# Towards Complaint-driven ML Workflow Debugging

Anonymous Author(s)

## 1 INTRODUCTION

Machine learning (ML) is increasingly a core part of a company's technical infrastructure, yet introduces considerable complexity. It requires creating, executing, and managing ML workflows that perform data extraction, labeling, model design and (re-)training, inference, and using the predictions as part of downstream analytics or applications. To facilitate this adoption, numerous ML platforms (e.g., TFX [2], Michelangelo [14], FBLearner Flow [5], Overton [11], MLFlow [10], and others [12]) have been proposed to manage and automate parts of the ML life-cycle.

As a running example, CompanyX (name anonymized) is a company that manages email campaigns for their enterprise customers. They train models to estimate user characteristics (e.g., churn likelihood, product affinities, etc). This empowers their customers to use user attributes and these models to define and monitor user cohorts used in email campaigns. Figure 1 shows the ML workflow that trains a churn rate prediction model and uses it to count the number of active users likely to churn, as well as the associated query that performs inference using the trained model.

Unfortunately, the power of ML—that it adapts the model to training data—also presents a risk due to the increased difficulty of debugging when errors arise. Debugging ML workflows requires reasoning about the correctness of the workflow logic, the input datasets, the models, and interactions between them. In the CompanyX example, a customer is alarmed that the size of their user cohort dropped considerably and wants to understand why. Yet there can be a multitude of reasons: the query may be incorrect, there may be errors in the `Users` or `Logins` tables, the model may be misspecified, and there could be errors in the training data. In short, developers *are not just debugging the code, but also the data.*

Traditional workflow and query debugging tools are capable of helping debug query [4] and query data (e.g. the `Users` table in CompanyX example) [9, 15] errors, however, they do not typically address data errors that affect ML models. To solve this problem, modern ML platforms provide ways to check the correctness of the data. For instance, they can detect schema and data type mismatches [3, 13], and provide methods to define and detect dataset distribution shifts between training, test, and/or production [3]. Further, systems such as [13] support "data unit tests" where developers can provide user-defined functions that compose a list of primitive statistical data-checks. Unfortunately, these approaches both place considerable burden on the developer to specify checks, and are limited in their efficacy for several reasons.

First, it is a tall order to ask developers to specify the data constraints that will guarantee stable performance of their downstream ML workflow. Even if an ML application developer knows the key performance indicators for his application it may be difficult for him to specify which features are critical to maintain their stability. For example, CompanyX's engineers may have an insight that large cohort size drops in a single day are strong indications of data-related errors. While, to directly specify this as a constraint over the workflow output of Figure 1, it's not clear how they can

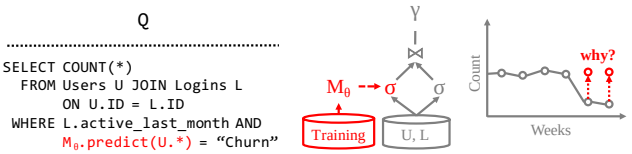


**Figure 1: CompanyX workflow and output visualization where the user specifies surprising output values. Training and model inference steps in red.**

specify which feature distribution shifts are important for each of their clients.

Second, is that these approaches primarily focus on *detecting* that a constraint was violated, but do not provide direct support to debug the workflow and identify *why* the violation occurred. This latter functionality is important because in comparison to data constraint violations, workflow constraints involve ML inferences and are thus harder to debug (see Section 2.1). For example, once the CompanyX customer noticed the cohort drop, it is useful to understand that it was due to a corrupt subset of training data records that introduced bias in the model.

Third, is that errors (e.g., corrupt values, distribution shifts, missing data) in the data may not cause noticeable errors in the downstream workflow results. As a hypothetical example, the training data for the churn prediction model may contain less spammer users while they become frequent during serving, yet does not introduce errors that affect the high *life time value* (LTV) user cohort in ways that matter to the customers. In these cases, generic data checks may generate many false positive alarms that reduce the likelihood that true errors are responded to [3].

## 2 COMPLAINT-DRIVEN WORKFLOW DEBUGGING

To address the above limitations, this paper advocates for a *complaint-oriented* approach towards specifying and debugging data errors in ML workflows. The approach takes as input *complaints* specified as constraints over the final or intermediate outputs of workflows that *use* trained ML models. In addition, debuggers may specify constraints, such as the operators or datasets in the workflow that they suspect may have caused constraint violations. The approach then outputs explanations in the form of specific operator(s) or data subsets, and how they may be changed to address the constraint violations. There are a number of benefits of this approach.

