Model Governance: Reducing the Anarchy of Production ML

Vinay Sridhar, Sriram Subramanian, Dulcardo Arteaga, Swaminathan Sundararaman, Drew Roselli, and Nisha Talagala, *ParallelM*

https://www.usenix.org/conference/atc18/presentation/sridhar

This paper is included in the Proceedings of the
2018 USENIX Annual Technical Conference (USENIX ATC '18).

July 11–13, 2018 • Boston, MA, USA

ISBN 978-1-939133-02-1

Model Governance: Reducing the Anarchy of Production ML

Vinay Sridhar Sriram Subramanian Dulcardo Arteaga
Swaminathan Sundararaman Drew Roselli Nisha Talagala
ParallelM

Abstract

As the influence of machine learning grows over decisions in businesses and human life, so grows the need for Model Governance. In this paper, we motivate the need for, define the problem of, and propose a solution for Model Governance in production ML. We show that through our approach one can meaningfully track and understand the who, where, what, when, and how an ML prediction came to be. To the best of our knowledge, this is the first work providing a comprehensive framework for production Model Governance, building upon previous work in developer-focused Model Management.

1 Introduction

Machine Learning (ML) and Deep Learning (DL) have recently made tremendous advances in algorithms, analytic engines, and hardware. However, production ML deployment is still nascent [17]. While production deployments are always challenging, ML generates unique difficulties [21, 7, 22]. We focus on the governance challenge: the management, diagnostic, compliance, and regulatory implications of production ML models. With recent demands for explainable/transparent ML [9, 3, 16, 5], the need to track provenance and faithfully reproduce ML predictions is even more serious. Given the strong data-dependent nature of ML/DL, even small changes in configurations can have unexpected consequences in predictions, making Governance critical to production ML.

Previous research has focused on *Model Management*: managing these models and enabling efficient reuse by developers [28, 25, 20]. Production deployment further complicates governance with i) complex topologies with retraining, ii) continuous inference programs that run in parallel with (re)training programs, iii) actions (such as model approvals) that need to be recorded for auditing, vi) model rollbacks that may occur in real time, and v) heterogeneous and distributed environments.

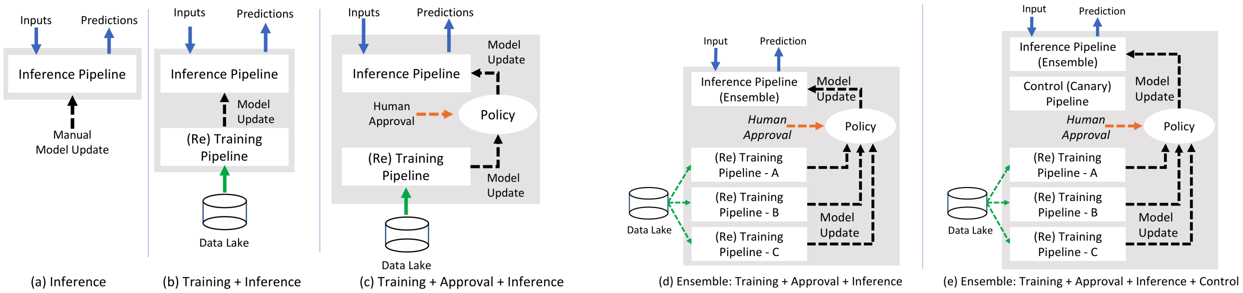
We define Production Model Governance as the ability to determine the creation path, subsequent usage, and consequent outcomes of an ML model, and the use of this information to accomplish a range of tasks including reproducing and diagnosing problems and enforcing compliance. In this paper, we propose and motivate a generic solution approach that can be adapted across different governance usage examples.

