam
The Netherlands

Matthijs Jansen
M.S.Jansen@vu.nl
Vrije Universiteit Amsterdam
The Netherlands

Alexandru Iosup
Vrije Universiteit Amsterdam
The Netherlands

Animesh Trivedi
Vrije Universiteit Amsterdam
The Netherlands

ABSTRACT

Today, machine learning (ML) workloads are nearly ubiquitous. Over the past decade, much effort has been put into making ML model-training fast and efficient, e.g., by proposing new ML frameworks (such as TensorFlow, PyTorch), leveraging hardware support (TPUs, GPUs, FPGAs), and implementing new execution models (pipelines, distributed training). Matching this trend, considerable effort has also been put into performance analysis tools focusing on ML model-training. However, as we identify in this work, ML model training rarely happens in isolation and is instead one step in a larger ML *workflow*. Therefore, it is surprising that there exists no performance analysis tool that covers the entire life-cycle of ML workflows. Addressing this large conceptual gap, we envision in this work a holistic performance analysis tool for ML workflows. We analyze the state-of-practice and the state-of-the-art, presenting quantitative evidence about the performance of existing performance tools. We formulate our vision for holistic performance analysis of ML workflows along four design pillars: a unified execution model, lightweight collection of performance data, efficient data aggregation and presentation, and close integration in ML systems. Finally, we propose first steps towards implementing our vision as GradeML, a holistic performance analysis tool for ML workflows. Our preliminary work and experiments are open source at https://github.com/atlarge-research/grademl.

KEYWORDS

GradeML, Performance analysis, Machine learning workflow, Data gathering, Modeling, MLDevOps

ACM Reference Format:
Tim Hegeman, Matthijs Jansen, Alexandru Iosup, and Animesh Trivedi. 2021. GradeML: Towards Holistic Performance Analysis for Machine Learning Workflows. In *Companion of the 2021 ACM/SPEC International Conference on Performance Engineering (ICPE '21 Companion), April 19–23, 2021, Virtual Event, France.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3447545.3451185

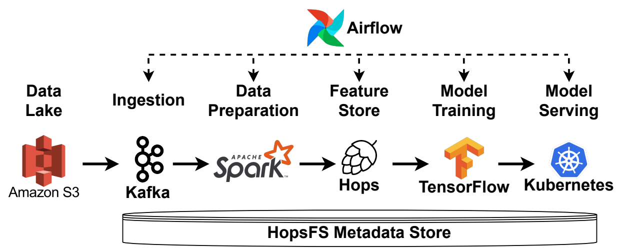


Figure 1: Example ML workflow based on Hopsworks [21].

1 INTRODUCTION

We live in the golden age of machine learning (ML) systems. ML-powered technologies are enabling breakthroughs in many areas, including healthcare (e.g., medical image analysis [36], precision agriculture [12], image processing [41], and autonomous vehicles [53]). With further innovations, in the coming decade we aim to address one of the grand challenges in the field - to reach human-level intelligence and cognitive abilities in each area [15, 20]. For this to happen, the systems able to run ML processes need to become much more capable than today, and in particular need to support increasingly more sophisticated ML workflows (e.g., as Figure 1 depicts). This raises the problem of systematically increasing performance and efficiency, a problem for which we have addressed only narrow aspects. Enlarging the scope of previous work in performance, which has focused mainly on the ML training step [17, 22, 50], in this work we envision holistic performance analysis for ML workflows.

The main motivation for this work is the rapid progress of ML processes, from ad-hoc setups of only train-then-use steps, to complex workflows integrating many steps. In a typical ML setup, a large set of data is acquired and processed to build effective ML models. In the past decade, ML model training has received much attention, becoming increasingly faster and more efficient through the use of better algorithms [60], systems architecture [18], and new classes of hardware [56]. However, in the state-of-practice at the beginning of the 2020s, ML model training is merely *one* step out of the large ML workflow consisting of *many steps* that orchestrate the emerging complexities of the ML life cycle.

To support an easy integration and automation of ML life cycle as a workflow, recently many ML workflow management systems have been proposed [21, 35, 61], e.g., Amazon SageMaker[1], Microsoft

[1]https://aws.amazon.com/machine-learning/

Azure Machine Learning[2], Google Cloud AutoML[3], MIT NorthStar [32], and the Ease.ml service [33]. These services are part of the bigger *Continuous Integration/Continuous Development (CI/CD)* trend in ML workflows [28]. These systems provide a unified interface and language for developers to integrate, schedule, and execute multi-framework ML workflow orchestration in a distributed setting. Typical steps include many (i.e., at least eight [2, 33]) and varied activities with significantly different performance profiles than training, including data acquisition [49], valuation [14], cleaning [8], validation [47], model training [60], model selection [31], model deployment [5], inference [10], and continuous integration [6].

