Component Mismatches Are a Critical Bottleneck to Fielding AI-Enabled Systems in the Public Sector

Grace A. Lewis, Stephany Bellomo, April Galyardt
{glewis, sbellomo, akgalyardt}@sei.cmu.edu
Carnegie Mellon Software Engineering Institute
Pittsburgh, PA USA

Abstract

The use of machine learning or artificial intelligence (ML/AI) holds substantial potential toward improving many functions and needs of the public sector. In practice however, integrating ML/AI components into public sector applications is severely limited not only by the fragility of these components and their algorithms, but also because of mismatches between components of ML-enabled systems. For example, if an ML model is trained on data that is different from data in the operational environment, field performance of the ML component will be dramatically reduced. Separate from software engineering considerations, the expertise needed to field an ML/AI component within a system frequently comes from outside software engineering. As a result, assumptions and even descriptive language used by practitioners from these different disciplines can exacerbate other challenges to integrating ML/AI components into larger systems. We are investigating classes of mismatches in ML/AI systems integration, to identify the implicit assumptions made by practitioners in different fields (data scientists, software engineers, operations staff) and find ways to communicate the appropriate information explicitly. We will discuss a few categories of mismatch, and provide examples from each class. To enable ML/AI components to be fielded in a meaningful way, we will need to understand the mismatches that exist and develop practices to mitigate the impacts of these mismatches.

1 Introduction

The public sector owns and uses many very large data collections for a variety of purposes. Machine learning or artificial intelligence (ML/AI) components hold substantial potential toward improving many functions and needs of the public sector, based on the analysis of this data. In practice however, fielding these components into public sector applications is severely limited by the fragility of ML/AI components and their algorithms. For systems that touch the government and public sector, some considerations for ML/AI systems are more at the forefront than in the commercial sector, including privacy, security and ethics.

One of the challenges in deploying complex systems is integrating all components and resolving any component mismatches. For systems that incorporate ML/AI components, the sources and effects of these mismatches may be different from other software integration efforts. For example, if an ML model is trained on data that is different from data in the operational environment, field performance of the ML component will be dramatically reduced. Separate from software engineering considerations, the expertise needed to field an ML/AI component within a system frequently comes from outside software engineering. As a result, assumptions and even descriptive language used by practitioners from these different disciplines can exacerbate other challenges to integrating ML/AI components into larger systems.

We are investigating classes of mismatches in ML/AI systems integration, to identify the implicit assumptions made by practitioners in different fields (data scientists, software engineers, operations staff) and find ways to communicate the appropriate information explicitly. We will discuss a few categories of mismatch, and provide examples from each class. To enable ML/AI components to be fielded in a meaningful way, we will need to understand the mismatches that exist and develop practices to mitigate the impacts of these mismatches. This paper reports on the goals of our study and the expected results.

2 Mismatch in Machine-Learning Enabled Systems

Despite the growing interest in ML and AI across all industries — including DoD, government, and public sector — development of ML and AI capabilities is still mainly a research activity or a stand-alone project, with the exception of large companies such as Google and Microsoft (Ghelani 2019). Deploying ML models in operational systems remains a significant challenge (Amershi et al. 2019)(Ransbotham et al. 2017)(Sculley et al. 2015)(Talby 2018).

We define an ML-enabled system as a software system that relies on one or more ML software components to provide required capabilities, as shown in Figure 1. The ML component in this figure receives (processed) operational data from one software component and generates an insight that is consumed by another software component. A problem in these types of systems is that their development and operation involve three perspectives, with three different and often completely separate workflows and people.

