Characterizing and Detecting Mismatch in Machine-Learning-Enabled Systems

Grace A. Lewis, Stephany Bellomo, Ipek Ozkaya
Carnegie Mellon Software Engineering Institute
Pittsburgh, PA USA
{glewis, sbellomo, ozkaya}@sei.cmu.edu

***Abstract*—Increasing availability of machine learning (ML) frameworks and tools, as well as their promise to improve solutions to data-driven decision problems, has resulted in popularity of using ML techniques in software systems. However, end-to-end development of ML-enabled systems, as well as their seamless deployment and operations, remain a challenge. One reason is that development and deployment of ML-enabled systems involves three distinct workflows, perspectives, and roles, which include data science, software engineering, and operations. These three distinct perspectives, when misaligned due to incorrect assumptions, cause ML mismatches which can result in failed systems. We conducted an interview and survey study where we collected and validated common types of mismatches that occur in end-to-end development of ML-enabled systems. Our analysis shows that how each role prioritizes the importance of relevant mismatches varies, potentially contributing to these mismatched assumptions. In addition, the mismatch categories we identified can be specified as machine readable descriptors contributing to improved ML-enabled system development. In this paper, we report our findings and their implications for improving end-to-end ML-enabled system development.**

***Index Terms*—software engineering, machine learning, software engineering for machine learning, model engineering**

I. Introduction

Despite advances in frameworks for machine learning (ML) model development and deployment, integrating models into production systems still remains a challenge [7] [9] [15] [19] [26]. One reason is that the development and operation of ML-enabled systems involves three perspectives, with three different and often completely separate workflows and people: the data scientist builds the model; the software engineer integrates the model into a larger system; and then operations staff deploy, operate, and monitor the ML system. These perspectives often operate separately, using different processes and vocabulary referring to similar concepts, leading to opportunities for mismatch between the assumptions made by each perspective with respect to the elements of the ML-enabled system, and the actual guarantees provided by each element. This problem is exacerbated by the fact that system elements evolve independently and at a different rhythm, which could over time lead to unintentional mismatch. Examples of mismatch and their consequences include (1) poor system performance because computing resources required to execute the model are different from computing resources available during operations, (2) poor model accuracy because model training data is different from operational data, (3) development of large amounts of glue code because the trained model input/output is incompatible with operational data types, (4) system failure due to inadequate testing because developers were not able to replicate the testing that was done during model training, and (5) monitoring tools are not set up to detect diminishing model accuracy, which is the evaluation metric defined for the trained model.

We therefore define **ML Mismatch** as a problem that occurs in the development, deployment, and operation of an ML-enabled system due to incorrect assumptions made about system elements by different stakeholders (*i.e.,* data scientist, software engineer, operations) that results in a negative consequence. ML mismatch can be traced back to information that could have been shared between stakeholders that would have avoided the problem.

The target objective of our study is to develop a set of machine-readable descriptors for system elements, as a mechanism to enable mismatch detection and prevention in ML-enabled systems. The goal of these descriptors is to codify attributes of system elements and therefore make more of the ML model and system development assumptions explicit. The descriptors can be used by system stakeholders in a manual way, for information awareness and evaluation activities; and by automated mismatch detectors at design time and run time for cases in which attributes lend themselves to automation. Immediate benefits of these descriptors include:

- They serve as checklists as ML-enabled systems are developed.
- They provide system stakeholders with examples of information to request or requirements to provide to teams.
- They enable identification of attributes for which automated detection is feasible and define new software components for ML-enabled systems that perform mismatch detection.

The research questions therefore defined for this study are:

- RQ1: What are common types of mismatch that occur in the end-to-end development of ML-enabled systems?
- RQ2: What are best practices for documenting data, models, and other system elements that will enable detection of ML mismatch?
- RQ3: What are examples of ML mismatch that could be detected in an automated way, based on the codification of best practices in machine-readable descriptors for ML

system elements?

The focus of this paper is to report on the first phase of the study addressing RQ1, which is the results of practitioner interviews and survey that were conducted to gather examples of real ML mismatches and their consequences. Section II presents the study design and Section III shows the results. In Section IV we discuss findings and analysis insights. Section V outlines Phase 2 of our study on the path towards automated mismatch detection. Finally, Section VI presents threats to validity, Section VII talks about related work, and Section VIII concludes the paper and presents next steps.

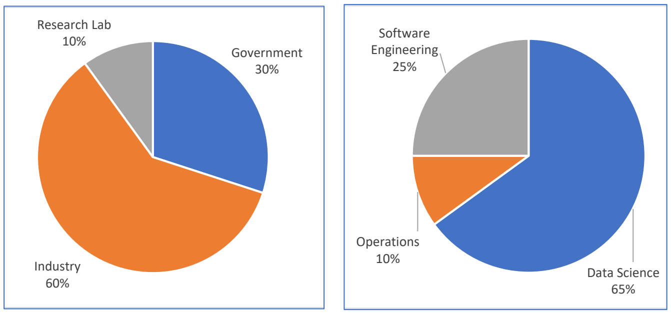
II. Study Design

We conducted 20 practitioner interviews to gather mismatch examples, and validated the interview results via a practitioner survey, as described in the following subsections.

A. Interviews

The goal of each interview was to gather examples of mismatch from practitioners in the roles of data scientist, software engineer, or operations for ML-enabled systems. Prior to each interview, each interviewee was sent a slide set describing the study. During the one-hour interview we followed a script to enable us to elicit examples of mismatch, consequences, and information that they believed that if shared would have avoided the mismatch. The presentation and interview guide are both available in the replication package for this study.[1]

We conducted the interviews between November 2019 and July 2020. All the people interviewed were contacts from existing or previous collaborations or work engagements. We only interviewed people that confirmed to have experience developing or deploying operational ML-enabled systems. We did not interview people that only had academic model development experience or developed models that ran standalone to produce reports. Demographics for interviewees are presented in Figure 1.



(a) Affiliation

(b) Primary Role

Fig. 1: Interviewee Demographics

Each interview was transcribed using a commercial transcription service and processed as follows. Steps were derived from well-established empirical software engineering guidelines [33] [37].