As this new generation of ML workflow systems emerges and matures, it needs increasingly better support for performance reasoning and modeling to operate efficiently and to scale. We identify two key trends: *First, a push for closely integrated, continuously optimized ML workflows:* ML workflow systems have simplified the development and deployment of complex ML workflows, so experts are pushing for fast, automated, real-time model updates [4, 6]. To address these needs, systems specialists push for more closely integrated, CI/CD, ML workflows. As the ML model training time is optimized (i.e., decreased), and as more and more complex steps are integrated in ML workflows, the non-training steps of the ML workflows rapidly become the dominant factor in determining the performance of the ML workflows. Hence, it is vital to understand the end-to-end performance implications of various ML operations.

Second, a push for cloud and multi-tenancy. The focus of early ML systems, such as TensorFlow and PyTorch, was to build standalone frameworks for ML model training. However, more recently we have witnessed a push for scalable, multi-tenant, cloud-ready deployments of ML training services and workflows [24, 25, 33, 63]. These systems promise to better utilize ML-oriented hardware (large DRAMs, accelerators) by executing on them multiple workloads with varying resource needs concurrently (multiplexing). Naturally, in this setting *performance interference* (during resource allocation and execution) from other tenants can influence the performance of both the entire ML workflow and individual steps, which might be running with strict deadlines to supporting ML QoS [29]. Approaches for reducing performance interference in multi-tenant BigData workloads have proven successful [38, 57, 62], so we expect ML to become more cloud-native as well.

In this work, we argue that, as the accessibility and affordability of ML workflows increases, we need *holistic* performance analysis tools for users to: (i) collect performance traces and data; (ii) analyze them in real time; (iii) support light-weight, high-performance integration within the ML workflow system. However, building any such tool presents four main challenges:

- **(C1) Holistic view of execution:** ML workflow systems include diverse data processing frameworks with their own performance characteristics and metrics, which makes obtaining a global view of workflow execution difficult.
- **(C2) Overhead:** Rich performance traces are desirable to provide deep insight into ML workflow performance, but obtaining performance traces without introducing prohibitive performance overhead remains a major challenge.
- **(C3) High-level results:** Performance tracing produces large volumes of low-level performance data. However, a typical ML user does not have the expertise required to interpret this data. A big challenge for a performance analysis tool is automatically analyzing low-level performance data to derive high-level insight into performance for users.
- **(C4) Integration:** Due to the diversity of data processing frameworks and environments encountered in ML workflow systems, integrating seamlessly with any workflow system, without requiring users to complete a complex setup and configuration process, is a difficult task.

In this paper, we pose the question: *How to ensure holistic performance analysis for ML workflows?* We present our vision for a *holistic framework* that can help to analyze the performance of CI/CD ML workflows in a systematic manner by addressing each of the four main challenges. Our main contributions are:

1. A qualitative and quantitative analysis of a selected set of popular performance analysis tools in ML (in Section 2) to make a case for a new approach to performance analysis for ML workflows. We select and conduct a preliminary analysis of two tools from the state-of-practice. We also select and analyze several state-of-the-art instruments.
2. We formulate our vision on holistic performance analysis for ML workflows (Section 3). We synthesize four key pillars in the design of a performance analysis tool and discuss their design challenges: a unified execution model, collecting execution traces, data aggregation and representation, and integration in ML systems.
3. We propose first steps towards GradeML, a holistic performance analysis tool for ML workflows (Section 4). We start from Grade10 [19], our work on holistic performance analysis for graph processing, and propose extensions to implement our vision. Our preliminary work and experiments are open source: https://github.com/atlarge-research/grademl

2 MOTIVATION

In this section, we analyze existing performance analysis tools for ML, which mostly focus on model training and inference, and more broadly for data processing systems. We discuss the limitations that inspired our vision for a new, holistic approach to performance analysis for ML workflow system.

2.1 State-of-the-Practice

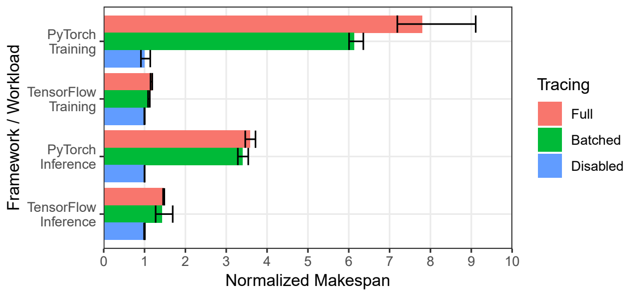
Modern machine learning frameworks, such as Tensorflow and PyTorch, include dedicated performance analysis tools.These tools collect comprehensive execution traces of ML applications, i.e., logs of every operation performed by the ML framework, including timestamps, durations, and other metadata. These traces can be visually inspected to determine which operations are most time-consuming, whether the application is bottlenecked on data processing or network communication, etc. However, in practice these tracing tools impose significant performance overhead.

