Introducing The ML FMEA

A Safe Machine Learning Approach

Paul Schmitt, CSEP
Autonomy Software Systems & Safety, Senior Manager
TORC Robotics
paul.schmitt@torc.ai

Bodo Seifert
Senior Automotive Functional Safety Engineer
TÜV Rheinland
Bodo.Seifert@us.tuv.com

Jerry Lopez
Senior Director Safety Assurance
TORC Robotics
james.lopez@torc.ai

Mario Bijelic
Machine Learning Researcher
TORC Robotics
mario.bijelic@torc.ai

Krzysztof Pennar
Safety Principal Engineer
TORC Robotics
krzysztof.pennar@torc.ai

Felix Heide
Head of Artificial Intelligence
TORC Robotics
felix.heide@torc.ai

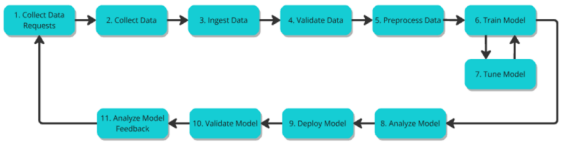

Figure 1a. The reference eleven-step ML Pipeline.

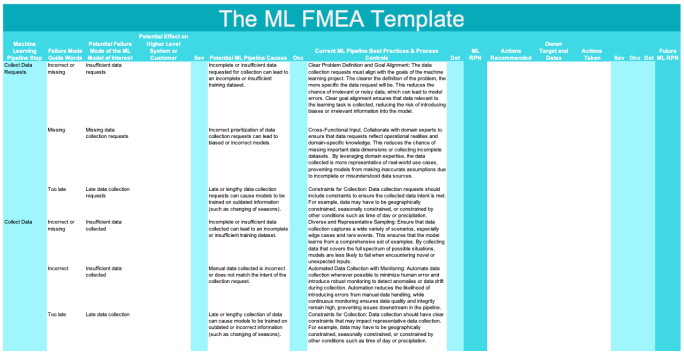

Figure 1b. Excerpt of the full ML FMEA Template containing the first two ML Pipeline steps: Collect Data Requests and Collect Data.

Abstract

The integration of Machine Learning (ML) into safety-critical applications continues to raise challenges related to risk management and standardization. This paper presents a structured approach to safe ML development, readily applicable to sectors such as automotive, autonomous vehicles and systems, defense, healthcare, pharmaceuticals, manufacturing, logistics, and aerospace. The proposed method addresses a current gap in existing AI and ML standards by combining established ML development practices with the Process Failure Mode and Effects Analysis (PFMEA) framework. This approach considers ML development as a holistic, iterative process, emphasizing the importance of

risk identification and mitigation throughout the pipeline rather than focusing exclusively on model performance. Each stage of a typical ML pipeline is examined, with associated failure modes and corresponding mitigations defined. To support practical implementation, the paper includes a pre-populated ML FMEA Template, designed to assist machine learning development teams in assessing, documenting, and communicating risks, while facilitating coordination with safety and assurance stakeholders.

Introduction

Incorporating machine learning into a safe product is a challenge within many industries today including automotive, autonomous vehicles, defense and security, healthcare, pharmaceuticals, manufacturing and industrial robotics, warehouse distribution, aerospace, etc. It presents a unique set of challenges that differ from those encountered in traditional, deterministic software systems. One primary challenge is the inherent complexity and opacity of ML models. Unlike conventional software, where decisions are based upon explicit, human-written code, ML models often operate as black boxes, making it difficult to understand how decisions are made. This lack of transparency can be problematic in safety-critical applications, where understanding the decision-making process is crucial for verifying and validating the system's behavior.

Another significant challenge is the variability in ML performance due to changes in input data. Machine learning models rely on data to learn and make predictions. However, if the operational environment differs from the training environment, the model's performance can degrade, leading to potentially unsafe decisions. Ensuring that the ML model can generalize well to new, previously unseen scenarios is vital, yet challenging, especially in safety-critical systems where failures can have severe consequences.

Additionally, the verification and validation (V&V) of ML components in safety-critical systems are complex and not yet standardized. Traditional V&V methods for software systems involve extensive testing and formal methods to prove correctness and reliability. However, applying these methods to ML models is difficult due to their probabilistic nature and the vastness of potential input scenarios. This necessitates the development of new V&V techniques tailored for ML, which can rigorously ensure the safety and reliability of these systems.

Regulatory and certification issues pose another challenge. Our automotive application has stringent regulatory requirements and standards that must be met. These standards are often based on the assumption of deterministic and well-understood system behaviors, which conflicts with the probabilistic and opaque nature of ML models.

Related Work

Standards Review

Several established standards that are widely used to guide the development and deployment of autonomous vehicles (e.g. ISO 21448 [1] and UL 4600 [2]) as well as emerging standards such as ISO PAS 8800 [3] make reference to the need to ensure the safety of ML applications. For example, the ISO 21448 Safety of the Intended Functionality (SOTIF) standard states that it is necessary to identify potential functional insufficiencies associated with the "specification of machine learning" as well as "measurement data for machine learning". Annex D of ISO 21448 describes the need to conduct an "analysis of off-line training process of machine learning algorithms" (Section D.2.5). UL 4600 also refers to several required activities in order to satisfy safety claims about ML algorithms used in autonomous vehicles. Some examples include:

1. Arguments that V&V procedures follow best practices for machine learning (8.5.2.2)
2. Evidence of suitable engineering rigor in the use of tools and techniques that are safety related (8.5.2.2)
3. Machine learning training and V&V shall use acceptable data (8.5.3)
4. Machine learning-based functionality shall be acceptably robust to data variation

The recently published technical regulation ISO TR 5469 [4] lists fault model methodologies including the performance of a FMEA “at the process level”. However, little guidance is provided. The emerging standard ISO PAS 8800 deals specifically with safety of ML algorithms similarly and prescribes the need for analyses to ensure that engineering rigor and best practices are applied to the ML pipeline.

In summary, while the above standards mention the need to systematically ensure safety of ML, they fail to prescribe specific techniques and methodologies to ensure that all engineering and V&V activities involving the holistic ML pipeline are performed in a satisfactorily rigorous manner in order to argue with sufficient evidence that the ML algorithm when deployed in an AV application is absent unreasonable risk.

Literature Review

In addition to the standards literature several notable and recent publications have begun exploration of ML within safety applications. Salay et al. [5] identified gaps within ISO 26262 Part 6 software development process and proposed new requirements to address. The requirements sought to increase the model’s functional clarity and interpretability. Studer et al. [6] adapt an established data mining approach for machine learning and propose a method for ML quality assurance. While notable, the topic of safety is not addressed. Faria’s (2018) [7] survey of ML safety presents ML characteristics that safety engineers should become familiar with in order to better understand potential failure modes such as Markov Decision Process and safe reinforcement learning issues. Mohseni et al. (2022) [8] provided a taxonomy of machine learning safety by linking key safety principles to machine learning safety limitations. They discuss three strategies that should be employed for safe machine learning; these are, inherently safe design, enhancing performance & robustness and run time error detection. Salay et al (2019) [9] applied an FMEA to an AI classification model and applied it to a AD case study. Most recently the International Systems Safety Society 2024 conference focused on machine learning safety. One example from this conference is the introduction of the STPAI where Murphy (2024) [10] applied STPA to an AI chatbot.

In summary, a literature review shows that the field of AI safety is continuing to grow and is moving from performance to quality assurance to safety via rethinking and adapting established formal safety analyses. None of the methods reviewed directly connect ML Pipeline step with specific ML failure modes with known mitigations.

The Contribution

This paper details two main contributions:

The ML FMEA Method. The contribution of this paper combines a proven method to mitigate risk in the automotive industry with machine learning best practices. It looks at machine learning as a process (i.e., the ML pipeline), rather than a model and applies the automotive Process Failure Mode & Effects Analysis (PFMEA) to identify, prioritize, and mitigate risk. The method directly connects ML Pipeline step with relevant failure modes with known mitigations.