1) Transcript segmentation: Each transcript was imported into an Excel spreadsheet by one researcher and divided into segments based on breaks in the interview that indicated when a new mismatch example was being discussed, *e.g.,* ”What is another example of a mismatch that you experienced?” Pre-defined headers were added to the spreadsheet to facilitate the next step of mismatch identification.
2) Mismatch identification: Each segment was evaluated independently by two researchers against the inclusion and exclusion criteria shown in Table I to determine if the segment exemplified a mismatch. A segment was identified as a mismatch example if it met all of the inclusion criteria and none of the exclusion criteria. For those identified as a mismatch, the researcher added a specific quote from the segment representative of the mismatch discussed, a short description, and the information that was not communicated that caused the mismatch.
3) Mismatch validation: Results from the previous step were merged into a single spreadsheet. A third researcher reviewed each segment for which there was a disagreement and discussed with researchers until there was a consensus. For each agreed-upon mismatch, the third researcher produced a consolidated quote, short description, and information that should have been shared.
4) Mismatch coding: After creating a spreadsheet with only the validated, identified mismatches for all interviews, two researchers performed mismatch-by-mismatch content analysis using open coding [17] to categorize the types of mismatch identified in the interviews. The question that each researcher answered to perform the coding was "What element of an ML system does the information that was not communicated refer to?" The initial set of codes was created from the examples of ML system elements provided to interviewees as part of the project introduction slides.[2] Codes were divided into major categories (*e.g.,* raw data, trained model, training data) and subcategories (*e.g.,* for trained model, subcategories included programming language, API, version, etc.). The list of categories and sub-categories was expanded as researchers identified new codes. Given that mismatches could refer to more than one code, a researcher could assign up to three category/sub-category pairs per mismatch. A third researcher consolidated codes after each round of open coding. Three rounds were conducted until agreement was reached.

The resulting spreadsheet with anonymous results is included in the replication package. Note that interview transcripts are not included in the replication package per our protocol for human subject research (HSR) approved by our Institutional Review Board (IRB).

[1] Replication package available at github.com/GALewis/ML-Mismatch/

[2] Slide 4 in ML-Mismatch-Project-Introduction.pdf in replication package.

TABLE I: Inclusion and Exclusion Criteria

	Inclusion Criteria
I1	The segment describes a situation in which a system stakeholder made an assumption about a system element that was incorrect (*e.g.,* software engineer assumed that the model was ready to process operational data as-is).
I2	The situation described in the segment would not have occurred if information would have been shared between stakeholders (*e.g.,* data science team should have included the data pre-processing code along with the model).
	Exclusion Criteria
E1	The segment refers to problems that are internal to the data science process followed (*e.g.,* model parameters selected by the data scientist were not correct).
E2	The segment refers to problems that are internal to the software engineering process (*e.g.,* different engineers were using different versions of Python).
E3	The segment refers to problems that are internal to the operations process (*e.g.,* the tool used for runtime monitoring did not have a good way to alert users of problems).
E4	The situation described in the segment cannot be solved by sharing information between stakeholders (e.g., the data science team did not have enough data to train the model properly).
E5	The segment refers to a statement that is not related to a mismatch example (*e.g.,* introductory statements, small talk, this is how we did version control in my previous job).

B. Validation Survey

We conducted a survey to validate the resulting ML mismatch categories with both the interviewees as well as a larger population. In addition to demographics information, the survey asked each participant to rate the importance of sharing information related to each of the identified categories and subcategories for preventing mismatch. The participants were also given the opportunity to add any information that they felt was important but missing from the survey. The survey questions are also included in the replication package.

To ensure that the survey reached participants who met the criteria of having experience developing or deploying operational ML-enabled systems, we sent the survey to the interviewees and also asked them to share it with people in their organization according to the criteria. We used Qualtrics (https://qualtrics.com) as the survey administration tool.

III. Study Results

A. Interview Results

A total of 140 mismatches were identified in the interviews, which resulted in 232 instances of information that was not communicated that led to mismatch. The resulting mismatch categories based on open coding and their occurrence are presented in Figure 2. Each category is divided into subcategories, which are shown in Figure 3, along with their occurrence in each category.

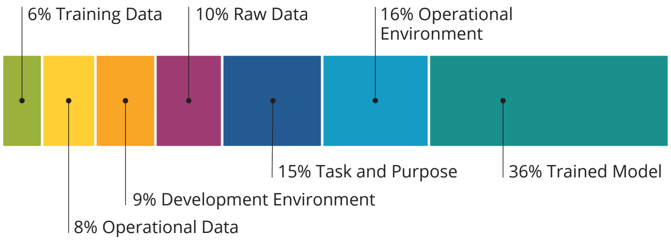


Fig. 2: Mismatch Categories

Most identified mismatches refer to incorrect assumptions about the *Trained Model* (36%), which is the model trained by data scientists that is passed to software engineers for integration into a larger system. The next category is *Operational Environment* (16%), which refers to the computing environment in which the trained model executes (i.e., model serving environment). Categories that follow are *Task and Purpose* (15%) which are the ”requirements” for the model, and *Raw Data* (10%) which is the operational or acquired data from which training data is derived. Finally, in a smaller proportion are the *Development Environment* (9%) used by software engineers for model integration and testing, the *Operational Data* (8%) which refers to the data processed by the model during operations, and the *Training Data* (6%) used to train the model.

Trained Model (TM). Within this category, most mismatches were related to lack of test cases and test data that could be used for integration testing (17%); and lack of model specifications and APIs that provide greater insight into inputs, outputs, and internals (if applicable) (17%). One software engineer interviewee stated *”I had many attempts but was never able to get from the [data scientists] a description of what components exist, what are their specifications, what would be some reasonable test we could run against them so we could reproduce all their results.”* Other subcategories under trained model included unawareness of decisions, assumptions, limitations, and constraints that affect model integration and deployment (14%); information necessary to interpret model output, results, or inferences (14%); programming language, ML framework, tools, and libraries used in the development and training of the model (12%); evaluation metrics and results of trained model evaluation such as false positive rate, false negative rate, and accuracy (11%); version information (8%); system configuration requirements for trained model to execute, such as number of CPUs and GPUs, libraries, tools, and dependencies (5%); and data buffering or time window requirements that would indicate that data has to be delivered in ”chunks” instead of streamed (2%).

