Software Engineering for Machine Learning

Characterizing and Detecting Mismatch and Predicting Inference Degradation in ML Systems

Grace A. Lewis

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

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.

DM21-0059

Introduction

An area of work within the SEI is developing practices, methods and tools for reliable end-to-end development, deployment, and evolution of AI-enabled systems.

Our goal is to develop empirically validated practices to guide AI engineering and support software engineering for machine learning (SE4ML) systems.

This webinar reports on two focus areas:

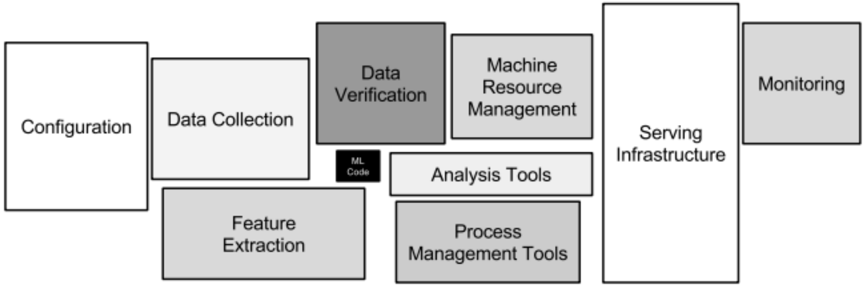
- Characterizing and Detecting Mismatch in ML-Enabled Systems
- Predicting Inference Degradation in Production ML Systems

Why Software Engineering for Machine Learning? 1

Machine learning components are parts of much larger systems

One challenge with ML components is that their performance depends on how similar operational data is to their training data (i.e., training-serving skew)

- Systems need to provide a way to know when model performance is degrading
- Systems need to provide enough information for retraining



"Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex." [Sculley 2015]

Why Software Engineering for Machine Learning? 2

ML-enabled systems need to be engineered such that

- System is instrumented for runtime monitoring of ML components and operational data
- Training-retraining cycle is shortened
- ML component integration is straightforward

Many existing SE practices apply directly but are simply not used in the data science field

Other SE practices will have no be adapted to extended to deal with ML components

January 2020 – December 2020

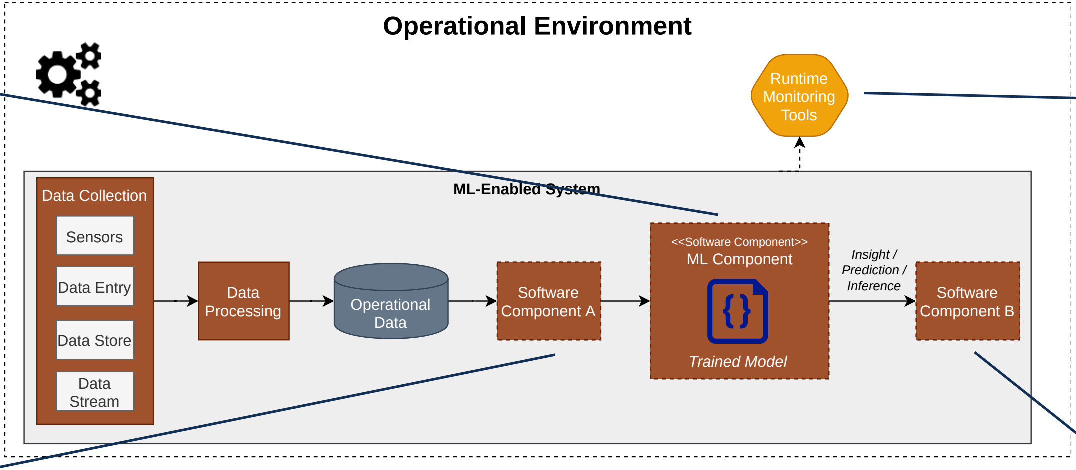
Characterizing and Detecting Mismatch in ML-Enabled Systems

ML-Enabled System

We define an ML-enabled system as a software system that relies on one or more ML software components to provide required capabilities.

A trained model(s) is wrapped as a special type of Software Component called an ML Component

ML component receives (processed) operational data from one software component …

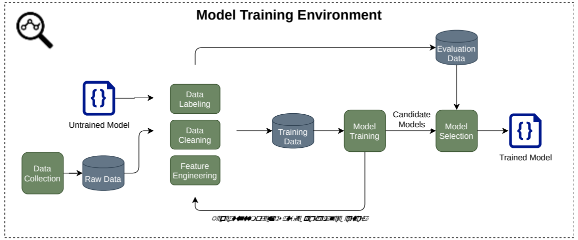


Runtime Monitoring Tools in the operational environment obtain and react to measures produced by the ML-Enabled System

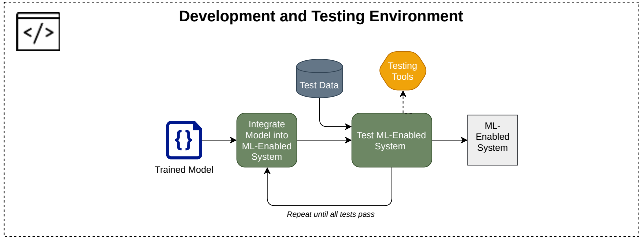
… and generates an insight for that data that is consumed by another software component.