**Easier Specification:** Complaints are akin to traditional software bug reports that ask the user to describe the problem that they encountered. However, in this case, they are expressed as logical statements over data records. In the CompanyX example, the constraint is simply that the cohort size should > 150.

Further, complaints can help different stakeholders collaboratively debug an ML workflow. This is because different teams are

often responsible for subsets of a ML workflow—a data modeling team may focus on model design, an ML engineering team implements and manages the training and deployment of the models, and end users execute queries that use the model predictions. Similarly, a given model may be used in multiple analysis workflows—different customers make use the same ML models to define different user cohorts and performance metrics. Each of these people have visibility into a portion of the full ML workflow, yet can contribute their domain expertise to provide more accurate workflow debugging.

**Useful Weak Signals:** Complaints can provide weak signals that can help debugging even in the absence of labels during serving. For example, label noise during training could be easily detected if we had access to the labels during serving. While exact serving labels might be unknown, violation of application specific constraints like the ones in CompanyX's case can be used to detect label noise phenomena. Another prominent case is debugging ML workflows involving multiple models. Once again, the correct intermediate predictions in a multi-model ML workflow may be unknown. However, violations of constraints on their aggregate outcome is a strong indication that at least one serving prediction is incorrect and thus they can be used to initiate debugging. The same applies to workflows that involve multiple predictions by a single model.

**Enabling MLOps debugging:** An automated tool for debugging of complaints would be powerful even from an organizational perspective. The burden of monitoring the ML workflow usually behooves members of MLOps teams. However, MLOps engineers may lack the familiarity with the data, code or models used for a specific ML workflow. Thus they are usually restricted to the detection of potential bugs and need to delegate the debugging of any finding to other teams. Unsure of the potential causes, they may need to notify all teams involved in the workflow. Given an automated debugging tool, the MLOps team can identify the part of the workflow responsible for the violation and hand off to the relevant team.

## 2.1 Major Challenges

Despite the benefits complaint driven debugging can bring, it also carries new challenges that are not obvious to solve.

**Complaint Ambiguity** For a given complaint there might not be a singular way on how to address it. For example, if the a client of CompanyX complaints about the sudden drop of his user cohort size as in Figure 1, changing any of the negative churn predictions for an active user to a positive one would address, at least partially, the client's complaint. We call complaints that can be resolved in multiple ways *ambiguous*. Handling complaints that are ambiguous allows us to make use of the weak signals complaints provide.

**Component Heterogeneity** Machine learning workflows often contains multiple components that may be written in a variety of libraries or even languages, and bear different properties like differentiability and continuity that can help debugging [8, 16]. ML workflow debugging should be able to handle all the components simultaneously.

**Component Entanglement** Reasoning about the interactions of different components, model based or not, is also a challenge. A special case is debugging complaints involving multiple models. In this case, fixes for each model can no longer be reasoned about independently. As an example, a client of CompanyX could define a selection predicate that involves predictions from two or more models. The same complaint issued on top of the new workflow is even more ambiguous as it is unclear which model caused the cohort size drop.

**Computational Complexity** Often, individual components inside an ML workflow already exhibit high computational complexity (e.g. table joins, feature extractions), let alone the ML workflow as a collection of complex components. Given that ML workflow debugging should be an interactive process, debugging tools must take computation complexity into consideration.

While complaint driven debugging is not new neither in the software engineering community [7] nor in the ML community [16], complaint driven debugging on ML workflow remains unsolved due to these challenges. We believe solving these challenges is the key to an effective yet efficient ML workflow debugging tool.

# 3 PRELIMINARY RESULTS FROM RAIN

Taking these challenges into consideration, we present Rain, a complaint-driven debugging system for relational workflows that leverage ML predictions (as in Figure 1). Users can execute an SPJA query, where some of the expressions perform ML inference, and specify value or existential constraints over attribute values in the final or intermediate result sets. The system leverages techniques from relational query provenance [1, 6] and ML interpretation [8] to identify the set of training records that would most address the constraint violations if they were deleted.