Our goal is to highlight the Model Governance problem and propose solutions. Our contributions are: i) we propose a definition for Production Model Governance and its necessary inclusive elements; ii) We propose a two-layer model for Governance. The lower layer contains each pipeline as a DAG and tracks everything (such as features, datasets, and code) similar to what is proposed by prior research. We add a second layer directed graph where each pipeline or policy invocation is a node and edges represent cross-pipeline, policy, and human action dependencies. We temporally track and correlate both of these levels to comprehensively cover Model Governance; iii) Using this model, we build a production Model Governance system that supports heterogeneous frameworks (currently, Spark, TensorFlow, Flink); iv) We propose a robust approach to a wide range of possible Governance applications via generic access to Governance metadata; and v) At the pipeline level, we expand upon prior research by illustrating a generic API-based instrumentation approach across analytic engines.

2 Motivation

Machine learning algorithms execute as ”pipelines”, which ingest data (via batch data lakes, stream brokers, etc.) and compute (feature engineering, model training, scoring, inference, etc.). Pipelines may run on engines (Spark, Flink, etc.) or as standalone programs.

To highlight the importance of Model Governance, we use an example medical application that leverages ML to recommend to a doctor which tests to run on a patient. The calling application sends user information to an ML prediction pipeline which returns a prediction. Figure 1 shows several examples of how this simple scenario can



(a) Inference (b) Training + Inference (c) Training + Approval + Inference (d) Ensemble: Training + Approval + Inference (e) Ensemble: Training + Approval + Inference + Control

Figure 1: **Evolution of ML Pipelines in production.** *(a) The prediction pipeline is executing in production and is using a model trained offline and uploaded. (b) A more dynamic scenario where the model itself is retrained on a schedule. (c) Yet another sophistication, where newly trained models undergo an approval process prior to production deployment. (d) Here an ensemble model is used for prediction (requiring each sub-model to be trained individually). Finally, (e) shows the scenario if a control pipeline (*canary*) is used in production to ensure that the primary prediction pipeline is behaving stably. The control pipeline could also be running a surrogate model to improve explainability.*

be put into production. While the basic function requires only one pipeline, once you add the need to improve accuracy via re-training, the need for human approvals, and state-of-the-art models, the complexity grows rapidly. We now illustrate sample Governance scenarios for the example in Figure 1(e).

Scenario 1: Say we needed to know why a certain patient was recommended a CT-Scan while another patient was not. For each recommendation, we would need to answer: Which model(s) were running in the ensemble? Which code was executing the models? When/How was each model trained (using which configurations and which features)? Which model provided the control pipeline and would its recommendation have differed? Which operator approved each of the models in the primary pipeline? Who approved the model in the control pipeline? Were any errors noted in this time frame? Can both predictions be reproduced in order to test for bias?

Scenario 2: Assume a data scientist wishes to leverage some of the models for a new production ML application. They would want to know under which circumstances the existing models were generated, as well as which datasets and features were used. These may also be required for production approval.

3 Model Governance

We define Production Model Governance as the ability to determine the creation path, subsequent usage, and consequent outcomes of an ML model, and the use of this information in various ways, as illustrated above. Given the wide range of usages, we believe any Model Governance solution should include:

Provenance/Lineage: For any ML prediction, the ability identify the exact sequence of events (datasets, trainings, code, pipelines, human approvals) that led to the event.

Reproducibility: The ability to replay the above sequence and replicate the prediction, thereby setting the context to investigate alternatives.

Audit and Compliance: The ability to evaluate all ML operations in an organization and determine compliance with regulations.

Leverage: The ability to reuse past ML work (such as algorithms, models, features) to determine whether the derived object is appropriate for the new usage.

Scale and Heterogeneity: The ability to work with many models, pipelines, and varied analytic engines and languages in a distributed setting such as Cloud/Edge.

Multiple Governance Metadata Usages: The ability to multi-purpose the metadata. For example, Data Scientists may analyze experiments or reuse models. Operators may diagnose issues, help address bias concerns, or ensure compliance to policies.