Operational Environment (OE). Within this category, most mismatches were associated with lack of runtime metrics, logs (including deployed model version), data, user feedback, and other data collected in the operational environment to help with troubleshooting, debugging, or retraining (54%). One data scientist interviewee stated *”A typical thing that might happen is that in the production environment, something would happen. We would have a bad prediction, some sort of anomalous event. And we were asked to investigate that. Well, unless we have the same input data in our development environment, we can’t reproduce that event.”* Other subcategories were unawareness of computing resources available in the operational environment, such as CPUs, GPUs, memory, and storage (32%); and required model inference time (*i.e.,* time for the model to produce a result) (14%).

Task and Purpose (TP). Within this category, most mismatches were related to lack of knowledge of business goals or objectives that the model was going to help satisfy (29%).

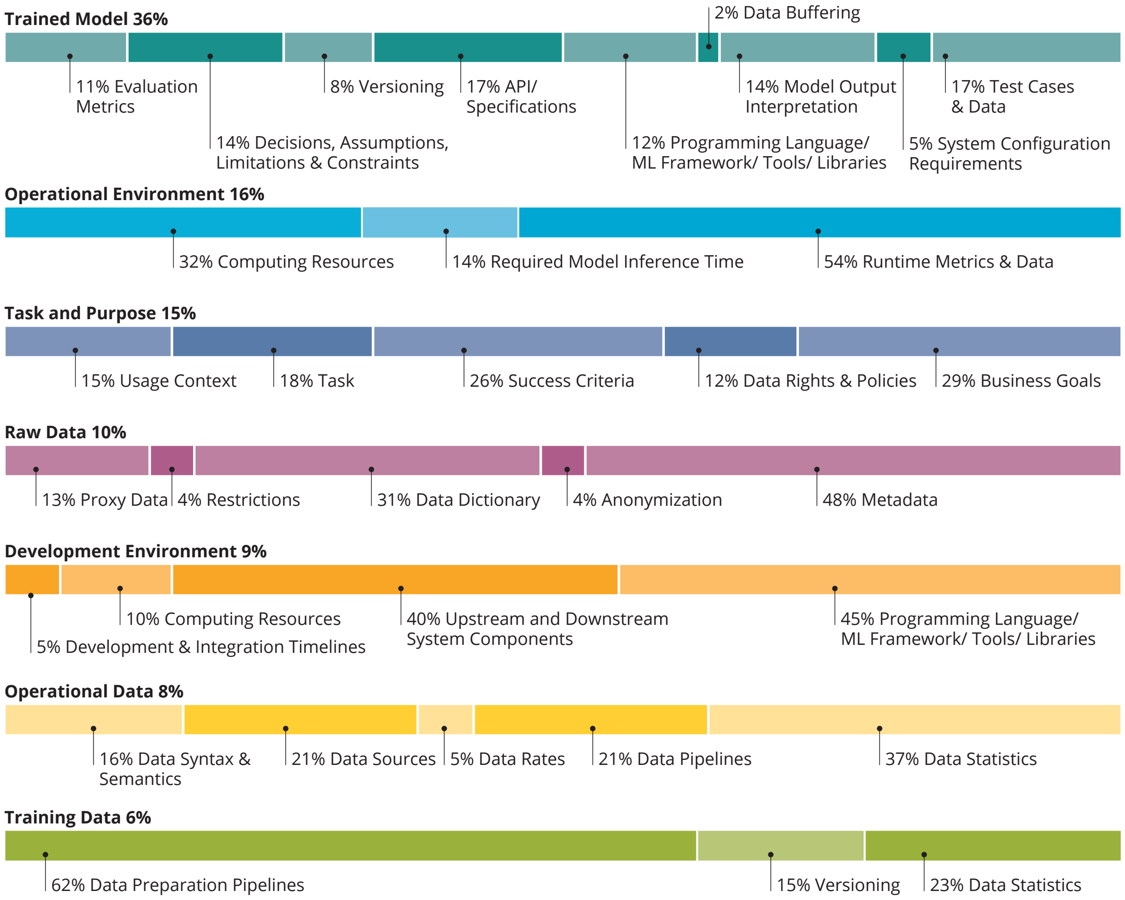


Fig. 3: Mismatch Subcategories

One data scientist interviewee stated *”It feels like the most broken part of the process because the task that comes to a data scientist frequently is – hey, we have a lot of data. Go do some data science to it – like go ... And then, that leaves a lot of the problem specification task in the hands of the data scientist.”* Other subcategories under task and purpose included unawareness of success criteria, client expectations, validation scenarios, or acceptance criteria for determining that the model is performing correctly (26%); task that model is expected to perform (18%); how the results of the model are going to be used by end users or in the context of a larger system (15%); and known data rights, legal, privacy, and other policies that need to be met by model and data (12%).

Raw Data (RD). Within this category, most mismatches were associated with lack of metadata about raw data, such as how it was collected, when it was collected, distribution, geographic location, and time frames (48%); and lack of description of data elements, such as field names, description, values, and meaning of missing or null values (31%). One data scientist interviewee stated *”Whenever they had data documentation available, that was amazing because you can immediately reference everything, bring it together, know what’s missing, know how it all relates. In the absence of that, then it gets incredibly difficult because you never know exactly what you’re seeing, like is this normal? Is it not normal? Can I remove outliers? What am I lacking? What do some of these categorical variables actually mean?”* Other subcategories were unawareness of (if applicable) the process used to generate or acquire proxy data due to sensitivities, legal, or policy reasons, including a mapping to operational data (13%); indication regarding data sensitivities that would prohibit for example upload to public cloud environments (4%); and the process used to anonymize data due to personally identifiable information constraints, including mapping to operational data (4%).

