The ML FMEA Handbook

A Practitioner's Guide to Failure Mode and Effects Analysis for Safety-Critical Machine Learning

Version 1.0

A structured method for identifying and managing risk across the machine learning development pipeline.

Prepared By:

Omkar Salokhe

Zach Anderson

July 2026

Compiled from the open-source Machine Learning Failure Mode and Effects Analysis (ML FMEA) framework and the work of the ML FMEA Collaborative.

Synthesized from Schmitt et al., "Introducing the ML FMEA" (SAE World Congress, 2025) and Wadhvana et al., "The ML FMEA in Action" (SAE 2026-01-0079, preprint).

Authors

Akshay Chalana - *CEO and Co-Founder, Saphira AI*

Bodo Seifert - *Senior Automotive Functional Safety Engineer, TÜV Rheinland Group*

Chaitanya Shinde - *Staff Safety Engineer*

David Ward - *Global Head of Safety, HORIBA MIRA Ltd.*

Fahim Mannan - *Staff Machine Learning Engineer*

Jerry Lopez - *Senior Director Safety Assurance*

Justin Poh - *Senior Safety Engineer*

Krzysztof Pennar - *Principal Safety Engineer*

Majed Mohammed - *Technical Fellow Functional Safety, APTIV*

Mario Bijelic - *Machine Learning Researcher*

Michael Wagner - *Chief Safety Officer, Edge Case*

Neil Wadhvana - *Machine Learning Group Lead*

Omkar Salokhe - *Systems Safety Engineer, Reynolds & Moore*

Paul Schmitt, CSEP - *Director of Engineering Safety, Reynolds & Moore*

Simon Diemert - *VP Engineering, Critical Systems Labs*

Zach Anderson - *Senior Systems Safety Engineer, Reynolds & Moore*

Editors and Reviewers

Ashley Weis - *Senior Technical Writer, Reynolds & Moore*

Trystan Deck - *Engineering Manager, Reynolds & Moore*

Table of Contents

Preface

This handbook helps engineers, data scientists, and safety practitioners begin using the Machine Learning Failure Mode and Effects Analysis (ML FMEA) method or deepen their understanding if they are already familiar with it. Like the machine learning systems it addresses, the method is best learned through worked examples, so this handbook relies on two examples drawn from real engineering projects: an autonomous mobile robot performing perception-based speed and safety-envelope control, and a humanoid robot performing inspection and tote retrieval in a manufacturing environment. Appendices provide reference artifacts, a prepopulated template, rating tables, and a glossary that can be used directly or modified to fit a specific application.

The introduction explains why machine learning (ML) systems resist traditional software safety analysis and how ML FMEA adapts the established Process Failure Mode and Effects Analysis (PFMEA) to close that gap. It frames the central idea underlying the method: that machine learning algorithm development should be treated as a process rather than as a single model to be tested at the end.

Step Zero is not part of the original 11-step pipeline. The ML FMEA Collaborative added both after publishing 'The ML FMEA in Action,' having identified them, through field application of the method, as a step whose failure modes were otherwise going unexamined. Because of that significance, this handbook gives its own chapter, Chapter 3 for Step Zero, with worked examples showing how the gaps surface in practice. The original 11 steps are treated together in Chapter 4's step-by-step tutorial, which follows the same guide-word format established by PFMEA.

Chapter 3 covers Step Zero, the scoping activity that precedes any data collection, explaining how to define what the machine learning component is intended to do, how its output influences the encompassing system, and which system-level hazards its failure could cause. This foundation comes before the method itself for good reason: severity cannot be scored without a named system-level effect.

Chapter 4, “How to Do a Basic ML FMEA,” is an in-depth tutorial on performing an ML FMEA, walking through the full pipeline step by step. It is useful both for the beginner and for practitioners looking to sharpen their analyses, and its “Tips and Best Practices” boxes capture lessons learned from applying the method across multiple organizations.

The chapters that follow put the method into practice: scoping an analysis before any data is collected, constructing ground truth and feedback signals, tailoring severity, occurrence, and detection criteria to ML, aligning the resulting artifacts with international safety standards, and applying the method across the automotive, robotics, aerospace, and defense industries. A closing chapter reflects on what ML, and safety teams have reported after using the method together.

Above all, this handbook aims to make the ML FMEA process intuitive for the people performing it. The framework itself is open source, and a living template is maintained by the community so the method can adapt to new applications and industries as they emerge. Readers are encouraged to work through problems relevant to their own domain along the way.