4 Approach and Design

Our solution must support simple to complex topologies, parallel pipelines, streaming and batch pipelines, pipelines changing state mid-run as new models arrive, policy actions and relationships between non-overlapping pipeline runs. For these we must provide: (i) Sufficient dependency information to infer provenance; (ii) Configuration including code and input parameters for reproducibility; (iii) A durable record of all metadata; (iv) Metadata from disjoint pipeline, possibly from different analytic engines and languages; (v) Metadata provenance, trends, and policy analysis and beyond.

4.1 The Intelligence Overlay Network

Core to our design is a two layer model we call the Intelligence Overlay Network (ION). An ION is a logical model that connects objects such as pipelines, policy execution modules, and messages between them. The first level of an ION is inspired by the traditional graph-based modeling of message passing parallel programs, where each node is a execution element and directed edges are messages between programs (allowing for cycles) [1]. In our case, execution nodes are ML pipelines or policy actions, and messages (such as ML models or events) are passed between them. At the second level of the ION, each execution node can itself be a DAG of components.

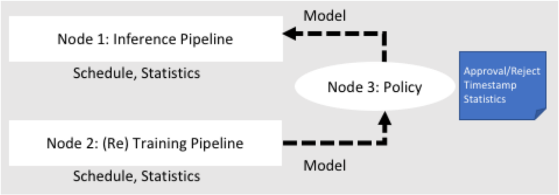


Figure 2: **ION for Flow in Figure 1(c)**

An ML pipeline is a single execution node in an ION. Figure 2 shows how Figure 1(c)'s pattern maps to an ION. The Inference and Training nodes are pipelines, the approval is a Policy node, and the edges show the path of a model. Figure 3 shows how the graph in Figure 2 is tracked by our system across time.

We apply this approach to ML workflows: (i) All execution elements are nodes. Nodes execute on a schedule (batch), continuously (streaming), or are event triggered; (ii) Nodes can pass messages and each message and send/receive events are recorded; (iii) Within each node, a DAG can be defined with its stages monitored.

The ION approach delivers important benefits. First, many ML pipelines map easily to the ION nodes as DAGs. Statistics from these pipelines are gathered as time-series variables that support Governance scenarios, for example, by providing required transparency to complex feature selection within a pipeline.

Second, the ION graph cleanly captures the dependencies and interactions between the pipelines, including their repeated executions over time, their connections to human actions, and their relationships to each other.

Finally, the combined statistics of both levels enables powerful usages, such as (a) tracking pipeline metrics like *confusion matrices* across multiple training runs, (b) comparing Models across multiple training runs, (c) tracking the approval actions of any human operator and cross-checking those against the performance of inference pipelines that ran the approved models.

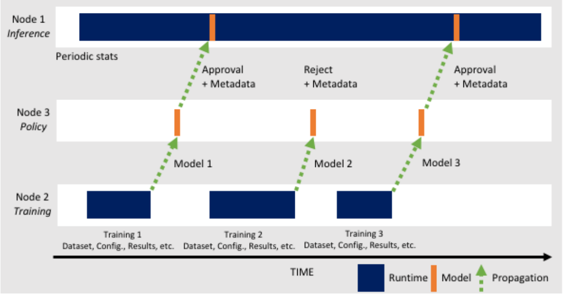


Figure 3: **ION Timeline**

4.2 System Design

We use an agent/server architecture. The agents run on each instance of an analytic engine (for example, a 15 node Spark cluster would have a single agent). Each agent communicates with its engine via standard interfaces. The server receives Governance metadata from the agents and interlinks the information via the ION. Currently our server also runs the policy execution code blocks. The server maintains a metadata database and manages garbage collection.

While recent Model Management approaches have been passive [28], we chose an active Agent based approach to enable disconnected operation (a common occurrence in Edge based ML environments [26]). During disconnections, the agent saves information locally and transmits it when connectivity is restored. If the agent is disconnected for long periods of time and runs out of local storage resources, some information can be lost. The agent/server architecture also enables scale and support of heterogeneous analytic engines.