The ML FMEA Template. Another contribution of this paper is providing the Machine Learning FMEA Template. The ML FMEA Template connects ML development pipeline

failure modes with machine learning best practices as mitigations. The intent of the template is to enable development teams to assess risk to the machine learning model development and tailor specific mitigations. Since the ML FMEA Template generally follows the Process FMEA flow, the approach is designed to be transparent and familiar to reviewers and experienced safety professionals.

Laying the Foundation

Before detailing the ML FMEA Method, a basic background on the machine learning development pipeline and the process failure analysis tool is needed.

The Machine Learning Pipeline

A typical machine learning pipeline is a structured process that begins with data collection, followed by preprocessing steps like cleaning, normalization, and feature engineering to prepare data for model training. Once the model is trained, tested, and optimized, it is deployed, with ongoing monitoring to ensure performance and retraining when necessary.

While there are several variants of the machine learning pipeline [11] [12] [13], this paper will utilize the eleven step pipeline shown in Figure 1 and referenced throughout the paper. To clarify the value addition of each pipeline step, the authors have phrased each pipeline step in the form of a function:

1. Collect Data Requests
2. Collect Data
3. Ingest Data
4. Validate Data
5. Preprocess Data
6. Train Model
7. Tune Model
8. Analyze Model
9. Deploy Model
10. Validate Model
11. Analyze Model Feedback

The Process FMEA

The Process Failure Mode and Effects Analysis (PFMEA) [14] [15] [16] [17] [18] is a proven method employed widely across the automotive, defense and military, energy, medical devices, pharmaceuticals, manufacturing, industrial robotics, and aerospace industries. A PFMEA is a systematic method used to identify and mitigate potential failure modes in processes within the automotive industry. Although its typical use is in manufacturing settings, it is often applied to processes in general. The primary objective of PFMEA is to enhance product quality, reliability, and safety by preemptively addressing process weaknesses that could lead to defects or failures. This analysis involves a detailed examination of each step in the process to identify possible failure modes, their causes, and their effects on the overall system. By assessing the severity, occurrence, and detection of each potential failure, PFMEA helps prioritize risks and develop effective countermeasures to prevent or control these failures.

The PFMEA process typically involves a cross-functional team comprising engineers and other relevant process stakeholders. Together, they systematically review each process step, brainstorming potential failure modes and documenting their findings. The team assigns a risk priority number (RPN) to each identified failure mode, calculated by multiplying the severity, occurrence, and detection ratings. High RPNs indicate areas that require immediate attention and corrective actions. By implementing these actions, such as redesigning process steps, enhancing process controls, or improving detection methods, the team aims to reduce the likelihood and impact of failures, thereby ensuring a more robust and reliable manufacturing process for automotive components. When a high RPN is identified the safety engineer prescribes a strategy to detect the failure and respond appropriately so that a safety critical system fails safe or fails operational.

The PFMEA process and associated artifacts are well established and understood across numerous industries. As such safety professionals are readily able to analyze the associated living documents and artifacts. They are able to identify high risk areas, areas needing corrective actions, and unmitigated risk gaps.

The ML FMEA Method

Here is a detailed breakdown of each step in the machine learning pipeline, including related potential failure points if the step is not executed well. Connected with these failure points are best practices or mitigation to minimize failures and ensure safe machine learning. The authors note that the mitigations are typically known to ML experts and deployed in an agile manner to address model performance deltas rather than from a holistic, proactive safety and failure mode method.

For added clarity, examples of mitigations are provided from an ML pipeline model development intended for use in an autonomous vehicle application.

Step 1: Collect Data Requests

Description: Data collection requests initiate the entire machine learning (ML) pipeline by specifying the types of data needed for model training and evaluation. The request defines the scope of the problem, the features to be used, and the relevant sources of data. This step is crucial because the quality, relevance, and availability of the data directly impact the model's performance and safety.

Potential Failure 1: Incomplete or insufficient data requested for collection can lead to an incomplete or insufficient training dataset.

- **Possible Mitigation: Clear Problem Definition and Goal Alignment.** The data collection requests must align with the goals of the machine learning project. The clearer the definition of the problem, the more specific the data request will be. This reduces the chance of irrelevant or noisy data, which can lead to model errors. Clear goal alignment ensures that data relevant to the learning task is collected, reducing the risk of introducing biases or irrelevant information into the model.

Potential Failure 2: Incorrect prioritization of data collection requests can lead to biased or incorrect models.

- **Possible Mitigation: Cross-Functional Input.** Collaborate with domain experts to ensure that data requests reflect operational realities and domain-specific knowledge. This reduces the chance of missing important data dimensions or collecting incomplete datasets. By leveraging domain expertise, the data

collected is more representative of real-world use cases, preventing models from making inaccurate assumptions due to incomplete or misunderstood data sources.

Potential Failure 3: Late or lengthy data collection requests can cause models to be trained on outdated information (such as changing of seasons).

- **Possible Mitigation: Constraints for Collection.** Data collection requests should include constraints to ensure the collected data intent is met. For example, data may have to be geographically constrained, seasonally constrained, or constrained by other conditions such as time of day or precipitation.

Step 2: Collect Data

Description: Data collection involves gathering raw data from identified sources such as sensors, databases, or user inputs. This is a critical step as the quality, volume, and diversity of the collected data will significantly influence the performance and generalization of the ML model.

Potential Failure 1: Incomplete or insufficient data collected can lead to an incomplete or insufficient training dataset.

- **Possible Mitigation: Diverse and Representative Sampling.** Ensure that data collection captures a wide variety of scenarios, especially edge cases and rare events. This ensures that the model learns from a comprehensive set of examples. By collecting data that covers the full spectrum of possible situations, models are less likely to fail when encountering novel or unexpected inputs.

Potential Failure 2: Manual data collected is incorrect or does not match the intent of the collection request.

- **Possible Mitigation: Automated Data Collection with Monitoring.** Automate data collection wherever possible to minimize human error and introduce robust monitoring to detect anomalies or data drift during collection. Automation reduces the likelihood of introducing errors from manual data handling, while continuous monitoring ensures data quality and integrity remain high, preventing issues downstream in the pipeline. An excellent example of automated data collection for autonomous vehicle model development is active learning with language embedding. [19]

Potential Failure 3: Late or lengthy collection of data can cause models to be trained on outdated or incorrect information (such as changing of seasons).

- **Possible Mitigation: Constraints for Collection.** Data collection should have clear constraints that may impact representative data collection. For example, data may have to be geographically constrained, seasonally constrained, or constrained by other conditions such as time of day or precipitation.

Step 3: Ingest Data

Description: Data ingestion is the process of collecting and importing data from various sources for immediate use or storage in a database. This step ensures that comprehensive and reliable data is gathered, forming the foundation for the entire machine learning pipeline.

Potential Failure 1: Incomplete or inaccurate data collection can lead to biased or incorrect models.

- **Possible Mitigation: Automate the Ingestion Process.** Utilize robust ETL (Extract, Transform, Load) tools and frameworks to automate data collection. Automation ensures consistent and error-free data collection, preventing gaps and inconsistencies that could compromise model performance. Automation minimizes human error, ensuring that data is ingested accurately and efficiently, thus reducing the risk of introducing incomplete or erroneous data.

Potential Failure 2: Security breaches during data ingestion can compromise sensitive data, leading to ethical and legal issues.

- **Maintain Data Security.** Secure data transmission through encryption and ensure compliance with data protection regulations. Use access controls and audit logs to monitor data access. Protecting sensitive information during ingestion prevents unauthorized access and ensures the trustworthiness of the data used for training.

Potential Failure 3: Delays in data ingestion can cause models to be trained on outdated information.

- **Possible Mitigation: Ensure Data Quality.** Implement initial data checks for integrity, accuracy, and completeness. Tools like Apache Griffin or Great Expectations can automatically detect and rectify anomalies. Ensuring data quality from the start reduces the risk of the model learning incorrect patterns, improving the reliability of the model's predictions.

Additional Examples: Data quality issues during the data ingestion process in autonomous driving systems often stem from synchronization, communication, and data format inconsistencies. Here is a breakdown of common challenges and mitigation strategies from an autonomous vehicle application example. Note that these example challenges are likely common across other industries such as defense, aerospace, medical device, pharmaceutical, etc., that employ Machine Learning within a complex system with redundant or complementary sensing systems.