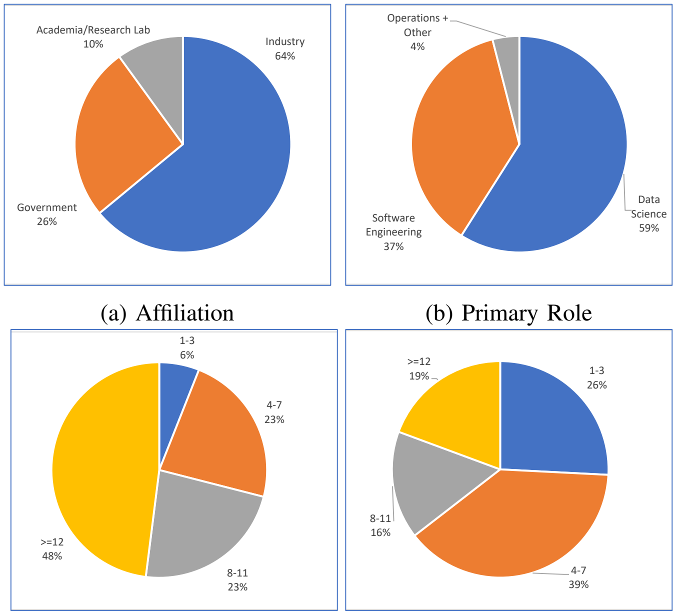
Development Environment (DE). Within this category, most mismatches were related to lack of knowledge of programming languages, ML frameworks, tools, and libraries used in the development environment (45%). One software engineer interviewee stated *”The weird failures that you see porting models from R prototypes to other languages is interesting … almost like re-optimizing the whole model for a new language … I was able to diagnose that the way floating point numbers are handled in R and Python does* *not translate directly.*" Other subcategories under development environment include unawareness of specifications or APIs for how data comes in from upstream components and is fed to downstream components (40%); computing resources available in the development environment, such as CPU, GPU, memory, and storage (10%); and development and integration timelines for integration of trained models into the larger system (5%).

Operational Data (OD). Within this category, most mismatches were associated with lack of operational data statistics, such as distribution and other metrics, that could be used by data scientists to validate appropriateness of training data (37%); and details on the implementation of data pipelines for the deployed model (21%). One operations interviewee stated *"There's the data inputs being restructured appropriately on the prototypes with this big complicated data pipeline leading up to them ... and we take it to deployment and you don't have the data coming through that same route anymore. You want to have it being straight from the sensor data. If they reconstruct that pipeline onboard ... there's so many opportunities there for mismatches."* Other subcategories were unawareness of sources for operational data for the operational model (21%); syntax and semantics of the data that constitutes the input for the operational model (16%); and rates at which operational data feeds into the operational model (5%).

Training Data (TD). Within this category, most mismatches were related to lack of details of data preparation pipelines to derive training data from raw data (62%). One software engineer interviewee stated *"A group developed the architecture for a whole ML pipeline ... but as a consequence of that, I think they sort of went a few steps further than they should have, creating lock-in, and kind of took over the feature engineering phase as well ... The mismatch was really at the design phase of the architecture of the machine learning pipeline where it really precluded us from doing more extensive research into alternative model architectures."* Other subcategories under training data include unawareness of training data statistics, such as distribution and other metrics (23%); and version information for training data (15%).

B. Validation Survey Results

A total of 31 survey responses were collected, which are included in the study replication package. Survey demographics are shown in Figure 4. We recognize the small number of respondents in the *Operations* role as a limitation, which is why our analyses will focus mostly on the *Data Science* and *Software Engineering* roles. However, we also highlight that because we were very specifically targeting practitioners, we asked our original interviewees to help us identify people in all three roles, and in most cases they could not identify an *Operations* person on their team. While simply a conjecture, the fact that *Operations* staff are not considered a key stakeholder in the end-to-end development and evolution of ML-enabled systems indicates the general lack of understanding of the key role of operations, and especially runtime monitoring, in these types of systems. For reporting purposes we combined *Operations* with the *Other* category, which were respondents who are currently in management-related roles.



(a) Affiliation

(b) Primary Role

(c) Total Years of Experience

(d) ML Years of Experience

Fig. 4: Survey Respondent Demographics

Results for all responses are shown in Figure 5. As shown in this figure, the importance of sharing information related to each subcategory to avoid mismatch was mostly rated between *Important* and *Very Important* for all, which demonstrates the validity of the identified causes for mismatch.

IV. Analysis and Discussion

In this section we analyze interview and survey results, and discuss implications for software engineering practices and tools when developing ML-enabled systems. While there are many observations that we could make about the data, we limit our analysis to those that inform software engineering best practices and tools. Numbers reported are extracted from Figures 3 and 5 and Table II.

Most mismatches identified during the interviews are related to incorrect assumptions about the *Trained Model* (36%). This is not surprising because the model constitutes the main "hand off" from a data science team to a software engineering team. Within this category, most mismatches were related to *Test Cases and Data* and *API/Specifications*, which are two pieces of information that are key for model integration into a larger system. However, survey data shows that what is most important to share about the trained model varies for each role. For the Data Scientist it is *Evaluation Metrics* because they (1) set expectations for model performance and (2) establish a baseline for runtime monitoring of model performance over time. What is most important for the Software Engineer is a tie between *Test Cases and Data*, *Decisions/Assumptions/Limitations/Constraints*, and *Model Output Interpretation*. For a software engineer these two last pieces