Problem: Multiple Perspectives

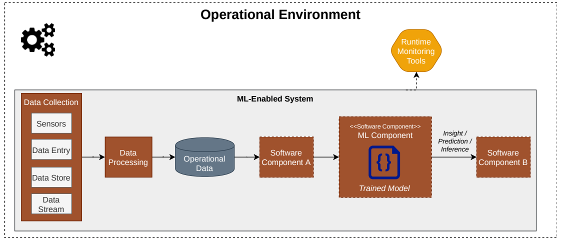
Data Scientist Perspective



Software Engineer Perspective



Operations Perspective



ML-enabled systems typically involve three different and separate workflows

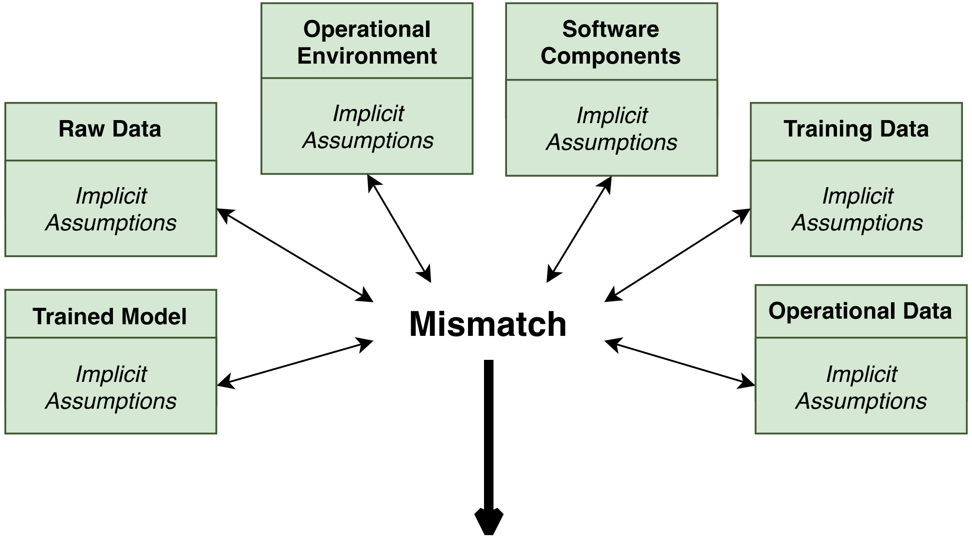
- Model training
- Model integration and testing
- Model operation

… performed by three different sets of stakeholders ...

- Data scientists
- Software engineers
- Operations staff

… with three different perspectives

Problem: Mismatch between Assumptions made by each Perspective



Late discovery of mismatch results in

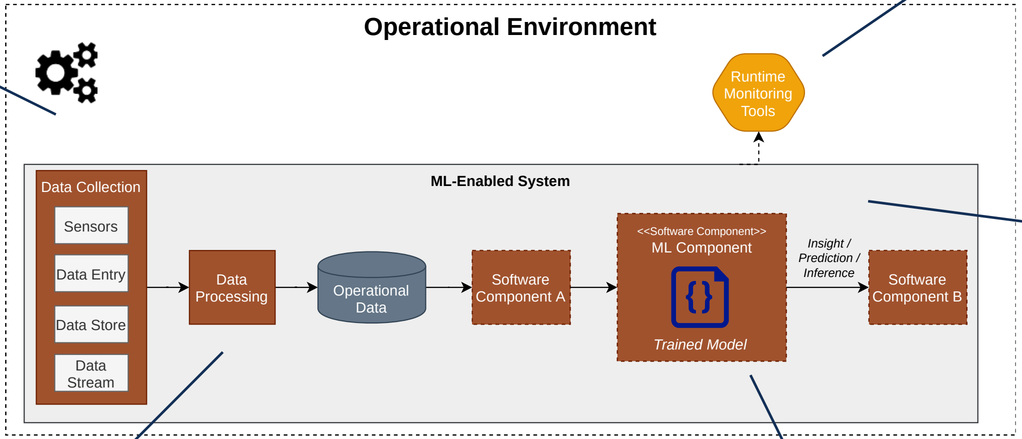
- System delivery delays due to rework
- Incorrect results
- Poor system performance
- **System or mission failure**

We define an **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 that results in a negative consequence.

We also posit that ML mismatch can be traced back to information that could have been shared between stakeholders that would have avoided the problem.

Examples of Mismatch

Poor system performance because computing resources for model testing different from operational computing resources (**computing resource mismatch**)



Tools not set up to detect diminishing model accuracy, which is the “goodness” metric defined for the ML component (**metric mismatch**)

System failure due to poor testing — developers not able to replicate testing done during model training (**test data mismatch**)

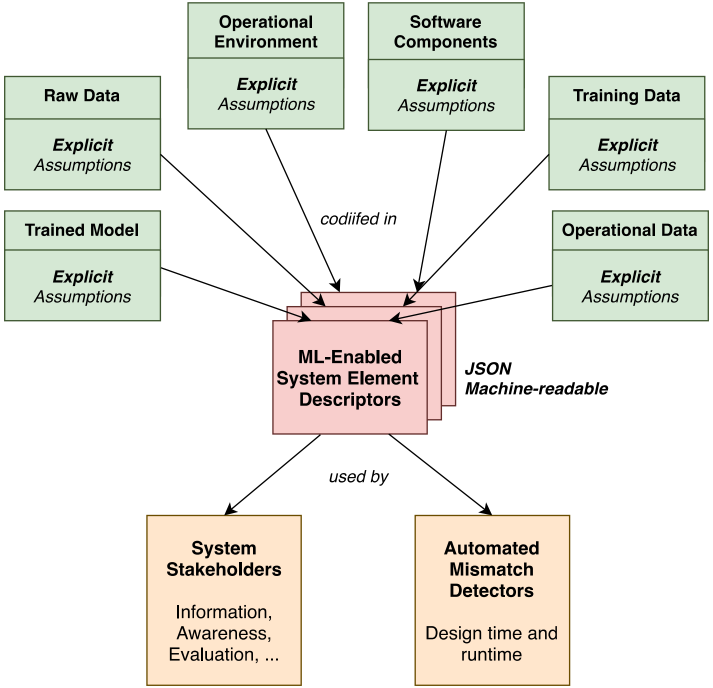
Poor model accuracy because model training data different from operational data (**data distribution mismatch**)