- **Sensor Synchronization Issues**: Misalignment in the timing of data collected from multi-modal sensors, such as cameras, RADAR and LiDAR, can result in inconsistencies. To mitigate this, time-stamping sensor data and utilizing real-time synchronization methods help align the data more accurately. Additionally, leveraging multi-modal redundancy—cross-referencing data from different sensors—can identify and correct temporal misalignments. [20] [21]
- **Dropped Messages**: Data packet loss during transmission can create gaps in the data stream, potentially missing crucial information. Buffering and retry mechanisms are effective in ensuring that lost packets are re-transmitted. Additionally, multi-modal redundancy, using other sensors to verify missing data, can help bridge these gaps.
- **Data Format Inconsistencies**: Data from different sensors often comes in various formats, resolutions, or coordinate systems (autonomous vehicle example: point clouds from LiDAR, images from cameras, and radar readings). Furthermore during the development process the respective firmware get updated, leading to increased capabilities over the development cycle. These inconsistencies can create challenges during data fusion, leading to misinterpretations or loss of information. Implementing standardized data formats, using preprocessing pipelines to align resolutions, and running consistency checks can ensure uniformity across sensor data, enabling seamless integration in later stages. [22]

These strategies focus on ensuring the consistent collection, transmission, and formatting of sensor data, creating a robust foundation for accurate analysis in the subsequent stages of the autonomous ML pipeline.

Step 4: Validate Data

Description: Data validation involves verifying that the collected data meets quality standards and is suitable for analysis. This step ensures the data is accurate, complete, and free from significant errors before it progresses further in the pipeline.

Potential Failure 1: Invalid or corrupt data can lead to erroneous model training and predictions.

- **Possible Mitigation: Schema Validation.** Enforce data structure and type constraints through schema definitions. Tools like JSON Schema or Apache Avro can automate schema validation. Schema validation catches and corrects errors early, preventing structural inconsistencies that could lead to model misinterpretation and incorrect learning.

Potential Failure 2: Undetected anomalies can introduce biases and reduce model performance.

- **Possible Mitigation: Anomaly Detection.** Implement automated checks to detect and handle anomalies such as outliers, missing values, and duplicate records. Identifying and mitigating anomalies ensures the model learns from clean, consistent data, reducing the risk of learning misleading patterns. A good example of anomaly detection for autonomous driving is the survey method. [23]

Potential Failure 3: Lack of validation can result in using incompatible or irrelevant data for training.

- **Possible Mitigation: Consistent Monitoring.** Continuously monitor data quality metrics and set up alerts for significant deviations. Early detection and rectification of data issues help maintain the consistency and reliability of the data, reducing the risk of model degradation.

Additional Examples: Validation processes ensure that ingested data maintains high quality, catching issues such as structural inconsistencies, projection errors, and anomalies. Key challenges and strategies include:

1. **Schema Validation**: Enforcing data structure and type constraints through schema definitions ensures that the ingested data adheres to expected formats. Tools like JSON Schema or Apache Avro can automate schema validation, making it easier to detect and correct structural inconsistencies. Validating the schema in the datalake helps catch errors early, preventing issues like unexpected data structures that could lead to model misinterpretation and incorrect learning.
2. **Anomaly Detection**: Automated checks for anomalies such as outliers, missing values, and duplicate records can help ensure the integrity of the data. These can be achieved by applying a large scale offline digital twin or model replaying the collected samples for comparison. This allows early detection of misbehaviors and provides important data samples for model refinement.
3. **Consistency Check**: Another source of incompatible data in complex systems are sensor measurement discrepancies. *Aerospace Example:* Inconsistent readings between redundant systems, such as differences in altitude measurements between two independent altimeters, may spawn autopilot misbehavior or fail-safe activation. *Manufacturing Example:* Within a manufacturing line sensors and monitoring systems such as temperature and pressure readings of a casting process. *Sterile*

Pharmaceutical Line Example: Air particle counters and sterility test results may not align. *Autonomous Vehicle Example:* An autonomous driving stack involves multi-modal sensors such as LiDAR and stereo cameras which may not align and impact perception accuracy. During validation, cross-referencing multi-modal data ensures accurate alignment and data integrity.

4. **Dirty Sensor Prevention**: Physical contaminants like dust or moisture on sensor lenses can distort data. Validation systems that compare current sensor readings against pre-recorded baselines can identify deviations caused by dirty sensors. Automated cleaning systems can mitigate this issue by keeping sensor surfaces clear.
5. **Sensor Configuration Check**: Misalignment or calibration errors between sensors can lead to measurement inaccuracies. Automated calibration tools and validation checks during data ingestion ensure consistent alignment. *Aerospace Example:* Misaligned or improperly calibrated sensors, such as those measuring pitch, roll, or yaw may compromise flight control system accuracy. *Manufacturing Example:* Misalignment of robotic arms or tools may lead to imprecise assembly or measurements. *Pharmaceutical Example:* Misaligned sensors or improperly calibrated pipetting systems used in liquid formulations may lead to incorrect ingredient volumes. *Autonomous Vehicle Example:* Cross-referencing depth information from multiple sensor types, like LiDAR and stereo cameras, further aids in validating sensor alignment. Mis-calibrated datasets are marked and further refined in pre-processing steps.
6. **Software Continuous Integration**: Software bugs or incompatibilities in sensor firmware can introduce data inconsistencies. Continuous integration testing and version control, paired with offline perception systems, help detect and address such anomalies during validation.

A powerful data validation process can be applied by implementing the above described steps. Thereby, identifying errors early on ensures high quality datasets for downstream machine learning. Further continuous monitoring alongside cross-referencing methods and automated calibration can help to spot issues in the data collection fleet early and fix issues in the upstream processing steps.

Step 5: Preprocess Data

Description: Data preprocessing involves cleaning and transforming raw data into a format suitable for model training. This step includes handling missing values, normalizing or scaling features, encoding categorical variables, and feature engineering.

Potential Failure 1: Poor handling of missing values can introduce biases.

- **Possible Mitigation: Standardize Data Cleaning Procedures.** Establish and follow standardized procedures for handling common data issues like missing values and outliers. Standardizing data cleaning procedures ensures consistency and reliability in the data used for training, reducing the risk of introducing biases and errors. As a healthcare industry approach example, active label cleaning is a proven approach to clean noisy annotation labels. [24]

Potential Failure 2: Incorrect normalization or scaling can distort relationships in the data.

- **Possible Mitigation: Automate Feature Engineering.** Use automated feature engineering tools like Feature tools to systematically create and evaluate new features. Automation reduces the risk of overlooking critical data transformations, ensuring that the model captures all relevant information.

Potential Failure 3: Inadequate feature engineering can lead to suboptimal model performance.

- **Possible Mitigation: Document Transformations.** Keep detailed records of all transformations applied to the data. Documentation ensures reproducibility and facilitates debugging, helping identify and correct preprocessing steps that may introduce errors. As an example for autonomous vehicle environmental sensing via lidar, LidarAugment can be employed to augment 3D objects for robust detection. [25]

Step 6: Train Model

Description: Model training involves using preprocessed data to train machine learning models. This step includes selecting appropriate algorithms, configuring model parameters, and fitting the model to the training data.

Potential Failure 1: Overfitting or underfitting can occur if the model is not trained properly.

- **Potential Mitigation: Use Cross-Validation.** Employ techniques like k-fold cross-validation to ensure the model's performance is consistent across different data subsets. Cross-validation provides a more reliable estimate of the model's generalization ability, reducing the risk of overfitting.

Potential Failure 2: Incorrect algorithm selection can lead to poor model performance.

- **Potential Mitigation: Hyperparameter Optimization.** Automate hyperparameter tuning with tools like Grid Search, Random Search, or Bayesian Optimization. Proper tuning ensures the model performs optimally, preventing underfitting or overfitting.

Potential Failure 3: Poor parameter configuration can prevent the model from learning effectively.

- **Possible Mitigation: Monitor Training Process.** Track training metrics such as loss and accuracy in real-time. Tools like Tensor Board provide visual insights into the training process. Real-time monitoring allows for early detection of issues and timely intervention, ensuring the model trains correctly.