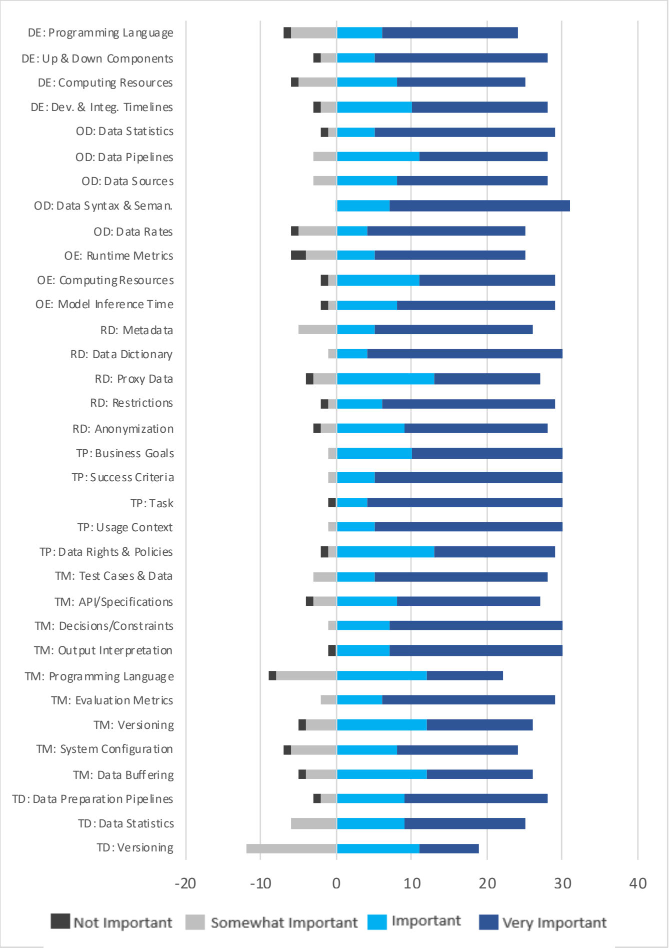


Fig. 5: Survey Responses

of information provide insights into (1) any additional components that needs to be developed to address incompatibilities and (2) how to properly pass model results to downstream components. For *Operations + Others* it is interesting to observe that *System Configuration* is one of the most important categories, but it is not for the data scientist nor the software engineer. This is an expected result as in the end this is the role that is responsible for serving that model and meeting any established service-level agreements (SLAs). It also hints at the fact that the operational environment might not be considered a constraint for model development, which leads to mismatches identified in the interviews in which complex models are created that cannot be served in the operational environment because there are not enough resources to do so.

Specifically related to *Test Cases & Data* there were 14 related mismatches. In some cases, software engineer interviewees reported receiving from data scientists the test data and results that they used for model testing and evaluation as information to use for model integration testing. However, they also reported that test data used for model development was often also the cause for mismatch because it is not enough for generating appropriate test cases. Such mismatch occurs because test data used for model development (1) does not take into account the often uncertain nature of ML models, *i.e.,* output should be expressed as acceptable boundaries or an expected order of results instead of exact values, and (2) does not include error cases such as input errors and out-of-distribution (OOD) data. While the concept of test cases is common for software engineering, it is not common for data scientists, which shows the value of having a shared understanding between different perspectives and roles.

TABLE II: Mismatch Categories Rated Very Important or Important (VI+I) per Role

	Data Science		Software Engineering		Operations + Other	
	VI+I	**%**	**VI+I**	**%**	**VI+I**	**%**
DE: Programming Language	11	69	8	80	5	100
DE: Up & Down Components	14	88	10	100	4	80
DE: Computing Resources	11	69	9	90	5	100
DE: Dev. & Integ. Timelines	14	88	9	90	5	100
OD: Data Statistics	15	94	10	100	4	80
OD: Data Pipelines	15	94	8	80	5	100
OD: Data Sources	14	88	9	90	5	100
OD: Data Syntax & Seman.	16	100	10	100	5	100
OD: Data Rates	11	69	10	100	4	80
OE: Runtime Metrics	13	81	9	90	3	60
OE: Computing Resources	14	88	10	100	5	100
OE: Model Inference Time	14	88	10	100	5	100
RD: Metadata	14	88	9	90	3	60
RD: Data Dictionary	16	100	10	100	4	80
RD: Proxy Data	14	88	9	90	4	80
RD: Restrictions	15	94	10	100	4	80
RD: Anonymization	14	88	9	90	5	100
TP: Business Goals	16	100	9	90	5	100
TP: Success Criteria	15	94	10	100	5	100
TP: Task	15	94	10	100	5	100
TP: Usage Context	15	94	10	100	5	100
TP: Data Rights & Policies	15	94	10	100	4	80
TM: Test Cases & Data	14	88	10	100	4	80
TM: API/Specifications	14	88	9	90	4	80
TM: Decisions/Constraints	15	94	10	100	5	100
TM: Output Interpretation	15	94	10	100	5	100
TM: Programming Language	11	69	7	70	4	80
TM: Evaluation Metrics	16	100	8	80	5	100
TM: Versioning	14	88	8	80	4	80
TM: System Configuration	12	75	7	70	5	100
TM: Data Buffering	14	88	8	80	4	80
TD: Data Preparation Pipelines	15	94	9	90	4	80
TD: Data Statistics	13	81	9	90	3	60
TD: Versioning	12	75	5	50	2	40

Data Syntax & Semantics for *Operational Data* rated most important in surveys. In addition, data scientists, software engineers, and operations people we surveyed unanimously agreed on the importance of sharing information about data syntax and semantics. However, there were only three mismatch examples related to this subcategory in our interview data. This could be because for many systems the *Raw Data* comes from the *Operational Data* and therefore is already well-known or documented, in which case no mismatch was observed. However, the surveys indicate that not having this type of information could lead to mismatch, which is why it is important to have. Further collection of mismatch examples would be needed to better understand this type of mismatch in practice.

With respect to the *Operational Environment*, the *Runtime Metrics* category received the most *Not Important* responses in the survey, yet in the interviews runtime metrics had the largest number of mismatch examples within this category (20). Runtime metrics in ML-enabled systems are critical for maintainability and continuous improvement of the model, especially to avoid model drift and to help decide when to retrain and redeploy a model. We attribute this disconnect to the fact that the survey had the lowest number of respondents from the operations community. ML-enabled system development still does not have agreed-upon, end-to-end development practices and most of the attention currently is in model development and not informed evolution, which can also explain the variance. To quote one of our survey respondents *"Mismatch in understanding what it means to run successful under real-world conditions is the #1 mismatch ... Some of the biggest mismatches in operational environment have to do with how a production system handles failure and overload."* We envision runtime metrics comprising *algorithm metrics* related to data drift (*e.g.,* differences in data distribution often referred to as training-serving skew, non-expected inputs) and *model performance metrics* (*e.g.,* accuracy, false positive and false negative rates, user feedback). This would require agreement between trained model evaluation metrics and operational environment runtime to ensure feasibility and completeness. Agreement on what information to log is also important (*e.g.,* should logging be done for all input/output pairs or only for anomalies).

Task and Purpose was a category in which most mismatches were rated as *Very Important* across all roles. We gathered 34 example of mismatch caused by not having a shared understanding of what basically constitutes the requirements for the model. As stated by one of the mismatch examples collected in our survey: *"It is key to understand the problem being solved ... It is easy to get trapped tuning a model that doesn't actually solve the problem."* We envision model requirements to include business requirements comprising the subcategories listed under *Task and Purpose*, in addition to technical requirements such as the subcategories listed under *Development Environment* and *Operational Environment* in Figure 3.