This approach requires no changes to the analytic engines. We can also connect to existing analytic engines and can share analytic engines with other programs that we do not monitor. Any program that already works in these engines works in this environment. An Agent can also support custom standalone programs. The required changes to the programs themselves, for all the cases, are discussed in Section 4.3.

Figure 4 shows the database schema. The Level 1 schema includes tracked objects within a pipeline. The Level 2 schema captures the ION pattern as well as specific elements of each ION instance. Contextual information (such as which machines a particular pipeline ran on) is also captured. All objects are timestamped. All objects link to an ION and from the ION to each other.

4.3 Information Import and Export

ML pipelines exchange Governance metadata with our system in three ways (see Figure 5). First, a JSON-based ION definition which contains links to pipeline code is uploaded to our server. Second, each pipeline is instrumented via an API library to provide runtime metadata. Pipelines can export time series variables, digests, configuration, models, etc. Supported Model formats are PMML, SavedModel or opaque. Since our system also supports opaque model formats, other industry standard model formats (like ONNX) that we do not yet interpret can be immediately used. Third, for standard pipelines and models, we auto-extract metadata (like [25]).

The API library has import and export capabilities. On the export side, running pipelines send metadata to our Governance system. On the import side, running pipelines can query the database for stored metadata and perform additional analytics (see Section 4). The library implementation is engine-specific and to date we have implemented our API library for Spark, Flink, and Tensorflow and in Scala, Java, and Python.

Unlike prior approaches [25], we employ both a fully declarative approach (where the developer decides what to instrument) and automatic extraction wherever standardized pipelines and model formats allow. While train-

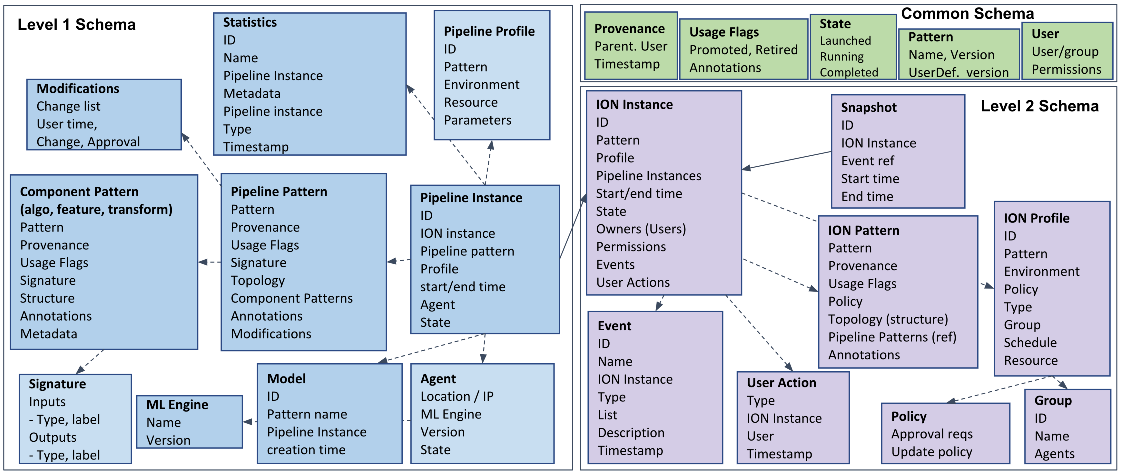
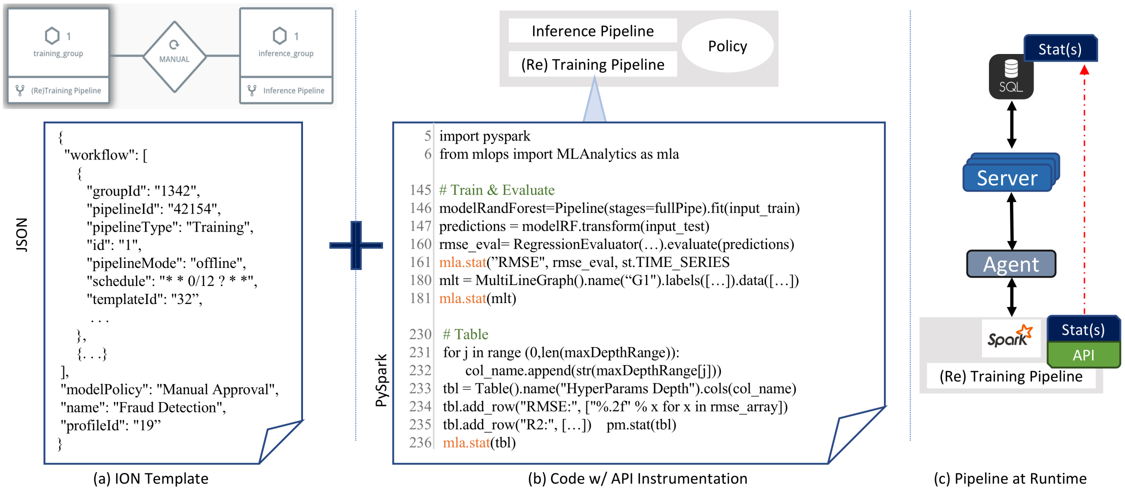