Step 7: Tune Model

Description: Model tuning involves fine-tuning the trained model to improve its performance. This step includes adjusting hyperparameters, selecting features, and potentially re-training the model with updated configurations.

Potential Failure 1: Suboptimal hyperparameter settings can degrade model performance.

- **Possible Mitigation: Systematic Hyperparameter Tuning.** Use systematic search methods or automated tools for hyperparameter tuning. Techniques like Bayesian Optimization or Hyperband systematically explore the hyperparameter space. Systematic tuning ensures optimal model performance and reduces the risk of suboptimal configurations that could degrade performance.

Potential Failure 2: Irrelevant or redundant features can increase model complexity and reduce accuracy.

- **Possible Mitigation: Feature Selection.** Evaluate the importance of features and remove irrelevant or redundant ones. Techniques like Recursive Feature Elimination (RFE) or LASSO can help. Feature selection reduces model complexity and improves generalization, preventing overfitting and enhancing accuracy.

Potential Failure 3: Lack of proper evaluation can result in a tuned model that does not generalize well.

- **Evaluate on Validation Set:** Use a separate validation set to assess the performance of the tuned model. Proper evaluation prevents overfitting on the training data, ensuring the model's reliability in real-world applications.

Step 8: Analyze Model

Description: Model analysis involves evaluating the performance of the model using various metrics and techniques. This step helps understand the model's behavior and identify areas for improvement.

Potential Failure 1: Over-reliance on a single metric can provide an incomplete picture of model performance.

- **Possible Mitigation: Comprehensive Metrics.** Use a range of evaluation metrics to cover different aspects of model performance. Multiple metrics provide a holistic understanding of model strengths and weaknesses, preventing optimization for a single aspect that might not capture all performance facets.

Potential Failure 2: Failure to conduct thorough error analysis can leave critical issues unaddressed.

- **Possible Mitigation: Error Analysis.** Conduct a thorough analysis of the model's errors to identify limitations and areas for improvement. Understanding misclassification patterns helps implement targeted refinements, preventing repeated mistakes and improving overall model accuracy.

Potential Failure 3: Lack of clear visualizations can make it difficult to interpret and act on model performance data.

- **Possible Mitigations: Visualizations.** Use visual tools like confusion matrices, ROC curves, and precision-recall curves to provide clear insights into the model's performance. Visualizations facilitate better interpretation and decision-making, helping identify and correct potential issues.

Step 9: Deploy Model

Description: Model deployment involves integrating the trained model into a production environment where it can start making predictions on live data. This step requires careful planning to ensure the model operates efficiently and reliably in production.

Potential Failure 1: Inadequate infrastructure can lead to performance bottlenecks.

- **Possible Mitigation: Continuous Integration / Continuous Deployment (CI/CD).** Implement CI/CD pipelines to automate the deployment process. CI/CD ensures smooth updates and minimizes manual errors, preventing deployment failures and maintaining model consistency.

Potential Failure 2: Poor monitoring can result in undetected performance degradation.

- **Possible Mitigation: Monitoring and Logging.** Continuously monitor the deployed model performance and log predictions. Set up alerts and analyze logs to detect issues early. Early detection allows for prompt intervention, preventing prolonged periods of poor performance and maintaining the model's reliability.

Potential Failure 3: Lack of rollback mechanisms can make it difficult to address issues post-deployment.

- **Possible Mitigation: Implement Software Rollback.** A software rollback feature can revert the software back to a known release with known limitations that may have adequate mitigations.

Potential Failure 4: Integration can mask model failures and introduce integration failures

- **Possible Mitigation: Performance Thresholds.** Define and adhere to performance thresholds that the model must meet before deployment. Ensuring the model meets required standards prevents the release of suboptimal models, maintaining high performance and reliability.

Potential Failure 5: Scaling of the deployed model can introduce unaccounted for effects.

- **Possible Mitigation: Scalability and Reliability.** Use scalable and reliable infrastructure to host the model. Cloud services like AWS, GCP, or Azure provide robust options. Scalable infrastructure ensures the model can handle varying loads and maintain performance, preventing downtime and degraded performance.

Potential Failure 6: Model learns incorrectly through on-line learning and failure is not identified through validation

- **Possible Mitigation: Do not enable on-line learning.** On-line learning can lead to unvalidated content in the algorithm which can cause unwanted output. By not enabling on-line learning additional deployment and testing cycles may be necessary but it will mitigate deploying untested algorithms that may have undesired outputs.

Step 10: Validate Model

Description: Model validation involves confirming that the model performs well on unseen data and meets required performance standards. This step typically involves using a test dataset or conducting a separate validation phase.

Potential Failure 1: Overfitting to the training data can result in poor performance on new data.

- **Possible Mitigation: Holdout Validation.** Use a holdout validation set or cross-validation to ensure the model's performance generalizes well to new data. This practice provides a realistic estimate of how the model will perform in real-world scenarios, reducing the risk of overfitting.

Potential Failure 2: Validation on an unrepresentative test set can provide a false sense of model reliability.

- **Possible Mitigation: Real-World Testing.** Validate the model with real-world data, if possible. Real-world testing highlights discrepancies between the training

environment and real-world scenarios, preventing unexpected failures post-deployment.

Potential Failure 3: Validation to a dependent or derivative test set can provide a false sense of model reliability.

- **Possible Mitigation: Independent data set.** Ensure an independent data set is collected, created, and curated for validation.

Potential Failure 4: Ignoring real-world scenarios can lead to unexpected failures post-deployment.

- **Possible Mitigation: Real-World Testing.** See potential failure 2.

Step 11: Analyze Model Feedback

Description: Model feedback involves collecting and analyzing the performance of the deployed model in the real world. This step helps identify any drift or degradation in model performance and provides insights for further improvements.

Potential Failure 1: Lack of feedback can result in undetected performance drift.

- **Possible Mitigation 1: Feedback Loops.** Establish feedback loops to collect data on the model's predictions and outcomes. Continuous learning and improvement ensure that the model adapts to changing conditions, maintaining its accuracy and relevance.
- **Possible Mitigation 2: Monitor for Drift.** Continuously monitor for data and concept drift to ensure early detection of performance degradation. Tools like Alibi Detect can identify changes in data dynamics. Monitoring for drift allows for timely retraining and updates to the model, preventing long-term degradation.
- **Possible Mitigation 3: Performance indicators**. Establish clear safety performance indicators that can capture ML issues. Safety performance indicators that have an established range can identify emerging issues with a model before they become a hazard.

Potential Failure 2: Ignoring user feedback can lead to models that do not meet user needs.

- **Possible Mitigation: User Dashboard**. Establish performance metrics from a user perspective. Establish an appropriate frequency to capture user feedback as well as the dashboard presentation view for key decision makers.

Potential Failure 3: Delayed updates can cause the model to become obsolete.

- **Possible Mitigation: Regular Updates.** Schedule regular updates and retraining sessions for the model to incorporate new data and maintain performance. Regular updates ensure the model stays current with the latest data, preventing obsolescence and maintaining high performance.

Summary

By following these best practices at each step of the machine learning pipeline, organizations can create a safer and more reliable machine learning system, minimizing failures and ensuring high-quality performance throughout the model's lifecycle.

The ML FMEA Template

Another contribution is the ML FMEA Template. The ML FMEA Template connects ML development pipeline failure modes with ML best practices as mitigations. In many ways, the ML FMEA Template is framing and connecting the material above within a familiar, tabular format. The intent of the ML FMEA Template is to enable development teams to identify and assess risk to the machine learning model development and tailor specific mitigations. Additionally, it provides familiar transparency to safety assessors.

Common Columns

Most ML FMEA columns are common with a typical Process FMEA template. These include Severity, Potential Effect on Higher Level System or Customer, Occurrence, Detection, RPN, Guide Words, Actions Recommended, and Owner. The utilization of these columns is unchanged from standard practice. See Figure 2 for the visualization.