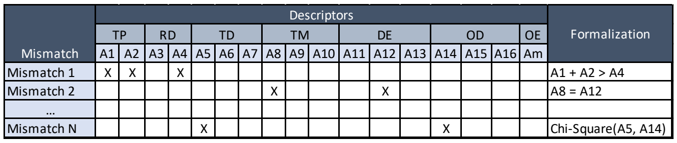
In general, the mismatch subcategory that was considered less important by both interviewees and all survey respondent roles was *Training Data: Version*. This was a surprising result because of the tight relationship between model performance and training data. For model troubleshooting and retraining, knowing the exact data that the deployed model was trained with would seem important. The other subcategories in *Training Data* were also generally rated low in importance even though the interviews showed several negative consequences of not having this information, such as the inability to perform data drift detection and other runtime monitoring when *Distribution* and *Data Statistics* are not known. In addition, when details of *Data Preparation Pipelines* are not known there is lack of clarity of how much data manipulation and validation happens in the model and how much happens outside the model. Data preparation pipelines was rated of higher importance by software engineers, likely because they have to deal with the consequences of not having this information. Data pipeline modularity would not only enable reuse of data processing code, but would also address mismatches related to lack of knowledge of data pipeline details.

Metadata in the *Raw Data* category related to 11 mismatch examples and was considered *Important/Very Important* by 81% of survey respondents. Interestingly, *Data Dictionary* was related to 7 mismatch examples and was considered *Important/Very Important* by 94% of survey respondents. Using data dictionaries is a practice that is common in the database community that not surprisingly would be well received in the data science community to better understand raw data. On the other hand, having access to metadata provides insights into how representative data is of operational data, which is equally or even more important which our survey results do not reflect as strongly.

Programming Language/ML Framework/Tools/Libraries was a subcategory that contained a large number of interview mismatch examples in both *Development Environment* (9) and *Trained Model* (10), which can be seen as counterparts. Most of these mismatches had to do with having to port models because of language differences, which when combined with lack of model API/Specifications is a very error-prone activity. However, these two categories were not as important among survey respondents. One explanation is that perhaps these respondents did not have to deal with model porting, which seemed to be common in interviewees mostly from Government in which models are developed by contractors outside of their organizations. Another explanation is a growing trend towards deploying models as microservices, which would in fact hide some of these differences [28].

V. Towards Automated Mismatch Detection

As stated earlier, the end goal of our study is to use the ML mismatch information extracted from the interviews and survey responses to develop empirically-grounded machine-readable descriptors for different elements of ML-enabled systems. To this effect, in parallel we conducted a multi-vocal literature review [10] to identify software engineering best practices and challenges in the development and deployment of ML-enabled systems from both the academic and the practitioner perspective. Attributes for documenting elements of ML-enabled systems were extracted or inferred from the primary studies (publication pending). In Phase 2 of our research, we will perform a mapping between different system element attributes and mismatches in which for each mismatch we identify the set of attributes that could be used to detect that mismatch, and formalize the mismatch as a predicate over those attributes, as shown in the Formalization column in Figure 6. As an example, the figure shows that Mismatch 1 occurs when the value of Attribute 1 plus the value of Attribute 2 is greater than the value of Attribute 5. We will then perform gap analysis to identify mismatches that do not map to any attributes, and attributes that do not map to any mismatch. We then complement the mapping based on information from the interviews and survey plus our domain knowledge, by adding attributes and potentially adding new mismatches that could be detected based on the available attributes.



Mismatch	Descriptors																	Formalization
	TP		RD		TD			TM			DE			OD			OE	
	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	Am	
Mismatch 1	X	X		X														A1 + A2 > A4
Mismatch 2								X				X						A8 = A12
...																		
Mismatch N					X									X				Chi-Square(A5, A14)

Fig. 6: Mapping between Mismatches and ML-Enabled System Element Attributes

The resulting attributes will be codified into JSON Schema documents (https://json-schema.org/) that can be used by automated mismatch detection tools. These tools can range from a simple web-based client that reads in all descriptors and presents then to a user evaluating documentation, to a more elaborate tool or system component such as the one presented in Figure 7, which uses the *Distribution* attribute from the *Training Data* descriptor for runtime data drift detection, by performing a chi-square test between the distributions of training data and the operational data.

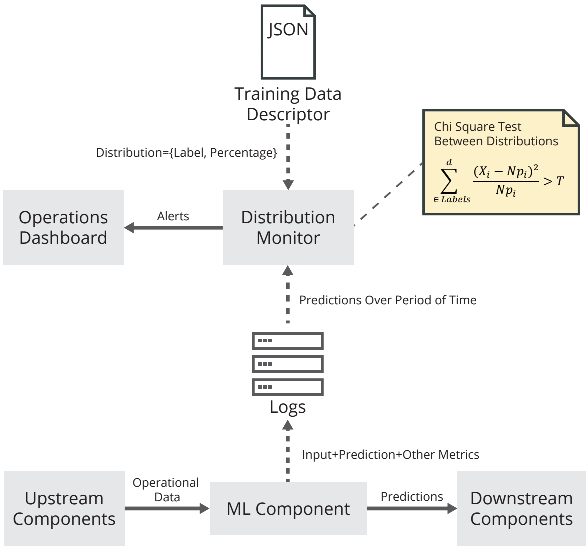

JSON
Training Data Descriptor
Distribution={Label, Percentage}
Chi Square Test Between Distributions
$\sum_{\in Labels}^{d} \frac{(X_i - Np_i)^2}{Np_i} > T$
Operations Dashboard
Alerts
Distribution Monitor
Predictions Over Period of Time
Logs
Input+Prediction+Other Metrics
Upstream Components
Operational Data
ML Component
Predictions
Downstream Components

Fig. 7: Data Drift Detection Tool

VI. Threats to Validity

External validity. We acknowledge that the selected interviewees might not be representative of all practitioners involved in the development and deployment of ML-enabled systems, and therefore the identified mismatches may not cover all potential mismatch types. To mitigate this threat we specifically invited practitioners of different affiliation and roles, as shown in Figure 1. For the validation survey we also obtained a distribution of responses of different affiliation, roles, and years of experience as shown in Figure 4. We are aware that the majority of interviewees and survey respondents are data scientists, and this might bias the results towards the data science perspective. To address this issue, in the interviews we always requested examples of mismatch caused by information not provided or received, which gave us a broader set of mismatch examples. However, we fully recognize this threat and propose to repeat the study at a larger scale and with a more balanced population. Finally, in this study we do not claim to have a complete representation of all ML mismatches; the mismatches we identified are meant to be a representative set to provide insights into the end-to-end ML-enabled system development challenges.