Figure 4: **Governance Schema**



```
{
 "workflow": [
    {
      "groupId": "1342",
      "pipelineId": "42154",
      "pipelineType": "Training",
      "id": "1",
      "pipelineMode": "offline",
      "schedule": "* * 0/12 ? * *",
      "templateId": "32",
       . . .
    },
    {. . .}
 ],
 "modelPolicy": "Manual Approval",
 "name": "Fraud Detection",
 "profileId": "19"
}
```

(a) ION Template

```
import pyspark
from mlops import MLAnalytics as mla

# Train & Evaluate
modelRandForest=Pipeline(stages=fullPipe).fit(input_train)
predictions = modelRF.transform(input_test)
rmse_eval= RegressionEvaluator(…).evaluate(predictions)
mla.stat(”RMSE", rmse_eval, st.TIME_SERIES
mlt = MultiLineGraph().name(“G1").labels([…]).data([…])
mla.stat(mlt)

 # Table
 for j in range (0,len(maxDepthRange)):
      col_name.append(str(maxDepthRange[j]))
 tbl = Table().name("HyperParams Depth").cols(col_name)
 tbl.add_row("RMSE:", ["%.2f" % x for x in rmse_array])
 tbl.add_row("R2:", […])    pm.stat(tbl)
 mla.stat(tbl)
```

(b) Code w/ API Instrumentation

(c) Pipeline at Runtime

Figure 5: **Interaction of ML Pipelines with our Governance System**

ing pipelines can be quite structured (like SparkML), production inference pipelines have not standardized on any pipeline or code structure, running the gamut from auto-generated (via JPMML inference) to latency optimized hand-coded programs. This range of inference pipeline structures renders approaches that rely only on automatic extraction impractical for generalized production usage. With our approach, even custom written standalone programs can be instrumented for Governance.

4.4 Correlation and Causality

IONs are used to create a coherent time view of all metadata by (a) merging the views of multiple concurrent pipelines or policy blocks and (b) relating different executions of the same logical node (e.g., re-training).

Causality: Within each ION node, we assume monotonically increasing timestamps. Across nodes within an ION, causality is established via messages.

Inter-Node Correlation: Across ION nodes, we assume correlation based on synchronized timestamps. Provenance is derived directly from messages and not from time correlation. Correlation is only used if a single sequence of statistics from the ION is retrieved for visualization. We have not used logical clocks to date because the governance usages we have worked with can be addressed sufficiently with the approaches above.

4.5 Governance Usages

Model Governance metadata has multiple uses. Any pipeline can use the API library to extract and compute additional analytics on governance metadata. For example, a pipeline can analyze all models approved by a user and correlate with training accuracy metrics or do additional model selection or model improvement. Compliance enforcement can be done via functions that periodically review recent metadata and confirm adherence to specific polices (such as human model approval or training accuracy thresholds for production deployed models). The metadata information can be used for advanced optimizations such as Meta-Learning. The applications of this approach are vast and to date we have only used our system to explore simple usages. Due to lack of space, we do not elaborate on this capability further in this paper but it is an area of our future work.