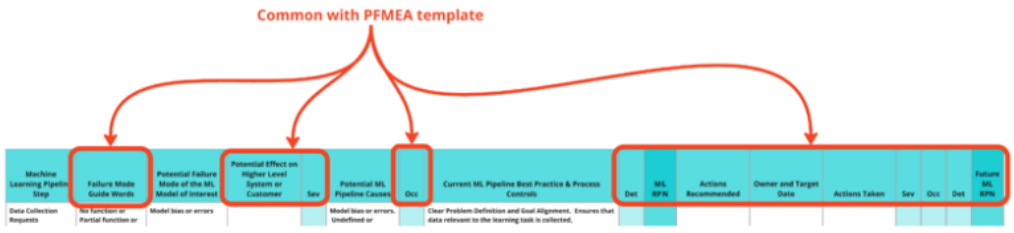

Figure 2. ML FMEA columns common with a typical Process FMEA template.

Modified Columns

Columns that are modified or pre-populated within the ML FMEA Template include the following.

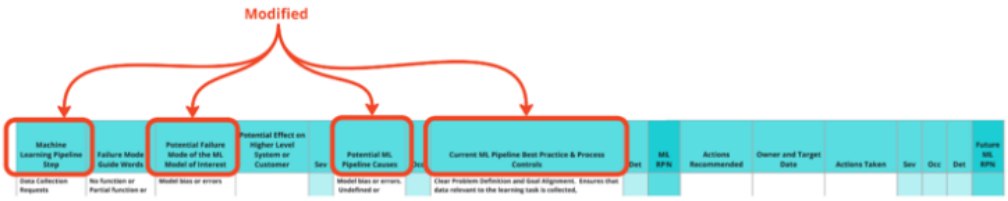

Figure 3. ML FMEA columns that are in common with a typical Process FMEA template.

Table 1. Tabulation of the ML FMEA columns, the correlating Process FMEA columns, and the modifications.

Column	Correlation	Change
Machine Learning Pipeline Step	This column maps to the PFMEA Process Step/Function column.	This column is populated with the steps of the ML Pipeline.
Potential Failure Mode of the ML Model of Interest	This column maps to the PFMEA Potential Failure Mode column.	This column is populated with failure modes relevant to the specific ML Pipeline step.
Potential ML Pipeline Causes	This column maps to the PFMEA Potential Effects of Failure column.	This column is populated with specific potential causes of the ML model failure modes.
Current ML Pipeline Best Practice or Process Control	This column maps to the PFMEA Current Controls	This column is populated with ML Pipeline best practices or process controls that potentially mitigate the particular ML failure mode.

Example rows

To clarify the benefits of the ML FMEA Template, a few example rows are provided from the Ingesting Data pipeline step.

As described within the ML FMEA Method Preprocess Data pipeline process step above a potential failure of Preprocess Data is “Poor handling of missing values can introduce biases”. See Figure 4. Within the ML FMEA Template this actually maps to a cause the Preprocess Data failure mode of “Insufficient data processing”, from the guide word “Missing”. Within the Template, the ML Pipeline Best Practice is “Standardize Data Cleaning Procedures:..” as described above.

From the guide word “Incorrect”, the next Preprocess Data failure mode “Insufficient data preprocessing” cause is “Incorrect normalization or scaling can distort relationships in the data”. Within the ML FMEA Template this is linked to the ML Pipeline Best Practice, “Automate Feature Engineering:…” as described above.

The third Preprocess Data failure mode, from the guide phrase “Too little” is “Insufficient data preprocessing”. The associated cause is “Inadequate feature engineering can lead to suboptimal model performance”. Within the ML FMEA Template this is linked to the ML Pipeline Best Practice, “Document Transformations”. See Figure 4.

Machine Learning Pipeline Step	Failure Mode Guide Words	Potential Failure Mode of the ML Model of Interest	Potential Effect on Higher Level System or Customer	Sev	Potential ML Pipeline Causes	Occ	Current ML Pipeline Best Practices & Process Controls	Det	ML RPN
Preprocess Data	Missing	Insufficient data preprocessing			Poor handling of missing values can introduce biases.		Standardize Data Cleaning Procedures: Establish and follow standardized procedures for handling common data issues like missing values and outliers. Standardizing data cleaning procedures ensures consistency and reliability in the data used for training, reducing the risk of introducing biases and errors. As a healthcare industry approach example, active label cleaning is a proven approach to clean noisy annotation labels.		
	Incorrect	Insufficient data preprocessing			Incorrect normalization or scaling can distort relationships in the data.		Automate Feature Engineering: Use automated feature engineering tools like Feature tools to systematically create and evaluate new features. Automation reduces the risk of overlooking critical data transformations, ensuring that the model captures all relevant information.		
	Too little	Insufficient data preprocessing			Inadequate feature engineering can lead to suboptimal model performance.		Document Transformations: Keep detailed records of all transformations applied to the data. Documentation ensures reproducibility and facilitates debugging, helping identify and correct preprocessing steps that may introduce errors. As an example for autonomous vehicle environmental sensing via lidar, LidarAugment can be employed to augment 3D objects for robust detection.		

Figure 4. Example ML FMEA rows from the Preprocess Data ML Pipeline step.

The Detailed Template

Figure 5 below is an excerpt of the ML FMEA Template containing the first two ML Pipeline Steps. The full ML FMEA Template is provided in the Appendix for reference. Additionally, the ML FMEA Template is also available for review and download from github at https://github.com/TallPaul67/MachineLearningFMEA.

The ML FMEA Template

Machine Learning Pipeline Step	Failure Mode Guide Words	Potential Failure Mode of the ML Model of Interest	Potential Effect on Higher Level System or Customer	Sev	Potential ML Pipeline Causes	Occ	Current ML Pipeline Best Practices & Process Controls	Det	ML RPN	Actions Recommended	Owner Target and Dates	Actions Taken	Sev	Occ	Det	Future ML RPN
Collect Data Requests	Incorrect or missing	Insufficient data requests			Incomplete or insufficient data requested for collection can lead to an incomplete or insufficient training dataset.		Clear Problem Definition and Goal Alignment: The data collection requests must align with the goals of the machine learning project. The clearer the definition of the problem, the more specific the data request will be. This reduces the chance of irrelevant or noisy data, which can lead to model errors. Clear goal alignment ensures that data relevant to the learning task is collected, reducing the risk of introducing biases or irrelevant information into the model.									
	Missing	Missing data collection requests			Incorrect prioritization of data collection requests can lead to biased or incorrect models.		Cross-Functional Input. Collaborate with domain experts to ensure that data requests reflect operational realities and domain-specific knowledge. This reduces the chance of missing important data dimensions or collecting incomplete datasets. By leveraging domain expertise, the data collected is more representative of real-world use cases, preventing models from making inaccurate assumptions due to incomplete or misunderstood data sources.									
	Too late	Late data collection requests			Late or lengthy data collection requests can cause models to be trained on outdated information (such as changing of seasons).		Constraints for Collection: Data collection requests should include constraints to ensure the collected data intent is met. For example, data may have to be geographically constrained, seasonally constrained, or constrained by other conditions such as time of day or precipitation.									
Collect Data	Incorrect or missing	Insufficient data collected			Incomplete or insufficient data collected can lead to an incomplete or insufficient training dataset.		Diverse and Representative Sampling: Ensure that data collection captures a wide variety of scenarios, especially edge cases and rare events. This ensures that the model learns from a comprehensive set of examples. By collecting data that covers the full spectrum of possible situations, models are less likely to fail when encountering novel or unexpected inputs.									
	Incorrect	Insufficient data collected			Manual data collected is incorrect or does not match the intent of the collection request.		Automated Data Collection with Monitoring: Automate data collection wherever possible to minimize human error and introduce robust monitoring to detect anomalies or data drift during collection. Automation reduces the likelihood of introducing errors from manual data handling, while continuous monitoring ensures data quality and integrity remain high, preventing issues downstream in the pipeline.									
	Too late	Late data collection			Late or lengthy collection of data can cause models to be trained on outdated or incorrect information (such as changing of seasons).		Constraints for Collection: Data collection should have clear constraints that may impact representative data collection. For example, data may have to be geographically constrained, seasonally constrained, or constrained by other conditions such as time of day or precipitation.									