We conduct a preliminary experiment with example training (MNIST) and inference (ImageNet) workloads using the PyTorch and TensorFlow frameworks in a local cluster [3]. Experiment

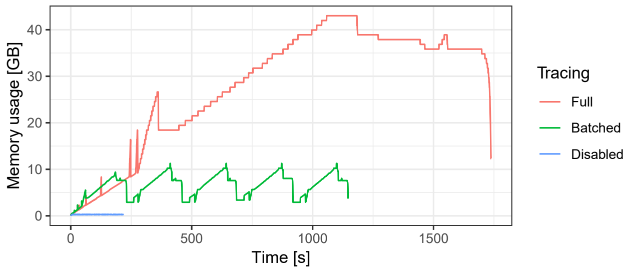
[2]https://docs.microsoft.com/en-us/azure/machine-learning/
[3]https://cloud.google.com/automl/

scripts, data, and details of our setup can be found on GitHub[4]. Figure 2a summarizes the observed overhead of tracing. It depicts the *normalized* makespan of the training and inference workloads when not tracing (*Disabled*), when recording a single trace for the entire workload (*Full*), and when recording the entire workload as a sequence of traces (e.g., one trace per training epoch or per inference batch, *Batched*). Makespans are normalized per workload, such that the average duration without tracing is 1. We observe that recording a single trace incurs the largest slowdowns, on average 1.17× for TensorFlow and 7.79× for PyTorch for model *training*, and on average 1.47× for TensorFlow and 3.59× for PyTorch for model *inference*. The overhead can be reduced by tracing a workload in multiple, smaller steps, but remains significant enough that tracing ML workloads in a production environment is not feasible.

Figure 2b depicts memory usage over time when training a small model in PyTorch. We observe that when not tracing, this workload's memory usage is at most 0.32 GB. Recording a single trace causes memory usage to peak at 43 GB, an increase of 140×. This memory overhead can be reduced by recording multiple, shorter traces, with a peak memory usage of 11.3 GB, or 37× our baseline for this application. The high memory overhead of tracing further limits the usability of these tools, especially when system memory is needed to train large models, or for inference workloads in memory-constrained environments such as edge devices.



(a) Normalized makespan for example training and inference workloads on PyTorch and Tensorflow.



(b) Memory usage of PyTorch during model training.

Figure 2: Overhead of tracing in example ML training and inference workloads.

2.2 State-of-the-Art

To overcome the limited use of tracing as provided by ML frameworks, the community has proposed several approaches towards a better understanding of ML performance. XSP [34] proposes across-stack profiling for in-depth characterization of the performance of deep neural network (DNN) inference workloads. Daydream [66] provides insight into DNN training performance by predicting the impact of potential hardware and software optimizations. Tian et al. [54] propose performance diagnosis and prediction for dataflow applications, including DNN training, by combining execution profiling and resource usage inference. Justus et al. [27] train a neural network to predict DNN training times, taking non-linear activities like data loading and non-optimal parallel execution into account.

There are also several distribution models that are built on top of existing ML frameworks that include performance analysis tools. For example, Horovod [50], GPipe [22] and PipeDream [17] extend frameworks like TensorFlow and Pytorch with data, model and pipeline parallel execution for ML applications. They include tools to do automatic performance tuning in the form of model partitioning, device placement and extended hyperparameter optimizations.

Benchmarks are another tool that can be used to better understand a system's capability to do ML. By executing a ML benchmark with support for varying neural networks, datasets and distribution models implemented on a variety of frameworks, one can get a better understanding of the performance characteristics of ML workloads and the interaction with underlying hardware. Examples include DAWNbench [9], MLPerf [40], Deep500 [7], HPC AI500 [26], and DDLBench [23].

These performance analysis tools all extend state-of-the-practice solutions meaningfully, but either focus on one system in the ML workflow or on a single metric across multiple systems. Therefore it is key for a holistic performance analysis framework to adopt and combine these tools to provide better insight into ML performance.

2.3 Performance Analysis in Data Processing

The broader field of data processing applications includes a variety of specialized approaches for performance analysis. For example, Grade10 [19] models the execution and resources of graph processing applications to correlate low-level performance issues with application-level execution stages. Ousterhout et al. [45] propose blocked time analysis for identifying network and storage bottlenecks in Spark applications. G^2 [16] provides a high-level query language for correlating events in distributed execution graphs. Various other approaches [11, 13, 55] collect and summarize low-level metrics for complex big data systems, like the Hadoop stack. These approaches were not designed for ML systems, but may be integrated in a larger performance analysis tool to analyze common data processing frameworks found in ML workflows.

Finally, performance analysis approaches for distributed systems or even single machines are commonly used to study data processing applications. CPU profilers [30, 48, 51], GPU profilers [43, 44], distributed resource monitoring systems [1, 58], and distributed tracing systems [37, 52, 64] collect a variety of raw performance data, each with their own trade-offs between performance overhead and completeness of data. Many of these systems and concepts are already used to *manually* diagnose the performance of ML workloads, but integration in a specialized performance analysis tool is necessary to automatically analyze and interpret the collected performance data in the context of ML workflow systems.