Large amounts of glue code because trained model input/output very different from operational data types (**API mismatch**)

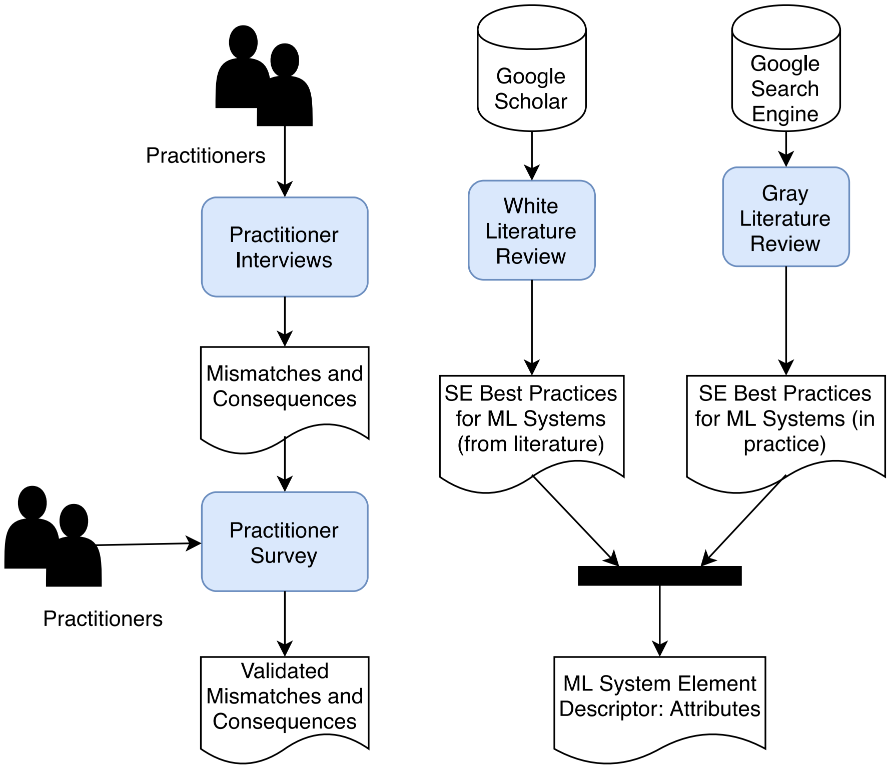
Solution: Mismatch Detection and Prevention in ML-Enabled Systems

Goal is to develop machine-readable descriptors for elements of ML-enabled systems that

- Can serve as checklists as ML-enabled systems are developed
- Provide stakeholders (e.g., program offices) with examples of information to request and/or requirements to impose
- Include attributes for which automated detection is feasible, and therefore define new software components that should be part of ML-enabled systems



Study Protocol — Phase 1

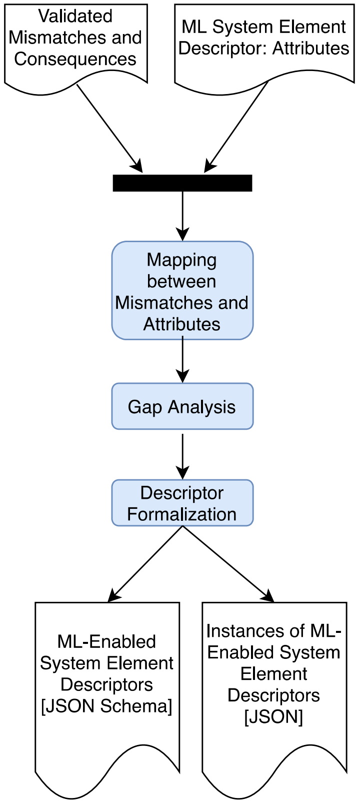


1. Identify examples of mismatches and their consequences via interviews
2. Validate mismatches via a practitioner survey

In parallel:

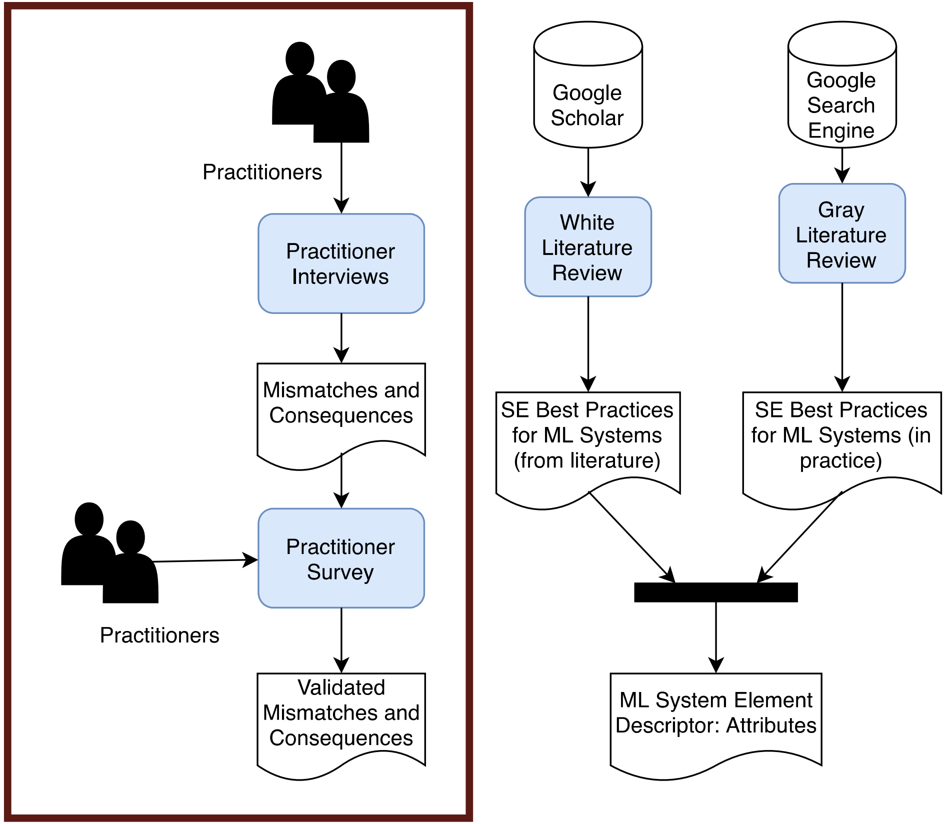
3. Identify attributes for describing elements of ML-enabled systems via a multi-vocal study
   - White Literature Review
   - Gray Literature Review