Figure 5. Excerpt of the full ML FMEA Template containing the first two ML Pipeline steps: Collect Data Requests and Collect Data. Full version is available for reviewing and download on github at github.com/TallPaul67/ MachineLearningFMEA

Discussion

This paper argues that the ML FMEA method and the ML FMEA Template are novel contributions that address gaps in the applicable standards involving the development and deployment of ML in safety critical applications across industries including automotive, autonomous vehicles, defense and security, healthcare, pharmaceuticals, manufacturing and industrial robotics, warehouse distribution, and aerospace. The ML

FMEA approach is not designed to be a stand-alone tool, but rather key part of a comprehensive safety management system and safety case.

Some of the benefits of the ML FMEA method and the ML FMEA Template applied to the ML pipeline include:

1. The ML FMEA method can directly identify steps in the ML pipeline that could have the highest impact on risk and can therefore get higher levels of attention and diligence.
2. The ML FMEA method can aid in demonstrating agreement with ISO TR 5469 which recommends a PFMEA for the offline training process.
3. The ML FMEA method can aid in compliance with ISO PAS 8800 which requires the employment of an engineering rigor best practice.
4. The ML FMEA Template can be used as a “checklist” to ensure that no critical steps were missed in the ML pipeline development.
5. The ML FMEA Template can serve as a quick reference in the event of a performance limitation or failure in the field, providing the ability to quickly go back and review all the potential failures and mitigations to check if something in the ML pipeline was not sufficiently addressed and led to the performance degradation or failure, resulting in an update to the ML FMEA Template.
6. The ML FMEA Template provides a systematic way of identifying and documenting functional insufficiencies and mitigations associated with the ML pipeline.
7. The ML FMEA Template and ML FMEA method can serve as important evidence artifacts supporting a safety case claim that the ML pipeline was developed with the highest level of rigor.
8. Since the ML FMEA Template generally follows the Process FMEA flow, the approach is designed to be transparent and familiar to reviewers and experienced safety professionals.

The ML FMEA is a specific methodology that can address many if not all of the required and recommended activities described in the most widely followed standards governing the development and deployment of autonomous vehicles.

Future Work

The field of safety engineering analysis applied to machine learning is new with limited examples. Future work is needed to demonstrate a proof of concept that provides an example benefits of an applied ML FMEA approach.

In a subsequent publication the authors intend to describe specific examples and benefits of applying the ML FMEA method.

Additionally, it is worth noting that during the research and development of this approach a trend emerged. When looking across the full ML Pipeline, potential failure modes fit into three categories:

1. Data Issues: including bias, insufficient data or outdated data
2. Model Limitations: including over or under fitting
3. Deployment limitations: including integration failures or scalability

The authors plan to study this classification further and evaluate as potential ML failure mode guide words, in a manner similar to FMEA guide words. [26]

References

[1] ISO21448, "Road Vehicles – Safety of the Intended Functionality," Geneva, Switzerland, 2022.

[2] UL4600, Standards for Safety for the Evaluation of Autonomous Products, Underwriters Laboratory, 2019.

[3] ISO PAS 8800: Road Vehicles – Safety and Artificial Intelligence, Geneva, Switzerland: International Organization for Standardization, Under development.

[4] ISO TR 5469: Artificial intelligence — Functional safety and AI systems, Geneva, Switzerland: International Standards Organization, 2024.

[5] R. Salay and K. Czarnecki, "Using Machine Learning Safely in Automotive Software: An Assessment and Adaption of Software Process Requirements in ISO 26262," 2018. [Online]. Available: https://arxiv.org/abs/1808.01614.

[6] S. Studer, T. B. Bui, C. Dreshcer, A. Hanuschkin, L. Winkler, S. Peters and K.-R. Müller, "Towards CRISP-ML(Q): A Machine Learning Process Model with Quality Assurance Methodology.," *Machine Learning and Knowledge Extraction,* vol. 3, no. 2, pp. 392-413, 2021.

[7] J. Faria, "Machine Learning Safety: An Overview," in *Safety-critical Systems Symposium*, 2018.

[8] S. Mohseni, H. Wang, C. Xiao, Z. Wang and J. Wadawa, "Taxonomy of Machine Learning Safety," [Online]. Available: https://arxiv.org/pdf/2106.04823.

[9] R. Salay, M. Angus and K. Czarnecki, "A Safety Analysis Method for Perceptual Components in Automated Driving,," in *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, Berlin, 2019.

[10] E. Murphy, "STPAI: Leveraging STPA to Safeguard the Development of an AI Chatbot," in *International System Safety Society*, Minneapolis, 2024.

[11] X. Huang, G. Jin and W. Ruan, Assurance of Machine Learning Lifecycle, Springer, 2012.

[12] J. Rellermeyer, P. Heck and Y. Xie, "Systematic Mapping Study on the Machine Learning Lifecycle," in *IEEE/ACM 1st Workshop on AI Engineering* , 2021.

[13] E. Zeydan, S. S. Arslan and M. Liyanage, "Managing Distributed Machine Learning Lifecycle for Healthcare Data in the Cloud," in *IEEE Access*, 2024.

[14] AIAG and VDA FMEA Handbook, Automotive Industry Action Group and Verband der Automobilindustrie, 2022.

[15] Potential Failure Mode and Effects Analysis Including Design and Process FMEA, SAE Standard J1739: SAE International, Revision January 2021.

[16] "Standard for Performing a Failure Mode and Effects Analysis and Establishing a Critical Items List," NASA Goddard Space Flight Center, [Online]. Available: https://rsdo.gsfc.nasa.gov/documents/rapid-iii-documents/mar-reference/gsfc-fap-322-208-fmea-draft.pdf.

[17] "Guidance for Performing Failure Mode and Effects Analysis with Performance Improvement Projects," Food and Drug Administration (FDA), [Online]. Available: https://www.cms.gov/medicare/provider-enrollment-and-certification/qapi/downloads/guidanceforfmea.pdf.

[18] "NCPS Patient Safety Improvement Handbook," U.S. Department of Veterans Affairs, [Online]. Available: https://www.patientsafety.va.gov/professionals/publications/handbook.asp.

[19] R. Greer and M. Trivedi, "Towards Explainable, Safe Autonomous Driving with Language Embeddings for Novelty Identification and Active Learning: Framework and Experimental Analysis with Real-World Data Sets," in *arxiv.org/pdf/2402.07320*, 2024.

[20] E. Olson, "A passive solution to the sensor synchronization problem," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[21] H. Sommer, R. Khanna, I. Gilitschenski, Z. Taylor, R. Siegwart and J. Nieto, "A low-cost system for high-rate, high-accuracy temporal calibration for LIDARs and cameras," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),*, 2017.

[22] M. Quigley, B. Gerkey and W. Smart, Programming Robots with ROS, O'Reilly, 2015.

[23] D. Bogdoll, M. Nitsche and J. M. Zöllner, "Anomaly Detection in Autonomous Driving: A Survey," in *arxiv.org/pdf/2204.07974*, 2022.

[24] M. Bernhardt, D. Castro, R. Tanno, A. Schwaighofer, K. Tezcan, M. Monteiro, S. Bannur, M. Lungren, A. Nori, B. Glocker, J. Alvarez-Valle and O. Oktay, "Active label cleaning for improved dataset quality under resource constraints," in *Nature Communications*, 2022.

[25] Z. Leng, G. Li, C. Liu, E. Cubuk, P. Sun, T. He, D. Anguelov and M. Tan, "LidarAugment: Searching for Scalable 3D LiDAR Data Augmentations," in *ICRA*, 2023.

[26] ISO26262, "Road vehicles — Functional safety," in *International Standards Organization*, Geneva, Switzerland, 2018.

Acknowledgments

The authors wish to acknowledge the support of TORC Robotics and TUV Rheinland in the collaboration, development and publishing of this material.

.

Appendix

The ML FMEA Template for reference is provided. Note the pre-populated columns Machine Learning Pipeline Step, Potential Failure Mode of the ML Model of Interest, Potential ML Pipeline Causes, Current ML Pipeline Best Practice or Process Control. The detailed ML FMEA Template is also available for review and download from github at https://github.com/TallPaul67/MachineLearningFMEA.