Object	Occurrence	Count	Size
Models	multiple	24	48KB
Statistics	multiple	536K	242MB
Signatures	multiple	1.8K	136MB
Pipelines	once	3	14KB
Node	once	3	1KB
Events	multiple	84	140KB
Logs	multiple	264	74MB
Total	N/A	5̃38K	452MB

Table 1: **Governance information captured from the three node ION after a day of execution.**

5 Evaluation

A governance solution should include lineage, reproducibility, audit, leverage, scale, heterogeneity, and multiple usages (see Section 3). We conduct a simple experiment to illustrate lineage, audit, scale, and heterogeneity in our solution. Additionally, we measure the overheads of Governance metadata gathering, including instrumentation and model propagation.

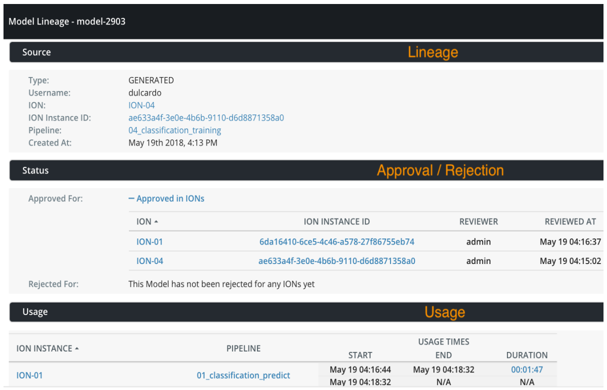
Our experiment consists of a three node ION: a Spark training pipeline, a Flink inference pipeline, and a policy module configured to always approve models (Figure 1c). The training pipeline is batch Logistic Regression with hyperparameter search in PySpark training a new model every hour. The selected model propagates via the Policy node to the Flink streaming inference in Scala. Governance metadata gathered includes input and prediction signatures (histograms), ML statistics, code, libraries, etc. ML statistics are any items reported via the API, including generic metrics such as accuracy, precision, recall, etc. and algorithm-specific metrics. Hyperparameter search configuration and result information is also reported via the API.

Figure 6 shows the dashboard view of a model generated in the ION. The governance dashboard provides the lineage information for every model generated (or uploaded), the training pipeline and the parameters used to generate the selected model, model approval information and the subsequent usage of the selected model in inference pipelines. Each hyperlink within the dashboard provides in-depth information on ION template, pipeline, configuration, statistics, etc. These in-depth views are not included due to lack of space.

Table 1 summarizes the list of objects that are captured in our system during the ION's execution. The overheads of our API instrumentation was less then 1% in this experiment and model propagation delay on average was around 3s for models of size 2KB. We were able to both replay and reproduce predictions using our system.

6 Related Work

The related work closest to ours includes [28, 25, 20]. We add several fundamental contributions over each. First we define and implement the ION model which links multiple pipelines and policy actions, including a full governance metadata and schema. Second, while both we and [25] integrate with multiple analytic engines, our approach adds a fully declarative instrumentation approach to the base auto-extraction approach, enabling our system to work with changing analytic engines and standalone ML programs. Finally, we have designed and built an import/export function where programs can access and compute additional analytics on Governance metadata, enabling a vast range of Governance usages which can themselves be tracked and managed. Additional model management approaches have been presented in [18]. Production ML challenges have been described in [21, 7, 22, 24, 4, 15, 8]. Meta-learning is discussed in [23, 13]. Overviews of analytic engines, libraries, and model serving systems are in [19, 10, 11, 12, 27, 2, 29, 6, 14].