Study Protocol — Phase 2



1. Mapping between mismatches and attributes
   - For each mismatch, what is the set of attributes needed for detection, expressed as a predicate over identified attributes
2. Gap analysis
   - Which mismatches do not map to any attribute (and vice versa)?
   - What additional attributes are necessary for detection?
3. Descriptor Formalization
   - Codify attributes into a JSON Schema descriptor specifications
   - Create sample instances for descriptors

Scope of Today's Presentation



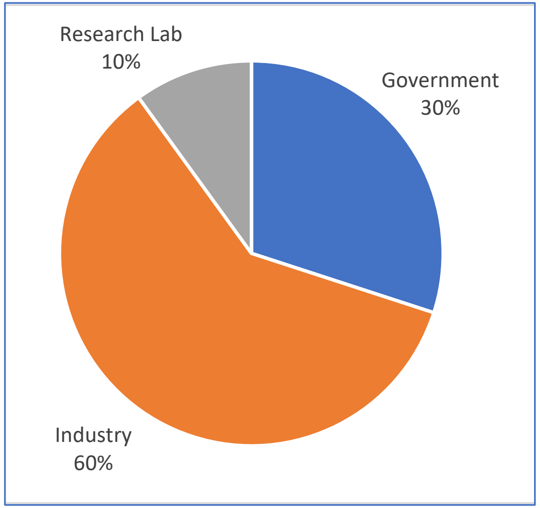
Interview — Data

Total Interviews = 20

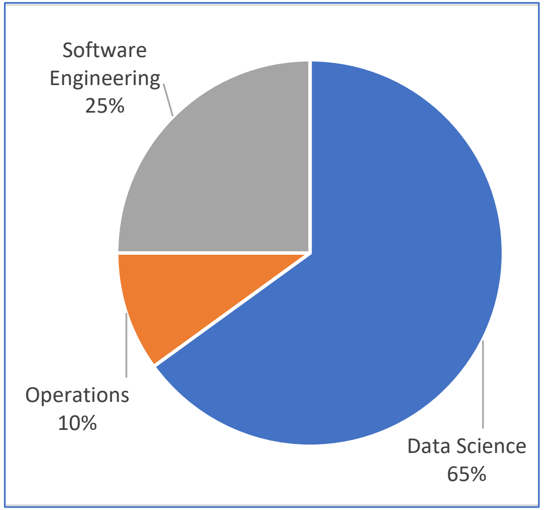
Total Mismatch Examples = 140

Total Instances of Information that was not Communicated that Led to Mismatch = 232