[4]https://github.com/atlarge-research/grademl

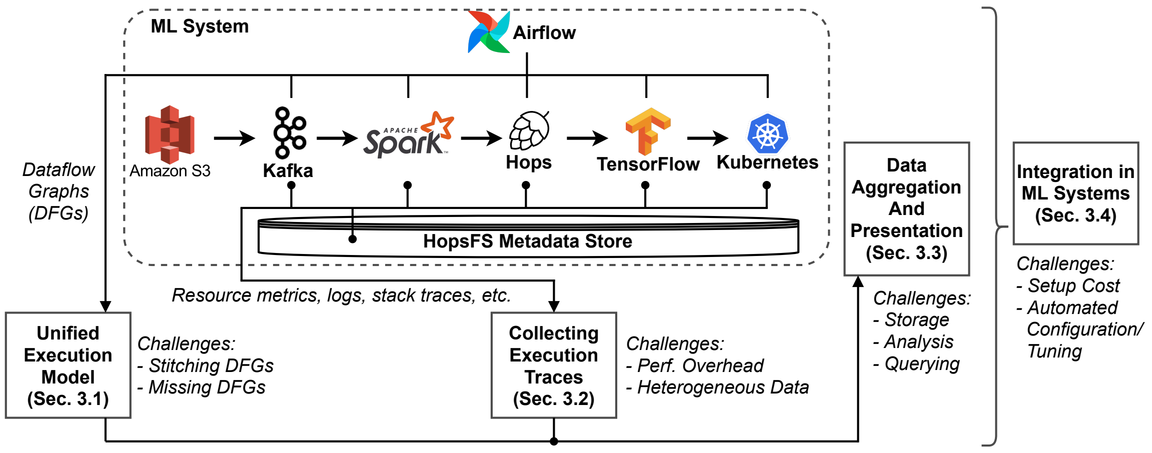


Figure 3: Vision for holistic performance analysis for ML workflows, including four key pillars and associated challenges.

3 VISION FOR HOLISTIC PERFORMANCE ANALYSIS FOR ML WORKFLOWS

In this section, we present our vision for a holistic performance analysis tool for ML workflows. We envision that a non-expert user should be able to collect and aggregate performance data with low overhead, and derive from this data high-level insight into their workflow's performance to answer questions such as: Which steps in the workflow have the biggest impact on end-to-end performance? How would increasing the resource allocation of one framework impact performance? Are expensive resources used efficiently throughout the workflow? We synthesize four pillars in the design of a holistic performance analysis tool for ML workflows, derived from our four main challenges (Section 1), limitations of existing approaches (Section 2), and our experience designing Grade10 [19], a holistic performance analysis tool for graph processing systems. For each pillar, we identify additional sub-challenges and discuss approaches to address these challenges. We summarize our vision, its pillars, and associated challenges in Figure 3.

3.1 Unified Execution Model

Typical ML workflow systems use a diverse set of data processing frameworks for different steps in a ML workflow. Analyzing the performance of each framework independently makes reasoning about their connections and collective performance difficult. Instead, addressing challenge **C1**, we propose a *unified execution model* to capture the different data processing steps and frameworks involved in a ML workflow. Such a model should allow for performance analysis tasks (e.g., detecting bottlenecks, load imbalance) to be framework-agnostic, thus avoiding costly specialization of every performance analysis task for every potential framework that may be used in a ML workflow. A unified execution model should also abstract away the implementation details of individual frameworks so users of the performance analysis tool do not need to learn the inner workings of each framework they use to understand the performance analysis outcomes presented to them.

A common approach for expressing the execution of data processing frameworks is through dataflow graphs, i.e., the set of tasks performed by an application and their data dependencies [54]. Dataflow graphs are used internally by many data processing frameworks (e.g., TensorFlow, PyTorch, Spark) to schedule the execution of tasks, so these graphs can be readily exported for use in performance analysis. Dataflow graphs also match well how users define their workflows in modern ML systems [5, 33]. Users define the high-level operations in their ML workflow and connect the outputs of some operations to the inputs of others to form a graph of operations. Thus, we propose a unified execution model based on dataflow graphs, by connecting the dataflow graphs of each constituent framework in a ML workflow. To build such a model, we must address two key challenges: (1) how to combine dataflow graphs of different frameworks into a single dataflow graph, and (2) how to obtain dataflow graphs for frameworks that do not use or expose such a graph?

First, the dataflow graphs of data processing frameworks capture only the operations within one framework. Stitching these dataflow graphs together into a unified model is non-trivial and requires inserting connections to represent data flowing between different data processing frameworks. Thus, we need to identify which data processing frameworks are connected by dataflows, and which operations in each framework act as the *source* and *sink* of a cross-framework dataflow. The existence of dataflows can be derived from the user-provided workflow definition, as it already encodes all frameworks present in the ML workflow and their data dependencies. Identifying the source and sink operations *within each framework* is more difficult, but we propose an approach based on inspecting network traffic to infer the timing of dataflows, which may be correlated with each framework's internal dataflow graph to identify active operations at the time a dataflow occurred.