Chapter 1: Introduction

Why machine learning breaks traditional safety analysis. The integration of machine learning into safety-critical systems continues to accelerate, spanning autonomous vehicles, humanoid robots, medical diagnostics, and aerospace. This adoption has outpaced the development of risk assessment methods tailored to the particular characteristics of ML. Unlike conventional software, in which behavior follows explicit, human-written logic, ML systems learn statistical patterns from data. Their behavior is therefore harder to predict, explain, and verify, and three properties in particular defeat the assumptions behind classical analysis.

- **Opacity**: ML models frequently operate as black boxes, and their decision-making process is difficult to inspect. That inspection is exactly what verification and validation (V&V) in safety-critical systems depends on.
- **Data Dependence:** Model performance is a function of data. When the operational environment differs from the training environment, performance can degrade silently and lead to unsafe decisions. Ensuring generalization to unseen scenarios is essential yet difficult.
- **Probabilistic, Nondeterministic Behavior:** Traditional verification assumes deterministic, well-understood behavior. The probabilistic nature of ML, combined with the vastness of possible inputs, conflicts with that assumption, and standardized V&V methods for ML do not yet exist.

Established safety frameworks such as IEC 61508, ISO 26262, ISO 21448 (Safety of the Intended Functionality, or SOTIF), ISO/TS 5083, and UL 4600 acknowledge the safety impact of ML components but offer limited prescriptive guidance for ML-specific failure modes, if any at all. Newer standards such as ISO/PAS 8800 and ISO/IEC TS 22440 provide a preliminary structure for artificial intelligence (AI) risk analysis and even point toward a process-level FMEA, but they describe what should be done without prescribing the specific tools to do it. The ML FMEA was developed to fill this gap.

Motivating Incidents

As with any safety method, the case for the ML FMEA is clearest in light of what goes wrong when learning-enabled components are deployed without systematic risk analysis. The following examples are representative of incidents that motivated the method.

1. In 2018, a pedestrian crossing a road at night while walking beside a bicycle was struck by an autonomous vehicle under human supervision.[1] Subsequent high-consequence events involving robotaxis and driver-assistance systems have occurred since.
2. In October 2023, a pedestrian thrown into the path of an autonomous vehicle by an adjacent, human-driven car was dragged approximately 20 feet after the vehicle's post-impact scene was misclassified as a side impact and a pullover maneuver was initiated.[2] The pedestrian was briefly detected by a side camera but was not tracked through the maneuver.

In each case, the failure did not originate in a single faulty line of code. It originated upstream, in how data was requested and collected, how ground truth or reward was constructed, how objectives were specified, or how the deployed model was monitored. Traditional analyses that examine the finished model in isolation tend to miss these contributors. The ML FMEA is designed to surface them.

The Core Idea: Machine Learning as a Process

The Process Failure Mode and Effects Analysis (PFMEA) is a proven, widely used method for identifying and mitigating failure modes in processes across the automotive, defense, medical device, pharmaceutical, manufacturing, and aerospace industries. A cross-functional team systematically reviews each step of a process, identifies potential failure modes, determines their causes and effects, and rates each by severity, occurrence, and detection. The product of those three ratings is a Risk Priority Number (RPN), which is used to prioritize mitigation effort.

The central insight of the ML FMEA is to treat machine learning development as a process, the ML pipeline, rather than as a single model to be evaluated only on performance metrics. By mapping PFMEA onto the stages of that pipeline, each step is examined for the ways it can fail, the causes of those failures, and the best practices that mitigate them. The approach retains the core principles of PFMEA described above while tailoring the severity, occurrence, and detection criteria to ML-specific risks such as label noise, dataset drift, incomplete requirements, and weak generalization.

Because the method follows the familiar PFMEA flow, it is transparent to experienced safety professionals and reviewers, while remaining accessible to ML developers who recognize each pipeline step from their daily work. This dual familiarity allows the ML FMEA to serve as a shared language across disciplines that have traditionally worked in isolation.

The PFMEA Framework

At its core, PFMEA is a structured method for identifying and mitigating potential failure modes in a defined process before those failures propagate to the end product or the end user. It decomposes a process into discrete, verb-plus-noun steps and, for each one, asks what can fail, why, what the effect would be, and how the failure would be caught, then rates the answer on severity, occurrence, and detection to prioritize corrective action. Appendix D walks through this mechanic in full, including the rating scales and the worksheet columns, for readers who want the complete reference before applying it to ML.