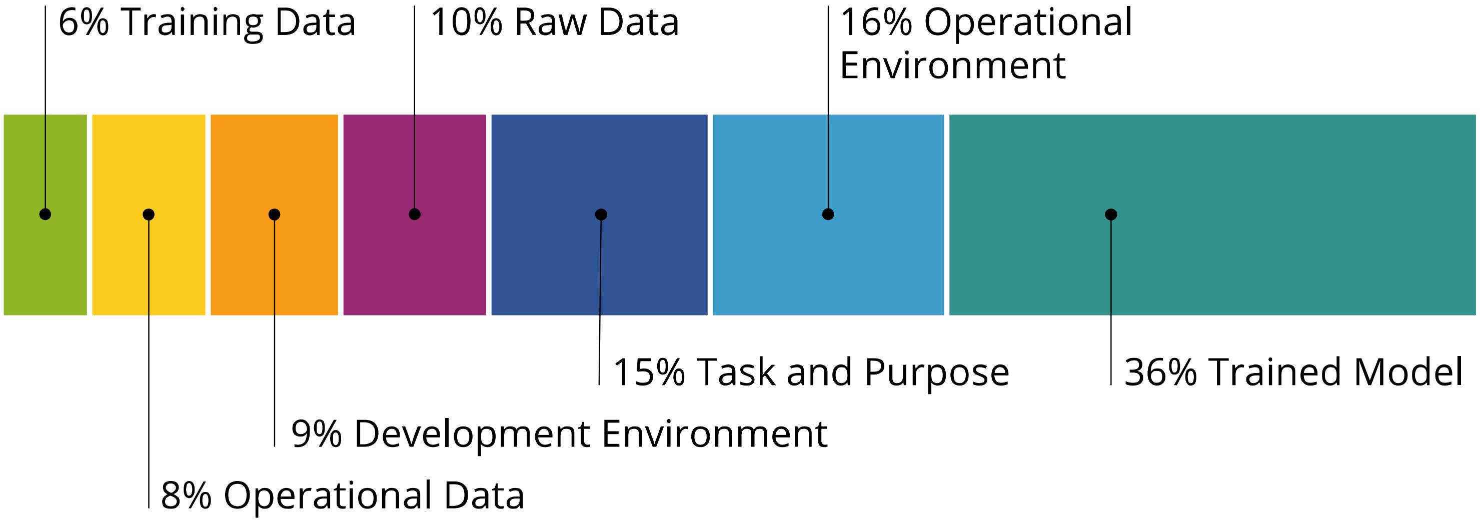
Affiliation



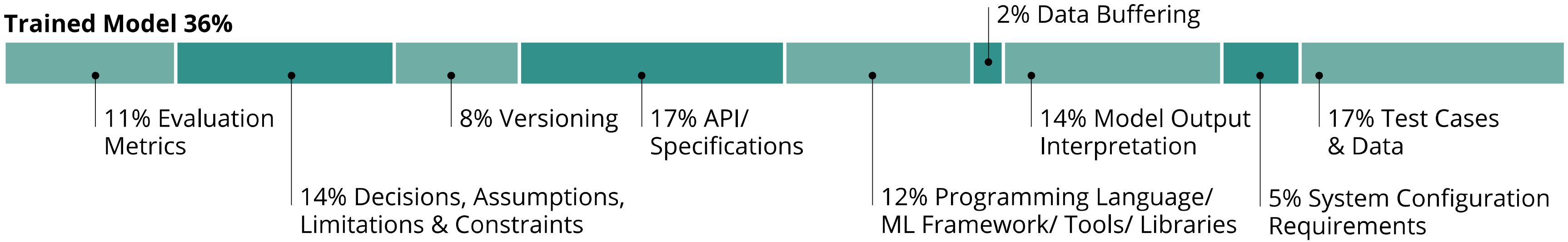
Primary Role



Results: Mismatch Categories



Trained Model Subcategories

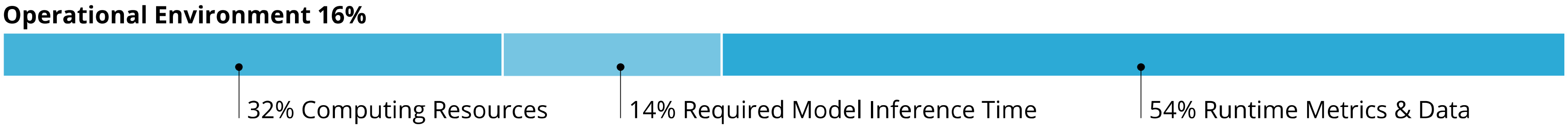


Most mismatches were related to

- lack of test cases and test data that could be used for integration testing
- lack of model specifications and APIs that provide greater insight into inputs, outputs, and internals (if applicable)

”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.”

Operational Environment Subcategories



Most mismatches were related to 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.

> *"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."*

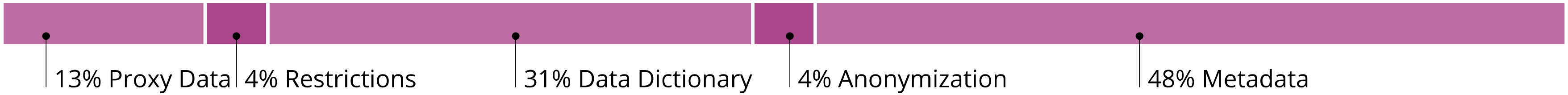
Task & Purpose Subcategories

Most mismatches were related to lack of lack of knowledge of business goals or objectives that the model was going to help satisfy.

”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.”

Raw Data Subcategories

Raw Data 10%

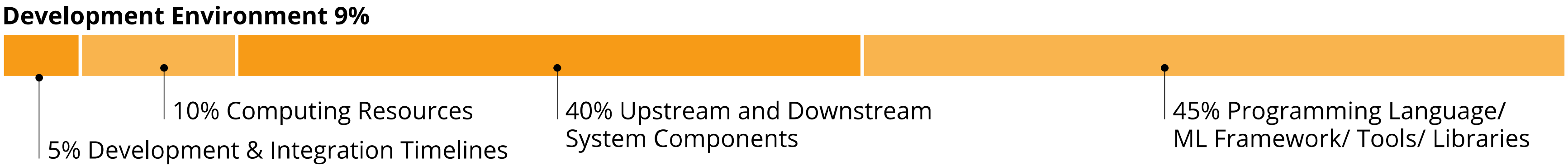


Most mismatches were associated with lack of

- metadata such as how it was collected, when it was collected, distribution, geographic location, and time frames
- description of data elements, such as field names, description, values, and meaning of missing or null values

> *"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?"*

Development Environment Subcategories

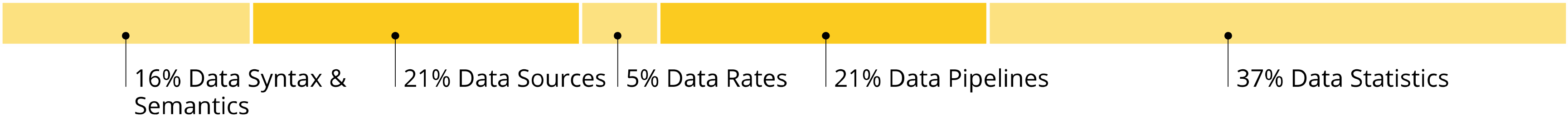


Most mismatches were related to lack of knowledge of programming languages, ML frameworks, tools, and libraries used in the development environment.

> *"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."*

Operational Data Subcategories

Operational Data 8%



Most mismatches were associated to

- lack of operational data statistics, such as distribution and other metrics, that could be used by data scientists to validate appropriateness of training data
- details on the implementation of data pipelines for the deployed model

> *"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."*

Training Data Subcategories

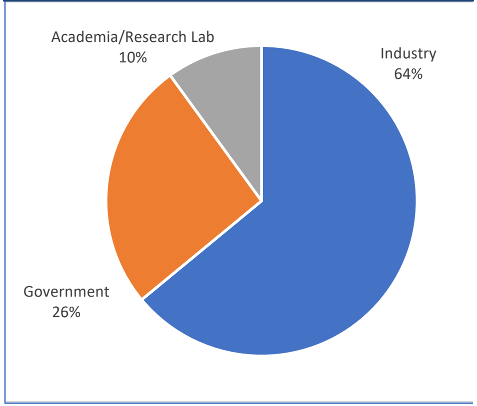
Most mismatches were related to lack of details of data preparation pipelines to derive training data from raw data.