Second, dataflow graphs may not be available for each framework in the ML workflow. A simple approach is to treat such a framework as a single black-box operation, but this severely limits reasoning about the performance of individual operations within a framework. Alternatively, a dataflow graph would have to be constructed. This can be done manually, e.g., by a framework developer or expert through source code annotation, or automatically, e.g., through source code analysis. Existing performance analysis approaches based on dataflow graphs assume the availability of a dataflow graph [54, 66] or facilitate manual source code annotation [19]. The feasibility of automatically constructing program flow graphs has been demonstrated in other applications [42, 64], but determining if these approaches can provide high-quality dataflow graphs for ML workflows remains an open challenge.

3.2 Collecting Execution Traces

Having discussed constructing a unified execution model describing the high-level operation of a ML workflow, we now pay attention to collecting execution traces for understanding the low-level performance characteristics of ML workflows. An execution trace describes the execution of an application through a variety of low-level performance data, such as resource consumption metrics, application logs, and program stack traces. Efficiently collecting rich execution traces (**C2**) for ML workflows imposes several challenges: (1) how to obtain low-level performance data without introducing significant performance overhead, and (2) how to combine performance data collected from heterogeneous systems and data sources into a holistic execution trace describing ML workflow performance?

When collecting execution traces from production ML systems, tracing overhead must be minimal to avoid any significant slowdown of ML workflows executing within the system. As demonstrated in our preliminary experiments with TensorFlow and PyTorch (Section 2.1), built-in tracers suffer from significant performance overhead. However, other tracing and monitoring approaches for distributed systems promise more lightweight data collection. For example, modern logging frameworks reduce the overhead of recording (performance) events [59, 65]. Distributed monitoring systems [1, 58] efficiently collect resource metrics in large clusters. Lightweight CPU profilers [30, 51] periodically collect stack traces to effectively estimate which parts of an application's code consume most time. Although less comprehensive than the complete application traces collected by TensorFlow/PyTorch, we believe – supported by related performance analysis approaches [19, 54] – that a combination of lightweight event logging, resource monitoring, and profiling provides sufficiently rich execution traces.

Execution traces collected from a multitude of sources need to be combined to form a holistic view of ML workflow performance. Combining these traces is non-trivial due to the heterogeneity of data sources. For example, data sources may use different identifiers to refer to the same entities (OS-level vs. application-level identification of processes, hostname- vs IP-based identification of machines), timestamped data may suffer from clock skew due to non-synchronized local clocks, and data may be collected at different time granularities. Existing approaches for combining execution traces in data processing systems provide general solutions that may be applicable to ML workflows. We propose an approach based on Stitch [46], which automatically extracts relationships between entities and identifiers from heterogeneous application logs and traces.

3.3 Data Aggregation and Presentation

Any performance analysis tool is only as good as its ability to inform the user about its findings. For system-level performance analysis tools, such as CPU profilers [30, 48, 51], presenting raw, low-level performance data is common and acceptable. However, when analyzing the performance of complex ML workflow systems, it is not feasible for users to manually inspect the large volumes of raw performance data that will be collected. Also, an average data scientist using a ML system is unlikely to have the expertise required to interpret low-level performance data for every data processing framework used throughout their workflow. To overcome these issues, many existing approaches for performance analysis in dataflow systems [11, 13, 19, 45] automate the collection and analysis of performance data and present a high-level view on performance (e.g., based on the dataflow graph) that users can inspect or query. Based on these approaches, we identify three challenges in allowing users of a holistic performance analysis tool to access performance data and derive insight into the performance of their ML workflows (**C3**): (1) performance data should be combined and made accessible through a centralized system, (2) large volumes of performance data should be analyzed automatically to derive high-level performance observations, and (3) users should be able to query performance data to answer specific questions about their workflow's performance.

By default, every system in a ML workflow will store their execution traces, in the form of metrics, profiles and logs, in a different location. This makes it difficult to combine performance data from different systems and to present users with a holistic view of their workflow's performance. A centralized storage system enables easy access and better integration for performance data. As the bulk of performance data will be timestamped (e.g., resource utilization metrics, application logs), we propose a time-series database as a suitable candidate for a centralized storage system.

Automated analysis of performance data is a necessity for a tool collecting large volumes of performance data. Key to such automation is defining what kinds of analysis to perform, i.e., what kinds of performance issues to look for and what kinds of aggregate performance data to compute. A common approach is picking a set of well-known performance issues to look for, e.g., resource bottlenecks [45], or workload imbalance [19]. However, a more comprehensive approach is needed to cover the diversity of performance issues that we expect to see in ML workflow systems. In particular, we propose an approach based on Daydream [66], which allows for analysis of arbitrary what-if scenario's based on dataflow graphs (to be extended to our unified execution model). Execution of complex analysis tasks can be performed using one of many (distributed) data processing frameworks, or be embedded into the ML system to use already available data processing facilities [39].