1. The *Data Scientist* builds the model: The workflow of the data scientist, as shown in Figure 2, is to take an untrained

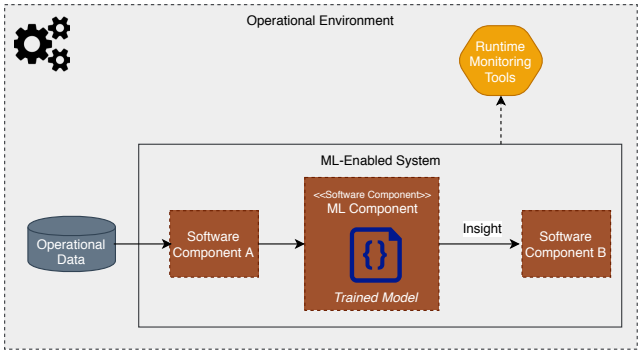

Figure 1: Elements of a Deployed ML-Enabled System and Operations Perspective

model and raw data, use feature engineering to create a set of training data that is then used to train the model, repeating these steps until a set of adequate models are produced, and then using a set of test data to test the different models and select the one that performs the best based on a set of defined evaluation metrics. Out of this workflow comes a trained model.

2. The *Software Engineer* integrates the trained model into a larger system: The workflow of the software engineer, as shown in Figure 3, is to take the trained model, integrate the model into the ML-enabled system, and test the system until it passes all tests. The ML-enabled system is then passed to operations staff for deployment.
3. *Operations Staff* deploy, operate, and monitor the system: As shown in Figure 1, in addition to the operation and monitoring of the ML-enabled system, operations staff are also responsible for operation and monitoring of operational data sources (*e.g.,* databases, data streams, data feeds).

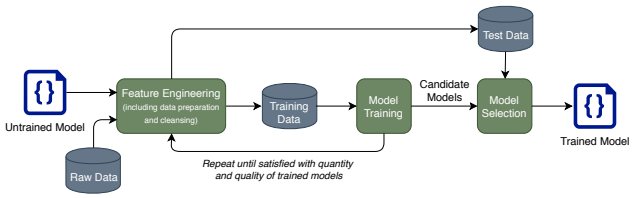

Figure 2: Data Scientist Perspective

Because these perspectives operate separately and often speak different languages, there are 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. In addition, we expect these perspectives to belong to three different organizations, especially in the public sector. Examples of mismatch and their consequences include

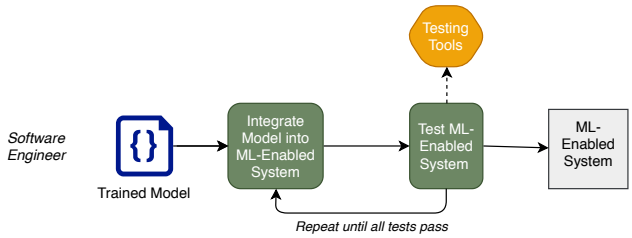

Figure 3: Software Engineer Perspective

- poor system performance because computing resources used during testing of the model are different from computing resources used during operations
- poor model accuracy because model training data is different from operational data
- development of large amounts of glue code because the trained model input/output is incompatible with operational data types
- system failure due to inadequate testing because developers were not able to replicate the testing that was done during model training
- monitoring tools are not set up to detect diminishing model accuracy, which is the "performance" metric defined for the trained model.

3 ML-Enabled System Element Descriptors

We are developing machine-readable *ML-Enabled System Element Descriptors* as a mechanism to enable mismatch detection and prevention in ML-enabled systems. The goal of the descriptors is to codify attributes of system elements and therefore make explicit all assumptions from all perspectives. 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 runtime for cases in which attributes lend themselves to automation. While there is existing, recent work in creating descriptors for data sets (Gebru et al. 2018), models (Mitchell et al. 2019), and online AI services (Hind et al. 2018), there are two main limitations in this work: (1) they do not address the software engineer and operations perspectives, and (2) they are not machine-readable. Our work addresses these two limitations, in addition to providing the following immediate benefits:

- Definitions of mismatch can serve as checklists as ML-enabled systems are developed
- Recommended descriptors provide stakeholders (*e.g.,* program offices) with examples of information to request and/or requirements to impose
- Means identified for validating ML-enabled system element attributes provide ideas for confirming information provided by third-parties
- Identification of attributes for which automated detection is feasible defines new software components for ML-enabled systems

4 Study Protocol

The technical approach for constructing and validating the ML-Enabled System Element Descriptors consists of three phases.