We conducted an array of experiments spanning various datasets, training data corruptions, models, queries and complaints. To simulate training data corruptions, given a corruption predicate and a corruption severity percentage $\alpha$, $\alpha$% of the training records that satisfy the given predicate have their training labels flipped to an incorrect label. A model trained on the corrupted data is then used as part of a relational workflow. Simulating a user observing a surprising query result, we input in Rain a complaint. For table outputs the complaint can specify that a tuple should not exist in the result. For numerical ones a complaint may specify the correct value or a direction (greater, smaller). Rain returns a ranking where training records ranked higher are deemed more likely to be corrupted. We evaluated Rain both in terms of recall and precision.

Our experimental results consistently show that complaints can be a powerful tool. A single complaint over an aggregate result can be as effective at recovering corrupted training records as specifying the labels of hundreds of individual serving predictions. This suggests that complaints can save developers significant amount of effort during debugging. At the same time, our experiments show that complaints need not be very precise to work. When for example training corruptions have caused an aggregate result to be lower than its true value, a complaint does not need to specify the exact correct value of the aggregate to be effective. A complaint that just specifies that the aggregate should be higher can recover a substantial number of corrupted training records. Additionally, even when a single complaint is not enough to identify all the training corruptions, leveraging multiple complaints can yield results that are better than using each complaint alone.

# REFERENCES


[1] Y. Amsterdamer, D. Deutch, and V. Tannen. Provenance for aggregate queries. In M. Lenzerini and T. Schwentick, editors, *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece*, pages 153–164. ACM, 2011.

[2] D. Baylor, E. Breck, H. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc, C. Y. Koo, L. Lew, C. Mewald, A. N. Modi, N. Polyzotis, S. Ramesh, S. Roy, S. E. Whang, M. Wicke, J. Wilkiewicz, X. Zhang, and M. Zinkevich. TFX: A tensorflow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, pages 1387–1395. ACM, 2017.

[3] E. Breck, N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich. Data validation for machine learning, 2019.

[4] A. Chapman and H. Jagadish. Why not? In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 523–534. ACM, 2009.

[5] Facebook. Introducing fblearner flow: Facebook's ai backbone. https://engineering.fb.com/core-data/introducing-fblearner-flow-facebook-s-ai-backbone/, 2016. [Online; accessed 14-January-2020].

[6] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In L. Libkin, editor, *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China*, pages 31–40. ACM, 2007.

[7] D. Jackson and M. Vaziri. Finding bugs with a constraint solver. In D. J. Richardson and M. J. Harold, editors, *Proceedings of the International Symposium on Software Testing and Analysis, ISSTA 2000, Portland, OR, USA, August 21-24, 2000*, pages 14–25. ACM, 2000.

[8] P. W. Koh and P. Liang. Understanding black-box predictions via influence functions. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1885–1894. PMLR, 2017.

[9] A. Meliou and D. Suciu. Tiresias: the database oracle for how-to queries. In K. S. Candan, Y. Chen, R. T. Snodgrass, L. Gravano, and A. Fuxman, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 337–348. ACM, 2012.

[10] MLflow. Mlflow - a platform for the machine learning lifecycle. https://mlflow.org/, 2019. [Online; accessed 14-January-2020].

[11] C. Ré, F. Niu, P. Gudipati, and C. Srisuwananukorn. Overton: A data system for monitoring and improving machine-learned products. *CoRR*, abs/1909.05372, 2019.

[12] S. Schelter, J.-H. Böse, J. Kirschnick, T. Klein, and S. Seufert. Automatically tracking metadata and provenance of machine learning experiments. In *Machine Learning Systems workshop at NIPS*, 2017.

[13] S. Schelter, D. Lange, P. Schmidt, M. Celikel, F. Bießmann, and A. Grafberger. Automating large-scale data quality verification. *PVLDB*, 11(12):1781–1794, 2018.

[14] Uber. Meet michelangelo: Uber's machine learning platform. https://eng.uber.com/michelangelo/, 2019. [Online; accessed 14-January-2020].

[15] E. Wu and S. Madden. Scorpion: Explaining away outliers in aggregate queries. *PVLDB*, 6(8):553–564, 2013.

[16] X. Zhang, X. Zhu, and S. J. Wright. Training set debugging using trusted items. In S. A. McIlraith and K. Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 4482–4489. AAAI Press, 2018.