In complex, heterogeneous systems like ML workflow systems, performance cannot exclusively be captured in a single metric of interest. Different components of the system exhibit different performance characteristics over time, and experience performance issues that may be independent or intertwined, concurrent or disjoint. To understand not only the performance of the system, but also the performance of individual components or system resources, users must be able to query performance data for their ML workflows. A basic approach could allow users to select parts of the unified execution model they are interested in and be presented with a summary of relevant performance characteristics and identified performance issues [19]. A more comprehensive approach could allow users to write custom queries using a language that designed for correlating timestamped events and metrics [16, 37].

3.4 Integration in ML Systems

A typical user of a ML system cannot be expected to have any expertise in deploying or configuring data processing frameworks. Instead, when selecting the data processing frameworks to use in a ML workflow, users are likely to use whatever frameworks are available to them, e.g., frameworks explicitly supported by their ML workflow management system [32, 33]. Because of this, it is important for a performance analysis tool for ML workflows to adapt to and seamlessly integrate with existing ML and data processing frameworks instead of the other way around (**C4**). This introduces two challenges: how to design a performance analysis tool that (1) requires minimal expertise and user input to set up, and (2) automatically configures and tunes itself once deployed?

A performance analysis tool that requires intricate knowledge of ML system internals to set up and use is not feasible for an average user. Furthermore, if a performance analysis tool requires extensive changes to the systems it needs to analyze, it cannot be readily deployed in managed or shared environments. Instead, setting up a performance analysis tool should require minimal changes to the existing ML system, and should require minimal user input. This can be accomplished by, e.g., supporting out-of-the-box many popular data processing frameworks, automatically identifying the components present in an ML system (from a central source like the workflow orchestrator/scheduler), and using dynamic code injection to insert instrumentation into existing systems [37, 42].

Any complex tool, including a holistic performance analysis tool for ML workflows, has many parameters that can be tuned to alter the accuracy or performance of the tool. For example, the sampling frequency of resource monitoring or CPU profiling tools offers a trade-off between accuracy (i.e., being able to observe rapid changes) and performance overhead (i.e., spending too much CPU time on data collection). Users of a performance analysis tool cannot be expected to invest time into configuring and tuning their tool, so the tool should configure and tune itself, automatically. As a first approach, our experience with Grade10 shows that sensible defaults can be found experimentally or through expertise. Automatically tuning a performance tool for for a particular ML system is left as the next research challenge.

4 ONGOING WORK: GRADEML

We are working on realizing our vision for holistic performance analysis for ML workflows by designing and implementing a novel performance analysis tool, GradeML. We start from Grade10 [19], our previous work on performance analysis for graph processing systems. Grade10 captures the execution of a graph processing application in a hierarchical execution model, and collects coarse-grained hardware and software resource consumption in a resource model. It combines both models using a resource attribution process to identify resource bottlenecks for individual steps in an application's execution model. Grade10 also identifies performance issues found in graph processing, such as workload imbalance.

Grade10 addresses some initial challenges laid out in Section 3. However, it has multiple limitations that we are addressing to create GradeML, our extension of Grade10 for ML systems. For example, Grade10's execution model strongly resembles the unified execution model proposed in Section 3.1, but we are working on adding support for stitching together multiple dataflow graphs.

For execution trace collection, Grade10 includes a lightweight resource monitoring component. Grade10 also supports heterogeneous data sources, but currently requires hand-written rules to combine them. Due to the large variety of data processing frameworks, hand-written rules may be feasible for a proof-of-concept tool, but for production use we are exploring an automated approach based on Stitch [46].

Grade10's existing support for data aggregation and presentation needs to be extended in several directions. Grade10 assumes an external process collects relevant performance data. In GradeML's design, performance data is automatically collected in a centralized time-series database managed by GradeML. Automated performance analysis is partially supported by Grade10, but needs a more comprehensive approach to support the variety of performance issues likely to exist in ML workflows. We plan to include simulation of what-if scenarios in GradeML's design, inspired by Daydream [66]. Finally, Grade10 allows users to query it for top performance issues and resource bottlenecks for any part of an application's execution model. We do not currently plan for GradeML to extend this capability towards supporting arbitrary queries about performance using a custom query language, as discussed in Section 3.3.

Finally, Grade10 was not designed for ML and thus does not integrate natively with ML workflow management systems. For GradeML we consider closely the challenges of setup complexity and tuning in the design of every component, especially for dataflow graph and performance data collection (e.g., using dynamic instrumentation instead of requiring custom system binaries, empirically identifying sensible defaults for key parameters).

5 CONCLUSION

With the popularity of machine learning systems and their application in many areas, significant research effort has been invested in making model training faster and more efficient. However, ML model training is only one part of a larger ML workflow, which ranges from data acquisition and cleaning to model inference and continuous integration. In this work, we addressed this large conceptual gap by formulating our vision for holistic performance analysis for ML workflows, which has inspired our ongoing work on a new performance analysis tool, GradeML.