Phase 1 - Information Gathering: As shown in Figure 4, this phase involves two parallel tasks. In one task, we elicit examples of mismatches and their consequences from practitioners via interviews and/or workshops. In the second task, we identify attributes currently used to describe elements of ML-enabled systems by mining project descriptions from GitHub repositories that contain trained and untrained models (Kalliamvakou et al. 2016), a literature survey, and a gray literature review (Garousi, Felderer, and Mäntylä 2019). This multi-modal approach provides both the practitioner and the academic perspective on best practices to describe ML-enabled system elements.

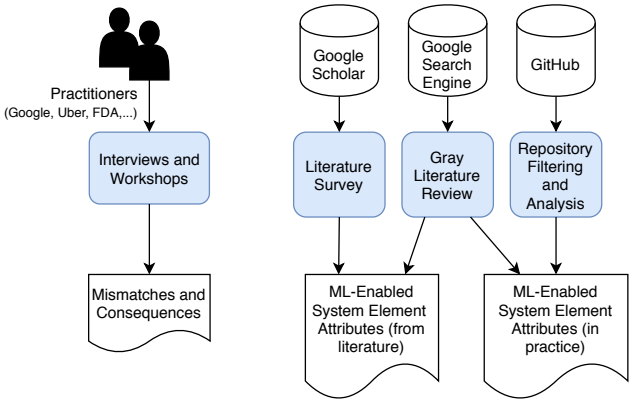

Figure 4: Information Gathering

Phase 2 - Analysis: The tasks in this phase are shown in Figure 5. Once mismatches and attributes of elements of ML-enabled systems have been elicited, there is a mapping stage in which an initial version of the spreadsheet shown in Figure 6 is produced. 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. The second step is to 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 our domain knowledge, by adding attributes and potentially adding new mismatches that could be detected based on the available attributes. Finally, there is a data source and feasibility analysis step where for each attribute we identify the data source (who provides the value), the feasibility of collecting those values (is it reasonable to expect someone to provide that value and/or is there a way of automating its collection), how can it be validated (if necessary to validate that the provided value is correct), and finally potential for automation (can the set of identified attributes be used in scripts or tools for detecting that mismatch). After the analysis stage we have an initial version of the spreadsheet, and an initial version of the descriptors derived from the spreadsheet.

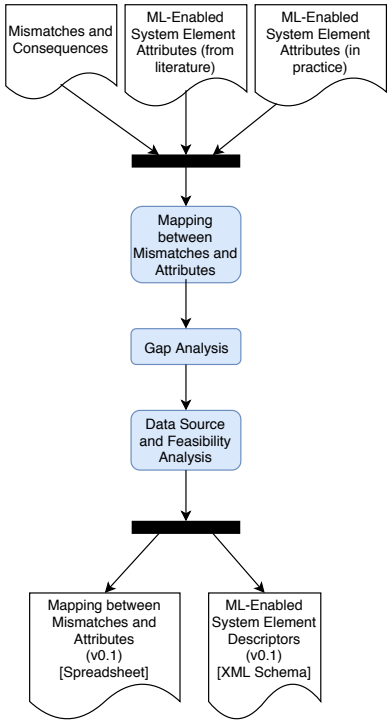

Figure 5: Analysis

Phase 3 – Evaluation: As shown in Figure 7, in this stage we re-engage with the interview or workshop participants from the Information Gathering stage to validate mapping, data sources, and feasibility. The evaluation target is a 90% agreement on the work developed in the Analysis stage. In addition, we develop a small-scale demonstration of automated mismatch detection. The target is to identify 2-3 mismatches in a project that can be detected via automation, and develop scripts that can detect the mismatch. In the end, the project outcomes are the validated mapping between mismatches and attributes, a set of descriptors created from that mapping, and instances of the descriptors.