Internal validity. The study was conducted following well-established empirical software engineering guidelines [33] [37]. The artifacts that guided the study, as well as anonymized results, are available in the replication package. The qualitative analysis for identifying mismatches from interview segments and coding mismatches is based on manual inspection and categorization of mismatches that can potentially lead to subjective results. We mitigated this potential threat by carefully following content and thematic analysis methodologies involving two researchers, with a third researcher performing validation and solving disagreements. In addition, the mismatch categories and subcategories were validated via survey by 31 practitioners whom rated all mismatches mostly between *Important* and *Very Important*, as shown in Section III-B.

Conclusion validity. A potential threat to conclusion validity may lie in our defined mismatch identification and coding, as other researchers may have identified different mismatch segments and codes and produced completely different results. We mitigated this potential threat by carefully documenting the process for mismatch identification and coding (Section II-A) and having three researchers follow the process in collaboration. Our replication package includes coded segments.

Construct validity. To make sure that interview and survey results are consistent and comparable, it is important to ensure that they are all conducted in a consistent and repeatable manner. We mitigated this threat by ensuring that all interviews were conducted in the same manner, following the interview protocol available in the replication package. The execution platform for all surveys was Qualtrics. Results were directly exported from Qualtrics to a spreadsheet for analysis to guarantee that all graphs and summary tables are generated consistently from the same data.

VII. Related Work

There are several published practitioner studies that focus on software engineering challenges and best practices for the development and deployment of ML-enabled systems. Amershi et al [1] conducted a study with developers at Microsoft to understand how teams build software applications with customer-focused AI features. The results were a set of challenges and best practices, as well as the beginnings of a model of ML process maturity, inspired by the finding that what teams perceive as challenges depends on the level of AI experience on the team. Lwakatare et al [19] developed a taxonomy of software engineering challenges based on studying the development of ML systems in six companies. The challenges focused mostly on data science and were organized around a set of maturity stages related to the evolution of use of ML in commercial software-intensive systems. Hill et al [12] interviewed data science researchers in the context of how do they develop intelligent systems powered by machine learning components in practice. The findings of this earlier study included the need for software engineering for machine learning. More recently, Serban et al [32] conducted an academic and gray literature review of best practices for development of ML applications, and validated adoption in real projects via a practitioner survey.

In this study we conducted practitioner interviews with the specific purpose of identifying information that should have been shared between stakeholders in order to avoid mismatch when developing and deploying ML-enabled systems. This information was then validated via a practitioner survey. As stated in Section V, the end goal of our study is to develop machine-readable descriptors for elements of ML-enabled systems, which would constitute a software engineering best practice for documenting elements of ML-enabled systems. The resulting descriptors constructed from this information address challenges highlighted in these and other studies related to requirements [9] [19] [18] [20] [22] [30]; model integration and testing [1] [8] [14] [22]; runtime metrics for model debugging and troubleshooting [4] [6] [9] [16] [18] [20] [23] [25] [31] [30] [34]; differences between training, development, and operational environments [18] [30]; and lack of metadata, documentation, and specifications for both models and data [1] [2] [4] [6] [8] [12] [18] [19] [20] [23] [24] [25] [29] [30] [31] [34] [35] [36].

From the descriptor perspective, while there is existing, recent work in creating descriptors for data sets [5] [11] [13], models [21], and online AI services [3] [27], there are two main limitations: (1) they do not address the software engineering and operations perspectives, (2) they are not machine-readable, and (3) they are targeted at selection or evaluation of existing data set and models and not at end-to-end system development. Our descriptors will address these three limitations.

VIII. Conclusions and Next Steps

Empirically-validated practices and tools to support software engineering of ML-enabled system development are still in their infancy. In this paper, we presented the results of our interview and survey study to understand the types of mismatch that occur in the development and deployment of ML-enabled systems due to incorrect assumptions made by different stakeholders. While the excitement around developing ML models is increasing, understanding how to deploy, operate, and sustain these models as part of sound end-to-end ML-enabled system development remains a challenge. The 7 categories along with their 34 subcategories of ML mismatches that we identified contributes to codifying the nature of the challenges. The Phase 1 results of our study demonstrate that improved communication and automation of ML mismatch awareness and detection can help improve software engineering of ML-enabled systems.

The next steps of our study include implementing the machine-readable descriptors as described in Section V, validating descriptors in industry, and implementing their detection using sample tools such as the web-based descriptor viewer and the runtime data drift detector described in Section V. Our vision is to make the descriptors publicly available and create a community around tool development and descriptor extensions, with the end goal of improving the state of the engineering practices for development, operation, and evolution of ML-enabled systems.

Acknowledgments

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center (DM20-0959).

References

[1] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann. Software Engineering for Machine Learning: A Case Study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 291–300. IEEE, 2019.

[2] S. Andrist, D. Bohus, E. Kamar, and E. Horvitz. What Went Wrong and Why? Diagnosing Situated Interaction Failures in the Wild. In *International Conference on Social Robotics*, pages 293–303. Springer, 2017.

[3] M. Arnold, R. K. E. Bellamy, M. Hind, S. Houde, S. Mehta, A. Mojsilović, R. Nair, K. N. Ramamurthy, A. Olteanu, D. Piorkowski, D. Reimer, J. Richards, J. Tsay, and K. R. Varshney. FactSheets: Increasing Trust in AI Services through Supplier's Declarations of Conformity. *IBM Journal of Research and Development*, 63(4/5):6:1–6:13, 2019.

[4] A. Arpteg, B. Brinne, L. Crnkovic-Friis, and J. Bosch. Software Engineering Challenges of Deep Learning. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 50–59. IEEE, 2018.

[5] E. M. Bender and B. Friedman. Data Statements for Natural Language Processing: Toward Mitigating System Bias and Enabling Better Science. *Transactions of the Association for Computational Linguistics*, 6:587–604, 2018.

[6] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley. The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction. In *2017 IEEE International Conference on Big Data*, pages 1123–1132. IEEE, 2017.