We conducted an analysis on state-of-the-practice and state-of-the-art work on machine learning and data processing. We formulated our vision for a holistic performance analysis tool for ML workflows through four design pillars: a unified execution model, lightweight collection of performance data, efficient data aggregation and presentation, and close integration in ML systems. As future work, we propose first steps towards building GradeML as implementation of our vision. We start from Grade10, our work on performance analysis for graph processing systems, and explain how it can be extended to meet our vision. Our preliminary work and experiments are open source:

https://github.com/atlarge-research/grademl

ACKNOWLEDGMENTS

Work supported by NWO projects MagnaData and OffSense.

REFERENCES

[1] Agelastos et al. 2014. The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications. In *SC*.

[2] Amershi et al. 2019. Software Engineering for Machine Learning: A Case Study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*.

[3] Bal et al. 2016. A Medium-Scale Distributed System for Computer Science Research: Infrastructure for the Long Term. *IEEE Computer* (2016).

[4] Banerjee et al. 2020. Challenges and Experiences with MLOps for Performance Diagnostics in Hybrid-Cloud Enterprise Software Deployments. In *2020 USENIX Conference on Operational Machine Learning (OpML 20)*.

[5] Baylor et al. 2017. Tfx: A tensorflow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[6] Baylor et al. 2019. Continuous Training for Production ML in the TensorFlow Extended (TFX) Platform. In *2019 USENIX Conference on Operational Machine Learning (OpML 19)*.

[7] Ben-Nun et al. 2019. A modular benchmarking infrastructure for high-performance and reproducible deep learning. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*.

[8] Chu et al. 2016. Data cleaning: Overview and emerging challenges. In *SIGMOD*.

[9] Coleman et al. 2017. Dawnbench: An end-to-end deep learning benchmark and competition. *Training* 100 (2017).

[10] Crankshaw et al. 2017. Clipper: A Low-Latency Online Prediction Serving System. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*.

[11] Dai et al. 2011. HiTune: Dataflow-Based Performance Analysis for Big Data Cloud. In *ATC*.

[12] Gadiraju et al. 2020. Multimodal Deep Learning Based Crop Classification Using Multispectral and Multitemporal Satellite Imagery. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery &; Data Mining*.

[13] Garduño et al. 2012. Theia: Visual Signatures for Problem Diagnosis in Large Hadoop Clusters. In *Strategies, Tools , and Techniques: Proceedings of the 26th Large Installation System Administration Conference, LISA 2012, San Diego, CA, USA, December 9-14, 2012*.

[14] Ghorbani and Zou. 2019. Data shapley: Equitable valuation of data for machine learning. *arXiv preprint arXiv:1904.02868* (2019).

[15] Yolanda Gil and Bart Selman. 2019. A 20-Year Community Roadmap for Artificial Intelligence Research in the US. arXiv:1908.02624 [cs.CY]

[16] Guo et al. 2011. G2: A Graph Processing System for Diagnosing Distributed Systems. In *ATC*.

[17] Harlap et al. 2018. Pipedream: Fast and efficient pipeline parallel dnn training. *arXiv preprint arXiv:1806.03377* (2018).

[18] He et al. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

[19] Hegeman et al. 2020. Grade10: A Framework for Performance Characterization of Distributed Graph Processing. In *CLUSTER*.

[20] Hestness et al. 2019. Beyond Human-Level Accuracy: Computational Challenges in Deep Learning. In *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*.

[21] Hopsworks. 2021. Hopsworks. https://www.hopsworks.ai/.

[22] Huang et al. 2019. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in Neural Information Processing Systems*.

[23] Jansen et al. 2020. DDLBench: Towards a Scalable Benchmarking Infrastructure for Distributed Deep Learning. In *ISC*.

[24] Jayaram et al. 2019. FfDL: A Flexible Multi-Tenant Deep Learning Platform. In *Proceedings of the 20th International Middleware Conference*.

[25] Jeon et al. 2019. Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads. In *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference*.

[26] Jiang et al. 2020. Hpc ai500: The methodology, tools, roofline performance models, and metrics for benchmarking hpc ai systems. *arXiv preprint arXiv:2007.00279* (2020).

[27] Justus et al. 2018. Predicting the computational cost of deep learning models. In *2018 IEEE International Conference on Big Data (Big Data)*.

[28] Karlaš et al. 2020. Building Continuous Integration Services for Machine Learning. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery &; Data Mining*.

[29] Kim and Lee. 2020. Reducing Tail Latency of DNN-Based Recommender Systems Using in-Storage Processing. In *Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems*.

[30] Knüpfer et al. 2008. The Vampir Performance Analysis Tool-Set. In *Tools for High Performance Computing - Proceedings of the 2nd International Workshop on Parallel Tools for High Performance Computing, July 2008, HLRS, Stuttgart*.

[31] Kotthoff et al. 2017. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *The Journal of Machine Learning Research* 18 (2017).