5 Summary and Next Steps

Our vision for this work is that the community starts developing tools for automatically detecting mismatch, and organizations start including mismatch detection in their toolchains for development of ML-enabled systems. As a step toward this vision, we are working on the following artifacts:

- List of mismatches in ML-enabled systems and their consequences
- List of attributes for ML-enabled system elements
- Mapping of mismatches to attributes (spreadsheet)
- XML schema for each descriptor (one per system element) plus XML examples of descriptors

Mismatch	Descriptors																	Formalization
	Trained Model		Training Data		Untrained Model			System Components			Operational Environment			Operational Data			Descriptor M	
	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	Am	
Mismatch 1	X	X			X													A1 + A2 > A5
Mismatch 2								X				X						A8 = A12
…																		
Mismatch N				X										X				Chi-Square(A4, A14)

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

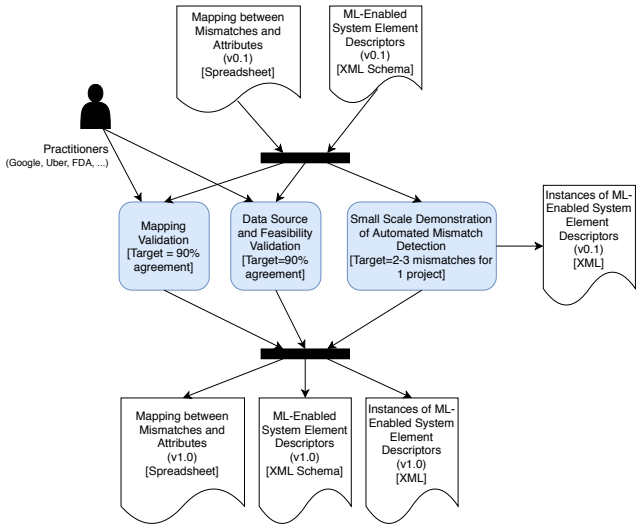

Figure 7: Evaluation

- Small-scale demonstration (scripts) of automated mismatch detection

We are using opportunities such as this workshop to (1) socialize the concept of mismatch and convey its importance for the deployment of ML-enabled systems into production, (2) elicit and confirm mismatches in ML-enabled systems and their consequences from people in the field, in particular from the public sector, (3) obtain early feedback on the study protocol and resulting artifacts.

Acknowledgements

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 (DM19-1007).

References

Amershi, S.; Begel, A.; Bird, C.; DeLine, R.; Gall, H.; Kamar, E.; Nagappan, N.; Nushi, B.; and Zimmermann, T. 2019. Software engineering for machine learning: a case study. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*, 291–300. IEEE Press.

Garousi, V.; Felderer, M.; and Mäntylä, M. V. 2019. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Information and Software Technology* 106:101–121.

Gebru, T.; Morgenstern, J.; Vecchione, B.; Vaughan, J. W.; Wallach, H.; Daumeé III, H.; and Crawford, K. 2018. Datasheets for datasets. *arXiv preprint arXiv:1803.09010*.

Ghelani, S. 2019. Ml models - prototype to production. Towards Data Science. https://towardsdatascience.com/ml-models-prototype-to-production-6bfe47973123.

Hind, M.; Mehta, S.; Mojsilovic, A.; Nair, R.; Ramamurthy, K. N.; Olteanu, A.; and Varshney, K. R. 2018. Increasing trust in ai services through supplier's declarations of conformity. *arXiv preprint arXiv:1808.07261*.

Kalliamvakou, E.; Gousios, G.; Blincoe, K.; Singer, L.; German, D. M.; and Damian, D. 2016. An in-depth study of the promises and perils of mining github. *Empirical Software Engineering* 21(5):2035–2071.

Mitchell, M.; Wu, S.; Zaldivar, A.; Barnes, P.; Vasserman, L.; Hutchinson, B.; Spitzer, E.; Raji, I. D.; and Gebru, T. 2019. Model cards for model reporting. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, 220–229. ACM.

Ransbotham, S.; Kiron, D.; Gerbert, P.; and Reeves, M. 2017. Reshaping business with artificial intelligence: Closing the gap between ambition and action. *MIT Sloan Management Review* 59(1).

Sculley, D.; Holt, G.; Golovin, D.; Davydov, E.; Phillips, T.; Ebner, D.; Chaudhary, V.; Young, M.; Crespo, J.-F.; and Dennison, D. 2015. Hidden technical debt in machine learning systems. In *Advances in neural information processing systems*, 2503–2511.

Talby, D. 2018. Lessons learned turning machine learning models into real products and services: Why model development does not equal software development. O'Reilly Media. https://www.oreilly.com/ideas/lessons-learned-turning-machine-learning-models-into-real-products-and-services.