”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.”

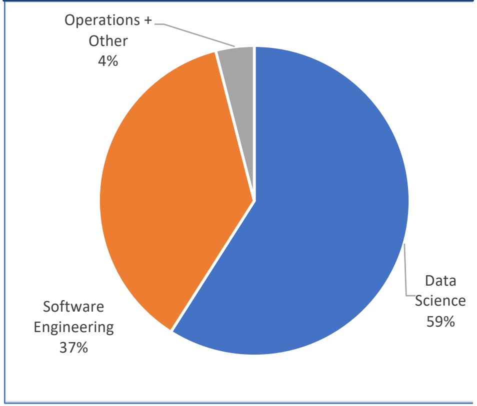
Mismatch Validation Survey — Data

Survey Responses = 31

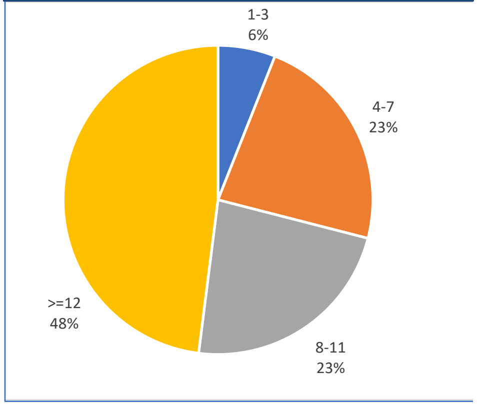
Affiliation



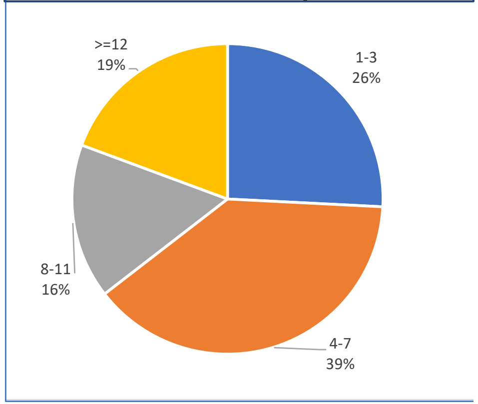
Primary Role



Years of Experience

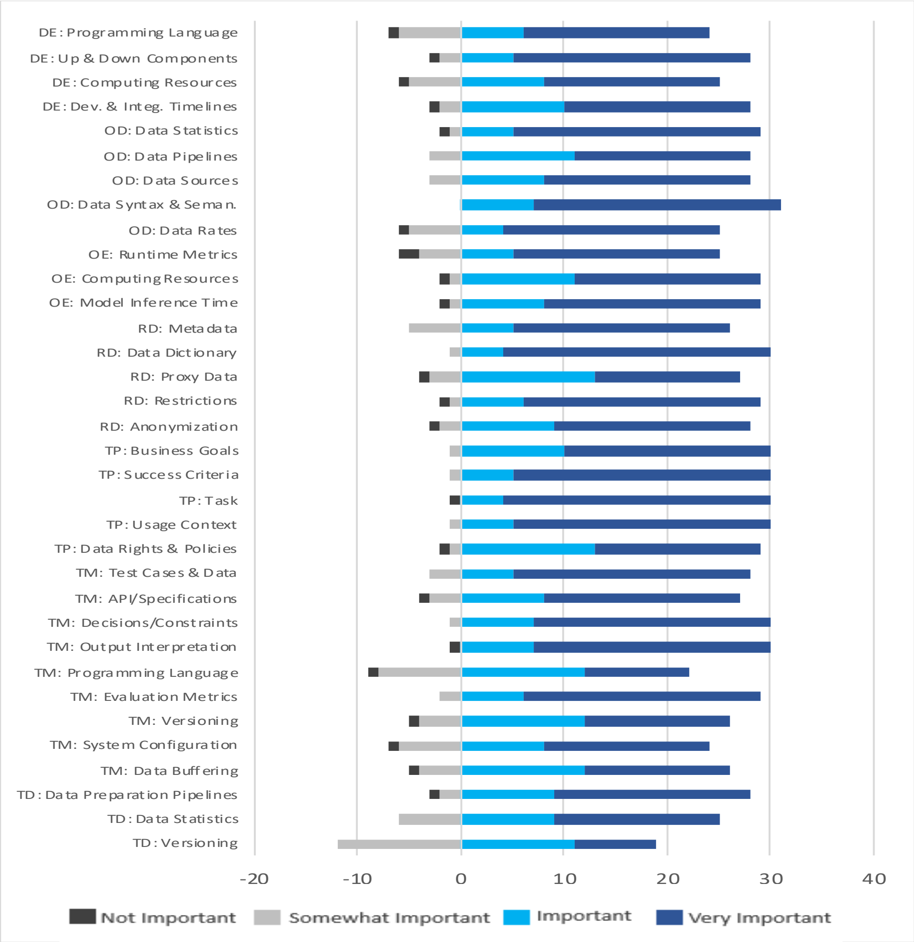


Years of ML Experience



Mismatch Validation Survey — Results

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.