[32] Kraska. 2018. Northstar: An interactive data science system. *Proceedings of the VLDB Endowment* 11 (2018).

[33] Li et al. 2018. Ease.Ml: Towards Multi-Tenant Resource Sharing for Machine Learning Workloads. *Proc. VLDB Endow.* 11 (2018).

[34] Li et al. 2019. Across-Stack Profiling and Characterization of Machine Learning Models on GPUs. *CoRR* abs/1908.06869 (2019).

[35] Lim et al. 2019. MLOp Lifecycle Scheme for Vision-based Inspection Process in Manufacturing. In *2019 USENIX Conference on Operational Machine Learning (OpML 19)*.

[36] Litjens et al. 2017. A survey on deep learning in medical image analysis. *Medical Image Analysis* 42 (2017).

[37] Mace et al. 2015. Pivot Tracing: Dynamic Causal Monitoring for Distributed Systems. In *SOSP*.

[38] Mace et al. 2015. Retro: Targeted Resource Management in Multi-tenant Distributed Systems. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*.

[39] Mai et al. 2020. KungFu: Making Training in Distributed Machine Learning Adaptive. In *OSDI*.

[40] Mattson et al. 2019. Mlperf training benchmark. *arXiv preprint arXiv:1910.01500* (2019).

[41] Mayer and Jacobsen. 2020. Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques, and Tools. *ACM Comput. Surv.* 53 (2020).

[42] Mirgorodskiy et al. 2008. Diagnosing distributed systems with self-propelled instrumentation. In *Middleware*.

[43] NVIDIA. 2021. NVIDIA Nsight. https://developer.nvidia.com/tools-overview.

[44] NVIDIA. 2021. NVIDIA Profiler. https://docs.nvidia.com/cuda/profiler-users-guide/index.html.

[45] Ousterhout et al. 2015. Making Sense of Performance in Data Analytics Frameworks. In *NSDI*.

[46] Pi et al. 2018. Profiling distributed systems in lightweight virtualized environments with logs and resource metrics. In *HPDC*.

[47] Polyzotis et al. 2019. Data validation for machine learning. *Proceedings of Machine Learning and Systems* 1 (2019).

[48] Reinders. 2005. VTune performance analyzer essentials. *Intel Press* (2005).

[49] Salloum et al. 2017. A Survey of Text Mining in Social Media: Facebook and Twitter Perspectives. *Advances in Science, Technology and Engineering Systems Journal* 2 (2017).

[50] Sergeev and Del. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799* (2018).

[51] Shende and Malony. 2006. The TAU Parallel Performance System. *IJHPCA* 20 (2006).

[52] Sigelman et al. 2010. Dapper, a large-scale distributed systems tracing infrastructure. (2010).

[53] Tian et al. 2018. DeepTest: Automated Testing of Deep-Neural-Network-Driven Autonomous Cars. In *Proceedings of the 40th International Conference on Software Engineering*.

[54] Tian et al. 2019. Towards Framework-Independent, Non-Intrusive Performance Characterization for Dataflow Computation. In *Proceedings of the 10th ACM SIGOPS Asia-Pacific Workshop on Systems*.

[55] Wang et al. 2012. VScope: Middleware for Troubleshooting Time-Sensitive Data Center Applications. In *Middleware 2012 - ACM/IFIP/USENIX 13th International Middleware Conference, Montreal, QC, Canada, December 3-7, 2012. Proceedings*, Vol. 7662.

[56] Wang et al. 2019. Benchmarking TPU, GPU, and CPU platforms for deep learning. *arXiv preprint arXiv:1907.10701* (2019).

[57] Wang et al. 2020. Metis: learning to schedule long-running applications in shared container clusters at scale. In *SC*.

[58] Yang et al. [n.d.]. End-to-end I/O Monitoring on a Leading Supercomputer. In *NSDI*.

[59] Yang et al. 2018. Nanolog: A nanosecond scale logging system. In *ATC*.

[60] You et al. 2018. Imagenet training in minutes. In *ICPP*.

[61] Zaharia et al. 2018. Accelerating the Machine Learning Lifecycle with MLflow. *IEEE Data Eng. Bull.* 41 (2018).

[62] Zhang et al. 2014. MIMP: Deadline and Interference Aware Scheduling of Hadoop Virtual Machines. In *CCGrid*.

[63] Zhang et al. 2020. Model-Switching: Dealing with Fluctuating Workloads in Machine-Learning-as-a-Service Systems. In *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*.

[64] Zhao et al. 2016. Non-Intrusive Performance Profiling for Entire Software Stacks Based on the Flow Reconstruction Principle. In *OSDI*.

[65] Zhao et al. 2017. Log20: Fully automated optimal placement of log printing statements under specified overhead threshold. In *SOSP*.

[66] Zhu et al. 2020. Daydream: Accurately Estimating the Efficacy of Optimizations for DNN Training. *arXiv preprint arXiv:2006.03318* (2020).