Model Lineage - model-2903

Source

Type: GENERATED
Username: dulcardo
ION: ION-04
ION Instance ID: ae633a4f-3e0e-4b6b-9110-d6d8871358a0
Pipeline: 04_classification_training
Created At: May 19th 2018, 4:13 PM

Status

Approved For: – Approved in IONs

ION ▴	ION INSTANCE ID	REVIEWER	REVIEWED AT
ION-01	6da16410-6ce5-4c46-a578-27f86755eb74	admin	May 19 04:16:37
ION-04	ae633a4f-3e0e-4b6b-9110-d6d8871358a0	admin	May 19 04:15:02

Rejected For: This Model has not been rejected for any IONs yet

Usage

ION INSTANCE ▴	PIPELINE	USAGE TIMES		
		START	END	DURATION
ION-01	01_classification_predict	May 19 04:16:44	May 19 04:18:32	00:01:47
		May 19 04:18:32	N/A	N/A

Figure 6: **Model Governance Page:** This page highlights the lineage, approval/rejection status across IONs, and usage of the selected model across IONs.

7 Discussion and Future Work

Our system addresses most of the goals laid out in Section 3. We can track any prediction (or other event) to all related pipelines, datasets, execution configurations, code and human actions, and reproduce the functional steps. Via both tracking of input dataset pathnames and dataset signatures, we identify which data was used to generate which model. Using the API library, a user can write a pipeline to extract and analyze any information in the database to evaluate compliance or do audits. As another example, programs using our API can analyze data trends and correlate them with model outcomes.

For future work, we plan to explore the possibilities of meta-learning via Governance metdata and expand on how Governance relates to AI explainability. We want to couple the provenance data with explainable ML to help answer the question of not just how/where/what/when but also why ML decisions were made.

References

[1] The message passing parallel programming model, 1995.

[2] Apache. flink. Internet draft, 2017.

[3] Gdpr online, 2017. http://www.eudgpr.org.

[4] Michelangelo: Uber's machine learning platform. Internet draft, 2017.

[5] New york bill on algorithmic bias, 2017. http://www.businessinsider.com/algorithmic-bias-accountability-bill-passes-in-new-york-city-2017-12.

[6] ABADI, M., AND ET AL. Tensorflow: Large-scale machine learning on heterogeneous systems. Internet draft, 2015.

[7] BAYLOR, D., BRECK, E., CHENG, H.-T., FIEDEL, N., FOO, C. Y., HAQUE, Z., HAYKAL, S., ISPIR, M., JAIN, V., KOC, L., KOO, C. Y., LEW, L., MEWALD, C., MODI, A. N., POLYZOTIS, N., RAMESH, S., ROY, S., WHANG, S. E., WICKE, M., WILKIEWICZ, J., ZHANG, X., AND ZINKEVICH, M. Tfx: A tensorflow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2017), KDD '17, ACM, pp. 1387–1395.

[8] BRECK, E., CAI, S., NIELSEN, E., SALIB, M., AND SCULLEY, D. Whats your ml test score? a rubric for ml production systems. reliable machine learning in the wild. *Neural Information Processing Systems (NIPS) Workshop.* (2016).

[9] BURT, A. Is there a 'right to explanation' for machine learning in the gdpr. Internet draft, 2017.

[10] CHEN, T., LI, M., LI, Y., LIN, M., WANG, N., WANG, M., XIAO, T., XU, B., ZHANG, C., AND ZHANG, Z. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems.

[11] CRANKSHAW, D., BAILIS, P., GONZALEZ, J. E., LI, H., ZHANG, Z., FRANKLIN, M. J., GHODSI, A., AND JORDAN, M. I. The missing piece in complex analytics: Low latency, scalable model management and serving with velox. *CoRR abs/1409.3809* (2014).

[12] CRANKSHAW, D., WANG, X., ZHOU, G., FRANKLIN, M. J., GONZALEZ, J. E., AND STOICA, I. Clipper: A low-latency online prediction serving system. *CoRR abs/1612.03079* (2016).