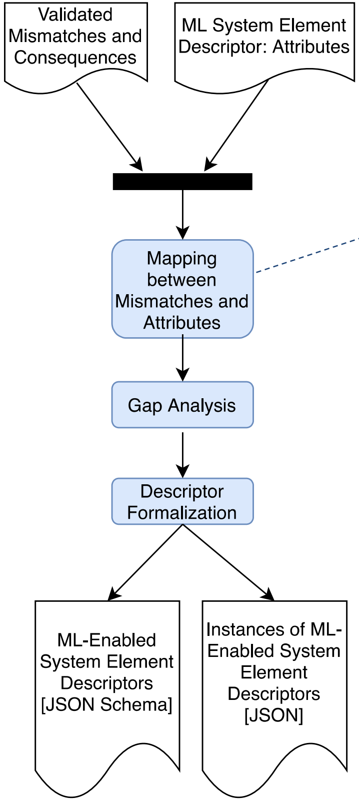
Mismatch Validation Survey — Observation

Not surprisingly, what is important varies per role …

… which makes it even more important to make all this information explicit.

	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

Next Steps: Develop Machine-Readable Descriptors

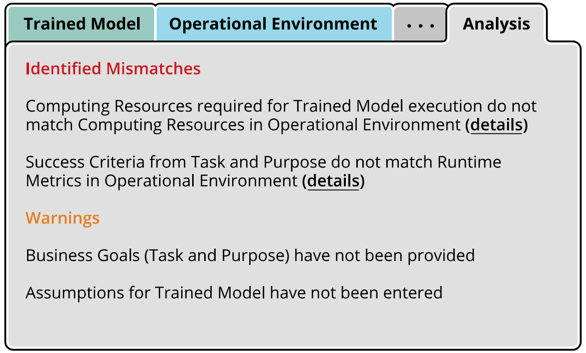


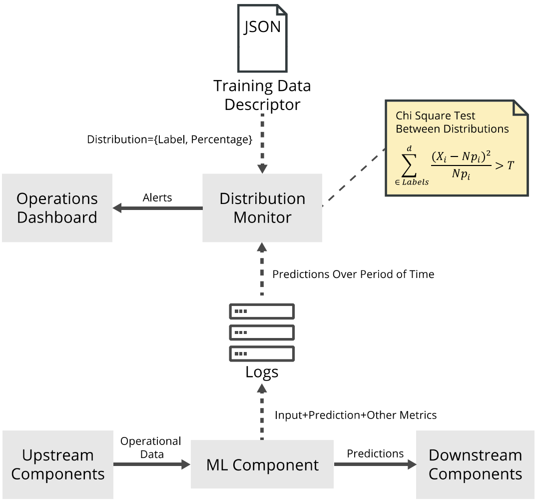
Mismatch	Descriptors																	Formalization
	TP		RD		TD			TM			DE			OD			OE	
	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	Am	
Mismatch 1	X	X		X														A1 + A2 > A4
Mismatch 2								X				X						A8 = A12
…																		
Mismatch N					X									X				Chi-Square(A5, A14)

Vision: Automated Mismatch Detection

Resulting attributes have been codified into machine-readable JSON Schema documents that can be used by automated mismatch detection tools.

Tools can range from a simple web-based client that reads in all descriptors and presents then to a user for evaluation, to a more elaborate tool or system component for runtime data drift detection.



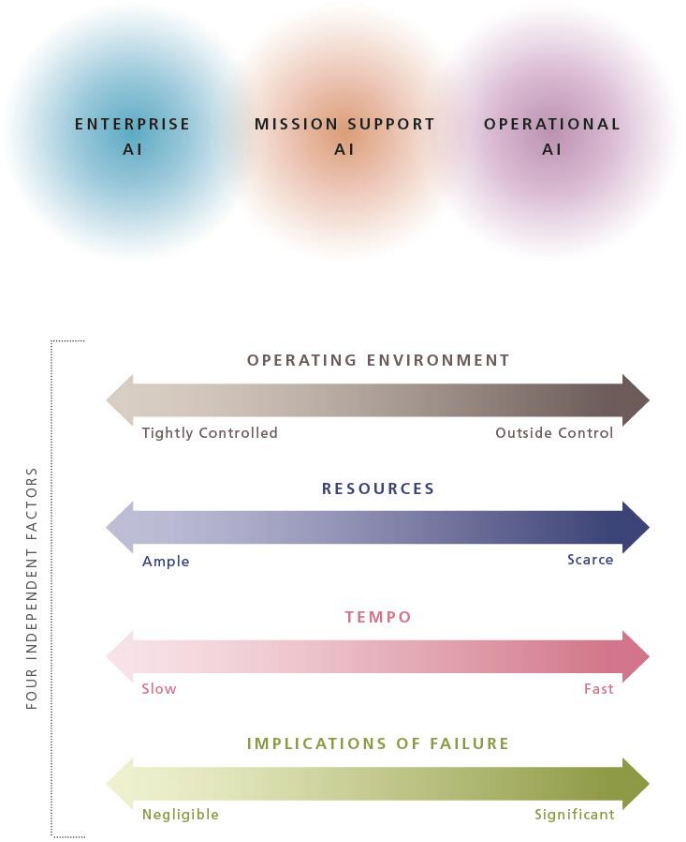


November 2020 – September 2021

Predicting Inference Degradation in Production ML Systems

Motivation

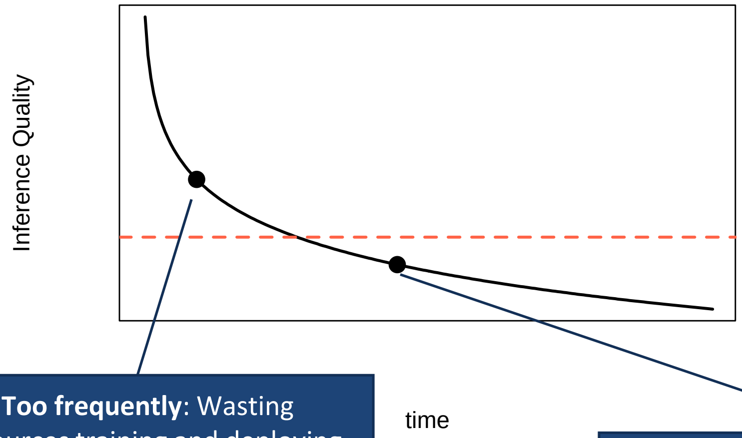
- Inference quality of deployed ML models changes over time due to differences between characteristics of training and operational data → **Inference Degradation**
- State of engineering practice in industry relies on periodic retraining and model redeployment strategies to evade inference degradation, as opposed to monitoring for inference degradation
- Strategy of periodic retraining and redeployment becomes more infeasible as DoD AI systems move into the Operational AI space