[7] T. Böhm. How to Bridge Machine Learning and Software Engineering: A Practical Workflow to Turn Data Science into Software. https://towardsdatascience.com/how-to-bridge-machine-learning-and-software-engineering-6f51b38f4b49, Jun 2019.

[8] D. A. Clifton, J. Gibbons, J. Davies, and L. Tarassenko. Machine Learning and Software Engineering in Health Informatics. In *2012 First International Workshop on Realizing AI Synergies in Software Engineering (RAISE)*, pages 37–41. IEEE, 2012.

[9] I. Flaounas. Beyond the Technical Challenges for Deploying Machine Learning Solutions in a Software Company. In *Human in the Loop Machine Learning Workshop at the International Conference on Machine Learning*, 2017.

[10] V. Garousi, M. Felderer, and M. V. Mäntylä. Guidelines for Including Grey Literature and Conducting Multivocal Literature Reviews in Software Engineering. *Information and Software Technology*, 106:101–121, 2019.

[11] T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. Wallach, H. Daumé III, and K. Crawford. Datasheets for Datasets. In *Proceedings of the 5th Workshop on Fairness, Accountability, and Transparency in Machine Learning*, 2018.

[12] C. Hill, R. Bellamy, T. Erickson, and M. Burnett. Trials and Tribulations of Developers of Intelligent Systems: A Field Study. In *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 162–170. IEEE, 2016.

[13] S. Holland, A. Hosny, S. Newman, J. Joseph, and K. Chmielinski. The Dataset Nutrition Label: A Framework to Drive Higher Data Quality Standards. *arXiv preprint arXiv:1805.03677*, 2018.

[14] F. Khomh, B. Adams, J. Cheng, M. Fokaefs, and G. Antoniol. Software Engineering for Machine-Learning Applications: The Road Ahead. *IEEE Software*, 35(5):81–84, 2018.

[15] P. Kriens and T. Verbelen. Software Engineering Practices for Machine Learning. https://arxiv.org/abs/1906.10366, June 2019.

[16] T. Kulesza, M. Burnett, W.-K. Wong, and S. Stumpf. Principles of Explanatory Debugging to Personalize Interactive Machine Learning. In *Proceedings of the 20th International Conference on Intelligent User Interfaces*, pages 126–137, 2015.

[17] W. Lidwell, K. Holden, and J. Butler. *Universal Principles of Design, Revised and Updated: 125 Ways to Enhance Usability, Influence Perception, Increase Appeal, Make Better Design Decisions, and Teach through Design*. Rockport Pub, 2010.

[18] J. Lin and D. Ryaboy. Scaling Big Data Mining Infrastructure: the Twitter Experience. *ACM SIGKDD Explorations Newsletter*, 14(2):6–19, 2013.

[19] L. E. Lwakatare, A. Raj, J. Bosch, H. H. Olsson, and I. Crnkovic. A Taxonomy of Software Engineering Challenges for Machine Learning Systems: An Empirical Investigation. In *International Conference on Agile Software Development*, pages 227–243. Springer, Cham, 2019.

[20] T. Menzies, C. Bird, T. Zimmermann, W. Schulte, and E. Kocaganeli. The Inductive Software Engineering Manifesto: Principles for Industrial Data Mining. In *Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering*, pages 19–26, 2011.

[21] M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D. Raji, and T. Gebru. Model Cards for Model Reporting. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, pages 220–229, 2019.

[22] C. Murphy, G. E. Kaiser, and M. Arias. An Approach to Software Testing of Machine Learning Applications. In *The 19th International Conference on Software Engineering and Knowledge Engineering*, 2007.

[23] B. Nushi, E. Kamar, E. Horvitz, and D. Kossmann. On Human Intellect and Machine Failures: Troubleshooting Integrative Machine Learning Systems. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2016.

[24] N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich. Data Management Challenges in Production Machine Learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1723–1726, 2017.

[25] T. Raeder, O. Stitelman, B. Dalessandro, C. Perlich, and F. Provost. Design Principles of Massive, Robust Prediction Systems. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1357–1365, 2012.

[26] W. Reisz. Josh Wills on Building Resilient Data Engineering and Machine Learning Products at Slack. https://www.infoq.com/podcasts/slack-building-resilient-data-engineering/, Dec. 2019.

[27] J. Richards, D. Piorkowski, M. Hind, S. Houde, and A. Mojsilović. A Methodology for Creating AI FactSheets. *arXiv preprint arXiv:2006.13796*, 2020.

[28] D. Sato, A. Wider, and C. Windheuser. Continuous Delivery for Machine Learning: Automating the end-to-end lifecycle of Machine Learning applications. https://martinfowler.com/articles/cd4ml.html#ModelDeployment, Sept. 2019.

[29] S. Schelter, J.-H. Boese, J. Kirschnick, T. Klein, and S. Seufert. Automatically Tracking Metadata and Provenance of Machine Learning Experiments. In *Machine Learning Systems Workshop at NIPS*, 2017.

[30] J. Schleier-Smith. An Architecture for Agile Machine Learning in Real-Time Applications. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2059–2068, 2015.

[31] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison. Hidden Technical Debt in Machine Learning Systems. In *Advances in Neural Information Processing Systems*, pages 2503–2511, 2015.

[32] A. Serban, K. van der Blom, H. Hoos, and J. Visser. Adoption and effects of software engineering best practices in machine learning. In *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) 2020*, 2020.

[33] F. Shull, J. Singer, and D. I. Sjøberg. *Guide to Advanced Empirical Software Engineering*. Springer, 2007.

[34] V. Sridhar, S. Subramanian, D. Arteaga, S. Sundararaman, D. Roselli, and N. Talagala. Model Governance: Reducing the Anarchy of Production ML. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 351–358, 2018.

[35] S. Tata, A. Popescul, M. Najork, M. Colagrosso, J. Gibbons, A. Green, A. Mah, M. Smith, D. Garg, C. Meyer, et al. Quick Access: Building a Smart Experience for Google Drive. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1643–1651, 2017.

[36] T. van der Weide, D. Papadopoulos, O. Smirnov, M. Zielinski, and T. van Kasteren. Versioning for End-to-End Machine Learning Pipelines. In *Proceedings of the 1st Workshop on Data Management for End-to-End Machine Learning*, pages 1–9, 2017.

[37] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering*. Springer Science & Business Media, 2012.