[13] FEURER, M., SPRINGENBERG, J. T., AND HUTTER, F. Initializing bayesian hyperparameter optimization via meta-learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence* (2015), AAAI'15, AAAI Press, pp. 1128–1135.

[14] FIEDEL, N. Serving models in production with tensorflow serving, 2017.

[15] FLAUONAS, I. Beyond the technical challenges for deploying machine learning solutions in a software company. Internet draft, 2017.

[16] GARFINKEL, S., MATTHEWS, J., SHAPIRO, S., AND SMITH, J. Towards algorithmic transparency and accountability. *Communications of the ACM, Vol. 60 No. 9, Page 5* (2016).

[17] INSTITUE, M. G. Artificial intelligence: The next digital frontier?, 2017.

[18] KUMAR, A., MCCANN, R., NAUGHTON, J., AND PATEL, J. M. Model selection management systems: The next frontier of advanced analytics. *SIGMOD Rec. 44*, 4 (May 2016), 17–22.

[19] MENG, X., BRADLEY, J., YAVUZ, B., SPARKS, E., VENKATARAMAN, S., LIU, D., FREEMAN, J., TSAI, D., AMDE, M., OWEN, S., XIN, D., XIN, R., FRANKLIN, M. J., ZADEH, R., ZAHARIA, M., AND TALWALKAR, A. Mllib: Machine learning in apache spark. *J. Mach. Learn. Res. 17*, 1 (Jan. 2016), 1235–1241.

[20] MIAO, H., LI, A., DAVIS, L. S., AND DESHPANDE, A. Towards unified data and lifecycle management for deep learning. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)* (April 2017), pp. 571–582.

[21] PIERRE ANDREWS, ADITYA KALRO, H. M. A. S. Productionizing machine learning pipelines at scale. *Machine Learning Systems Workshop at ICML.* (2016).

[22] POLYZOTIS, N., ROY, S., WHANG, S. E., AND ZINKEVICH, M. Data management challenges in production machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data* (New York, NY, USA, 2017), SIGMOD '17, ACM, pp. 1723–1726.

[23] SCHELTER, S., SOTO, J., MARKL, V., BURDICK, D., REINWALD, B., AND EVFIMIEVSKI, A. Efficient sample generation for scalable meta learning. In *2015 IEEE 31st International Conference on Data Engineering* (April 2015), pp. 1191–1202.

[24] SCULLEY, D., HOLT, G., GOLOVIN, D., DAVYDOV, E., PHILLIPS, T., EBNER, D., CHAUDHARY, V., YOUNG, M., CRESPO, J.-F., AND DENNISON, D. Hidden technical debt in machine learning systems. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2* (Cambridge, MA, USA, 2015), NIPS'15, MIT Press, pp. 2503–2511.

[25] SEBASTIAN SCHELTER, JOOS-HENDRIK BOESE, J. K. T. K. S. S. Automatically tracking metadata and provenance of machine learning experiments. *Workshop on ML Systems at NIPS (MLSys) Workshop.* (2017).

[26] TALAGALA N., SUNDARARAMAN S., S. V. A. D. L. Q. S. S. G. S. R. D. K. S. Eco: Harmonizing edge and cloud with ml/dl orchestration. In *Proceedings of USENIX HotEdge 2018* (2018).

[27] VANSCHOREN, J., VAN RIJN, J. N., BISCHL, B., AND TORGO, L. Openml: networked science in machine learning. *CoRR abs/1407.7722* (2014).

[28] VARTAK, M., SUBRAMANYAM, H., W-E., L., VISWANATHAN, S., AND MADDEN, S. Demonstration of modeldb: a system for managing machine learning models. *Neural Information Processing Systems (NIPS) Workshop.* (2016).

[29] ZAHARIA, M., XIN, R. S., WENDELL, P., DAS, T., A. M. D. A. M. X. R. J. V. S. F. M. G. A. G. J. S. S., AND STOICA, I. Apache spark: a unified engine for big data processing. *Commun. ACM 59* (2016), 56–65.