The ML FMEA Template

Machine Learning Pipeline Step	Failure Mode Guide Words	Potential Failure Mode of the ML Model of Interest	Potential Effect on Higher Level System or Customer	Sev	Potential ML Pipeline Causes	Occ	Current ML Pipeline Best Practices & Process Controls	Det	ML RPN	Actions Recommended	Owner Target and Dates	Actions Taken	Sev	Occ	Det	Future ML RPN
Collect Data Requests	Incorrect or missing	Insufficient data requests			Incomplete or insufficient data requested for collection can lead to an incomplete or insufficient training dataset.		Clear Problem Definition and Goal Alignment: The data collection requests must align with the goals of the machine learning project. The clearer the definition of the problem, the more specific the data request will be. This reduces the chance of irrelevant or noisy data, which can lead to model errors. Clear goal alignment ensures that data relevant to the learning task is collected, reducing the risk of introducing biases or irrelevant information into the model.									
	Missing	Missing data collection requests			Incorrect prioritization of data collection requests can lead to biased or incorrect models.		Cross-Functional Input. Collaborate with domain experts to ensure that data requests reflect operational realities and domain-specific knowledge. This reduces the chance of missing important data dimensions or collecting incomplete datasets. By leveraging domain expertise, the data collected is more representative of real-world use cases, preventing models from making inaccurate assumptions due to incomplete or misunderstood data sources.									
	Too late	Late data collection requests			Late or lengthy data collection requests can cause models to be trained on outdated information (such as changing of seasons).		Constraints for Collection: Data collection requests should include constraints to ensure the collected data intent is met. For example, data may have to be geographically constrained, seasonally constrained, or constrained by other conditions such as time of day or precipitation.									
Collect Data	Incorrect or missing	Insufficient data collected			Incomplete or insufficient data collected can lead to an incomplete or insufficient training dataset.		Diverse and Representative Sampling: Ensure that data collection captures a wide variety of scenarios, especially edge cases and rare events. This ensures that the model learns from a comprehensive set of examples. By collecting data that covers the full spectrum of possible situations, models are less likely to fail when encountering novel or unexpected inputs.									
	Incorrect	Insufficient data collected			Manual data collected is incorrect or does not match the intent of the collection request.		Automated Data Collection with Monitoring: Automate data collection wherever possible to minimize human error and introduce robust monitoring to detect anomalies or data drift during collection. Automation reduces the likelihood of introducing errors from manual data handling, while continuous monitoring ensures data quality and integrity remain high, preventing issues downstream in the pipeline.									
	Too late	Late data collection			Late or lengthy collection of data can cause models to be trained on outdated or incorrect information (such as changing of seasons).		Constraints for Collection: Data collection should have clear constraints that may impact representative data collection. For example, data may have to be geographically constrained, seasonally constrained, or constrained by other conditions such as time of day or precipitation.									
Ingest Data	Incorrect or missing	Insufficient data ingestion			Incomplete or inaccurate data collection can lead to biased or incorrect models.		Automate the Ingestion Process. Utilize robust ETL (Extract, Transform, Load) tools and frameworks to automate data collection. Automation ensures consistent and error-free data collection, preventing gaps and inconsistencies that could compromise model performance. Automation minimizes human error, ensuring that data is ingested accurately and efficiently, thus reducing the risk of introducing incomplete or erroneous data.									
	Incorrect or missing	Insufficient data ingestion			Security breaches during data ingestion can compromise sensitive data, leading to ethical and legal issues.		Secure data transmission through encryption and ensure compliance with data protection regulations. Use access controls and audit logs to monitor data access. Protecting sensitive information during ingestion prevents unauthorized access and ensures the trustworthiness of the data used for training.									
	Too late	Late data ingestion			Delays in data ingestion can cause models to be trained on outdated information.		Ensure Data Quality: Implement initial data checks for integrity, accuracy, and completeness. Tools like Apache Griffin or Great Expectations can automatically detect and rectify anomalies. Ensuring data quality from the start reduces the risk of the model learning incorrect patterns, improving the reliability of the model's predictions. Examples: •Sensor Synchronization Issues: Misalignment in the timing of data collected from multi-modal sensors, such as cameras, RADAR and LIDAR, can result in inconsistencies. To mitigate this, time-stamping sensor data and utilizing real-time synchronization methods help align the data more accurately. Additionally, leveraging multi-modal redundancy—cross-referencing data from different sensors—can identify and correct temporal misalignments. •Dropped Messages: Data packet loss during transmission can create gaps in the data stream, potentially missing crucial information. Buffering and retry mechanisms are effective in ensuring that lost packets are re-transmitted. Additionally, multi-modal redundancy, using other sensors to verify missing data, can help bridge these gaps. •Data Format Inconsistencies: Data from different sensors often comes in various formats, resolutions, or coordinate systems (autonomous vehicle example: point clouds from LiDAR, images from cameras, and radar readings). Furthermore during the development process the respective firmware get updated, leading to increased capabilities over									
Validate Data	Incorrect or missing	Insufficient data validation			Invalid or corrupt data can lead to erroneous model training and predictions.		Schema Validation: Enforce data structure and type constraints through schema definitions. Tools like JSON Schema or Apache Avro can automate schema validation. Schema validation catches and corrects errors early, preventing structural inconsistencies that could lead to model misinterpretation and incorrect learning.									
	Incorrect or missing	Insufficient data validation			Undetected anomalies can introduce biases and reduce model performance.		Anomaly Detection: Implement automated checks to detect and handle anomalies such as outliers, missing values, and duplicate records. Identifying and mitigating anomalies ensures the model learns from clean, consistent data, reducing the risk of learning misleading patterns.									
	Incorrect or missing	Data validation not provided			Lack of validation can result in using incompatible or irrelevant data for training.		Consistent Monitoring: Continuously monitor data quality metrics and set up alerts for significant deviations. Early detection and rectification of data issues help maintain the consistency and reliability of the data, reducing the risk of model degradation. Examples: 1.Schema Validation: Enforcing data structure and type constraints through schema definitions ensures that the ingested data adheres to expected formats. Tools like JSON Schema or Apache Avro can automate schema validation, making it easier to detect and correct structural inconsistencies. Validating the schema in the datalake helps catch errors early, preventing issues like unexpected data structures that could lead to model misinterpretation and incorrect learning. 2.Anomaly Detection: Automated checks for anomalies such as outliers, missing values, and duplicate records can help ensure the integrity of the data. These can be achieved by applying a large scale offline virtual driver stack replaying the collected samples against the online stack and the driver in the loop. This allows to detect early misbehavior of the vehicle and important data samples for model refinement. 3.Consistency Issues: An autonomous driving stack involves multi-modal sensors and between real-world scene and individual subcomponent outputs can reveal for example in mapping depth information from LiDAR and stereo cameras, which can impact perception accuracy.									
Preprocess Data	Missing	Insufficient data preprocessing			Poor handling of missing values can introduce biases.		Standardize Data Cleaning Procedures: Establish and follow standardized procedures for handling common data issues like missing values and outliers. Standardizing data cleaning procedures ensures consistency and reliability in the data used for training, reducing the risk of introducing biases and errors. As a healthcare industry approach example, active label cleaning is a proven approach to clean noisy annotation labels.									