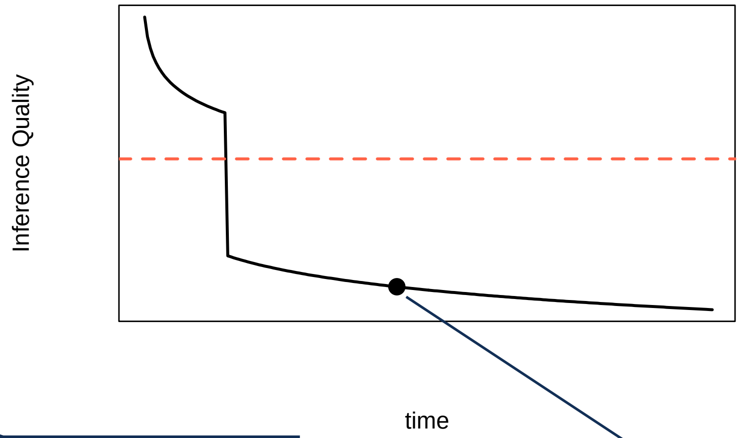


The Spectrum of DoD AI Applications*

* RAND Corporation. (2019). The Department of Defense Posture for Artificial Intelligence: Assessment and Recommendations. https://www.rand.org/pubs/research_reports/RR4229.html

Problem: Inference Degradation is Hard to Identify Timely and Reliably





Too frequently: Wasting resources training and deploying retrained models when it was not necessary to do so, in addition to normal redeployment risks

Too late: A non-optimal course of action recommended by an ML capability may have already been put in place by the time the model is retrained and redeployed

Not all degradation is gradual: ML capability had to be taken offline while the model was retrained and redeployed

Failure to recognize inference degradation can lead to misinformed decisions, costly reengineering, and potential system decommission.

Solution

Develop a set of empirically-validated metrics that are predictors of when a model's inference quality will degrade below a threshold due to different types of data drift, and therefore requires retraining.

The metrics will be able to determine

1. When a model really needs to be retrained so as to avoid spending resources on unnecessary retraining
2. When a model needs to be retrained before its scheduled retraining time so as to minimize the time that the model is producing sub-optimal results

Metrics will be validated in the context of models using convolutional neural networks (CNNs), which are commonly used in DoD applications for object detection.

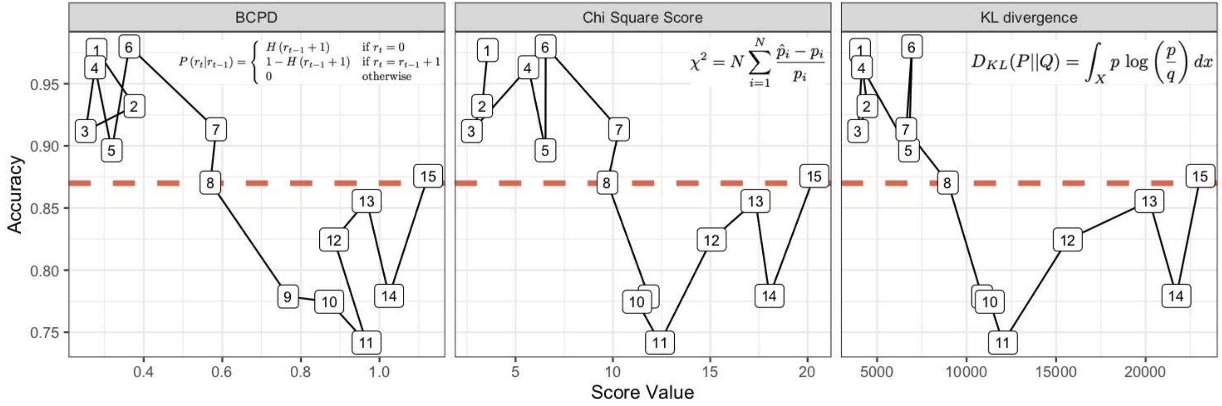
Approach

Create a test harness and data sets to baseline existing drift metrics.

Test the ability of single metrics to predict inference quality over time for models based on CNNs.

Develop complex metrics based on performance of single metrics.

Validate new metrics with respect to accuracy and timeliness.



Each graph illustrates the relationship between a change in the data drift metric and a change in the inference quality metric (in this case accuracy)

Towards Empirically-Validated SE4ML Practices

Development of novel metrics for predicting inference degradation for CNN-based models, data sets and extensible test harness software released as open source will improve timely and resource effective retraining of ML models

Definitions of mismatch can serve as checklists as ML-enabled systems are developed

Recommended descriptors provide stakeholders with examples of information to request and/or requirements to impose

Identification of attributes for which automated detection is feasible defines

- New software components that should be part of ML-enabled systems
- New tools for automated mismatch detection

Contact Information

Grace A. Lewis

Principal Researcher and TAS Initiative Lead

Tactical and AI-Enabled Systems

glewis@sei.cmu.edu

http://www/sei.cmu.edu/staff/glewis