	Incorrect	Insufficient data preprocessing	Incorrect normalization or scaling can distort relationships in the data.	Automate Feature Engineering: Use automated feature engineering tools like Feature tools to systematically create and evaluate new features. Automation reduces the risk of overlooking critical data transformations, ensuring that the model captures all relevant information.
	Too little	Insufficient data preprocessing	Inadequate feature engineering can lead to suboptimal model performance.	Document Transformations: Keep detailed records of all transformations applied to the data. Documentation ensures reproducibility and facilitates debugging, helping identify and correct preprocessing steps that may introduce errors. As an example for autonomous vehicle environmental sensing via lidar, LidarAugment can be employed to augment 3D objects for robust detection.
Train Model	Incorrect	Insufficient model training	Overfitting or underfitting can occur if the model is not trained properly.	Use Cross-Validation: Employ techniques like k-fold cross-validation to ensure the model's performance is consistent across different data subsets. Cross-validation provides a more reliable estimate of the model's generalization ability, reducing the risk of overfitting.
	Incorrect	Insufficient model training	Incorrect algorithm selection can lead to poor model performance.	Hyperparameter Optimization: Automate hyperparameter tuning with tools like Grid Search, Random Search, or Bayesian Optimization. Proper tuning ensures the model performs optimally, preventing underfitting or overfitting.
	Too little	Insufficient model training	Poor parameter configuration can prevent the model from learning effectively.	Monitor Training Process: Track training metrics such as loss and accuracy in real-time. Tools like Tensor Board provide visual insights into the training process. Real-time monitoring allows for early detection of issues and timely intervention, ensuring the model trains correctly.
Tune Model	Too little	Insufficient model tuning	Suboptimal hyperparameter settings can degrade model performance.	Systematic Hyperparameter Tuning: Use systematic search methods or automated tools for hyperparameter tuning. Techniques like Bayesian Optimization or Hyperband systematically explore the hyperparameter space. Systematic tuning ensures optimal model performance and reduces the risk of suboptimal configurations that could degrade performance.
	Too much	Model is overtuned	Irrelevant or redundant features can increase model complexity and reduce accuracy.	Feature Selection: Evaluate the importance of features and remove irrelevant or redundant ones. Techniques like Recursive Feature Elimination (RFE) or LASSO can help. Feature selection reduces model complexity and improves generalization, preventing overfitting and enhancing accuracy.
	Too little	Insufficient model tuning	Lack of proper evaluation can result in a tuned model that does not generalize well.	Evaluate on Validation Set: Use a separate validation set to assess the performance of the tuned model. Proper evaluation prevents overfitting on the training data, ensuring the model's reliability in real-world applications.
Analyze Model	Too little	Insufficient model analysis	Over-reliance on a single metric can provide an incomplete picture of model performance.	Comprehensive Metrics: Use a range of evaluation metrics to cover different aspects of model performance. Multiple metrics provide a holistic understanding of model strengths and weaknesses, preventing optimization for a single aspect that might not capture all performance facets.
	Not provided	Missing model analysis	Failure to conduct thorough error analysis can leave critical issues unaddressed.	Error Analysis: Conduct a thorough analysis of the model's errors to identify limitations and areas for improvement. Understanding misclassification patterns helps implement targeted refinements, preventing repeated mistakes and improving overall model accuracy.
	Not provided	Missing model analysis	Lack of clear visualizations can make it difficult to interpret and act on model performance data.	Visualizations: Use visual tools like confusion matrices, ROC curves, and precision-recall curves to provide clear insights into the model's performance. Visualizations facilitate better interpretation and decision-making, helping identify and correct potential issues.
Deploy Model	Too little	Insufficient model deployment	Inadequate infrastructure can lead to performance bottlenecks.	Continuous Integration / Continuous Deployment (CI/CD). Implement CI/CD pipelines to automate the deployment process. CI/CD ensures smooth updates and minimizes manual errors, preventing deployment failures and maintaining model consistency.
	Incorrect	Insufficient model deployment	Poor monitoring can result in undetected performance degradation.	Monitoring and Logging. Continuously monitor the deployed model's performance and log predictions. Set up alerts and analyze logs to detect issues early. Early detection allows for prompt intervention, preventing prolonged periods of poor performance and maintaining the model's reliability.
	Too little	Insufficient model deployment	Lack of rollback mechanisms can make it difficult to address issues post-deployment.	Implement Software Rollback. A software rollback feature can revert the software back to a known release with known limitations that may have adequate mitigations.
	Too little	Insufficient model deployment	Integration can mask model failures and introduce integration failures	Performance Thresholds. Define and adhere to performance thresholds that the model must meet before deployment. Ensuring the model meets required standards prevents the release of suboptimal models, maintaining high performance and reliability.
	Too little	Insufficient model deployment	Scaling of the deployed model can introduce unaccounted for effects.	Scalability and Reliability. Use scalable and reliable infrastructure to host the model. Cloud services like AWS, GCP, or Azure provide robust options. Scalable infrastructure ensures the model can handle varying loads and maintain performance, preventing downtime and degraded performance.
	Incorrect	Insufficient model deployment	Model learns incorrectly through on-line learning and failure is not identified through validation	Do not enable on-line learning. On-line learning can lead to unvalidated content in the algorithm which can cause unwanted output. By not enabling on-line learning additional deployment and testing cycles may be necessary but it will mitigate deploying untested algorithms that may have undesired outputs.
	Too little	Insufficient model deployment	Lack of feedback can result in undetected performance drift.	Feedback Loops. Establish feedback loops to collect data on the model's predictions and outcomes. Continuous learning and improvement ensure that the model adapts to changing conditions, maintaining its accuracy and relevance. Monitor for Drift. Continuously monitor for data and concept drift to ensure early detection of performance degradation. Tools like Alibi Detect can identify changes in data dynamics. Monitoring for drift allows for timely retraining and updates to the model, preventing long-term degradation. Performance Indicators. Establish clear safety performance indicators that can capture ML issues. Safety performance indicators that have an established range can identify emerging issues with a model before they become a hazard.
	Incorrect	Insufficient model deployment	Ignoring user feedback can lead to models that do not meet user needs.	User Dashboard. Establish performance metrics from a user perspective. Establish an appropriate frequency to capture user feedback as well as the dashboard presentation view for key decision makers.
	Too late	Late model deployment	Delayed updates can cause the model to become obsolete.	Regular Updates. Schedule regular updates and retraining sessions for the model to incorporate new data and maintain performance. Regular updates ensure the model stays current with the latest data, preventing obsolescence and maintaining high performance.
Validate Model	Not provided	Missing model validation	Overfitting to the training data can result in poor performance on new data.	Holdout Validation. Use a holdout validation set or cross-validation to ensure the model's performance generalizes well to new data. This practice provides a realistic estimate of how the model will perform in real-world scenarios, reducing the risk of overfitting.
	Too little	Insufficient model validation	Validation on an unrepresentative test set can provide a false sense of model reliability.	Real-World Testing. Validate the model with real-world data, if possible. Real-world testing highlights discrepancies between the training environment and real-world scenarios, preventing unexpected failures post-deployment.
	Too little	Insufficient model validation	Validation to a dependent or derivative test set can provide a false sense of model reliability.	Independent data set. Ensure an independent data set is collected, created, and curated for validation
	Too little	Insufficient model validation	Ignoring real-world scenarios can lead to unexpected failures post-deployment.	Real-World Testing. Validate the model with real-world data, if possible. Real-world testing highlights discrepancies between the training environment and real-world scenarios, preventing unexpected failures post-deployment.

Analyze Model Feedback	Not provided	Insufficient model feedback analysis		Lack of feedback can result in undetected performance drift.		Feedback Loops. Establish feedback loops to collect data on the model's predictions and outcomes. Continuous learning and improvement ensure that the model adapts to changing conditions, maintaining its accuracy and relevance. Monitor for Drift. Continuously monitor for data and concept drift to ensure early detection of performance degradation. Tools like Alibi Detect can identify changes in data dynamics. Monitoring for drift allows for timely retraining and updates to the model, preventing long-term degradation. Performance indicators. Establish clear safety performance indicators that can capture ML issues. Safety performance indicators that have an established range can identify emerging issues with a model before they become a hazard.		
	Not provided	Insufficient model feedback analysis		Ignoring user feedback can lead to models that do not meet user needs.		User Dashboard. Establish performance metrics from a user perspective. Establish an appropriate frequency to capture user feedback as well as the dashboard presentation view for key decision makers.		
	Too late	Late model feedback analysis		Delayed updates can cause the model to become obsolete.		**Regular Updates.** Schedule regular updates and retraining sessions for the model to incorporate new data and maintain performance. Regular updates ensure the model stays current with the latest data, preventing obsolescence and maintaining high performance. Not sure where is the best place to insert this but a discussion on on-line vs off-line training may be relevant. In order to have a validated ML you can only support offline training so that may be a mitigation to specify.		