What matters for the argument here is portability. Because the framework's concepts, steps, failure modes, causes, effects, controls, ratings, and cross-functional review are process-agnostic, the same structure has been applied unchanged to manufacturing lines, medical procedures, pharmaceutical batch processes, aerospace assembly, and defense system development. This portability makes PFMEA a natural foundation for the ML FMEA: the ML development pipeline is a process, and PFMEA is the method one applies to a process.

What the Method Provides

ML FMEA is built around three foundational elements:

1. **A process-oriented ML pipeline** that maps the flow of data, models, and decisions from data ingestion to field deployment and feedback.
2. **A role-based ML FMEA template** that lets subject-matter experts from different disciplines identify, describe, and score potential failure modes in a familiar tabular format.
3. **An open-source toolkit**, including a prepopulated template and rating tables, maintained by the ML FMEA Collaborative so the method stays current and adaptable.

The method is not a standalone tool. It is intended as a key part of a comprehensive safety management system and safety case, producing auditable, repeatable, risk-aware evidence about how a learning-enabled system was developed.

Chapter 2: Benefits of the ML FMEA

The ML FMEA does more than assess risk: teams that adopt it also report real gains in how well ML and safety disciplines understand one another. Those benefits differ somewhat depending on which side of the table a practitioner sits, described in turn below.

Benefits for ML Development Teams

- **Unified Risk Repository:** Holding data and model failure modes in one database, connected to safety metrics, gives ML teams visibility they previously lacked and encourages proactive coordination among data engineers, ML scientists, and safety assessors.
- **Systematic Communication:** The structured template and scoring tables serve as a common language for explaining model risk to non-ML safety assessors and auditors, bridging development documentation and formal safety-case evidence.
- **Efficiency Gains:** Well-chosen mitigations frequently resolve more than one failure mode at a time, cutting the total number of fixes a team has to implement and track (Chapter 5 walks through a concrete example).

Practitioners also requested enhancements: dashboard visualizations highlighting “problem steps,” graph-based views of how failure modes propagate across data, model, and deployment stages, and hierarchical handling so that component-level FMEAs can nest within the broader pipeline model.

Benefits for Safety Engineering Teams

- **Role Clarity:** Safety engineers gravitate naturally toward severity assessment, drawing on system-level hazard knowledge, while ML developers lead occurrence and detection assessment, drawing on pipeline knowledge. This division improves both accuracy and efficiency.
- **Communication Bridge:** The tabular format and shared terminology let safety reviewers probe data assumptions, training procedures, and model limitations in a structured, non-adversarial way, fostering shared ownership of safety.
- **Conceptual Challenges:** The distinction between severity, occurrence, and detection can blur for probabilistic or adaptive behaviors; calibration examples and the refined rating tables are essential for consistent interpretation.

Chapter 3: Step Zero, Defining Scope

When the ML FMEA framework was applied across multiple projects, teams discovered that a preliminary activity was needed before Step 1 (Collect Data Requests).[11] In practice, three questions blocked progress before any data was collected: what problem the model was meant to solve, how its performance would be evaluated for safety, and which pipeline stages were in or out of scope. To resolve these recurring issues, the method introduced Step Zero: Define Scope. Introducing Step Zero fundamentally changed both the quality and efficiency of the analysis; teams entered later sessions with a shared understanding of context and consequence rather than debating terminology and scope.

Step Zero has five activities: clarifying the problem, documenting the pipeline, establishing a common vocabulary, clarifying what "safety" means, and analyzing hazards. The time investment is small, often a single facilitated session, but the clarity it provides benefits the entire project lifecycle.

Clarifying the Problem

Early workshops often assumed the model's purpose was already understood. In practice, vague or evolving problem statements led to inconsistent risk assessments and misaligned mitigations. Defining the problem first, what the model is intended to achieve and how its output influences the encompassing system under given operating conditions, proved essential. This resembles the "Understand the Objectives" phase of ISO/IEC TR 29119-11 (Software and Systems Engineering, Software Testing, Part 11: Guidelines on the Testing of AI-Based Systems) and aligns with governance practices in ISO/IEC 42001 and ISO/IEC 23053. Once objectives are explicit, teams can also judge whether ML is the right solution at all, or whether a deterministic, rules-based approach would be more transparent and verifiable.

Autonomous Vehicle Example: For the perception system on a developmental automated driving system, clarity about the operational design domain (ODD) is essential. If the model is assumed to detect pedestrians only within marked crosswalks, the case of a pedestrian crossing a road at night outside a designated crossing is treated as an out-of-scope condition rather than an in-scope scenario the system must handle. Failure modes such as classification instability under low-illumination conditions, loss of tracking history when an object is reclassified across categories, and absent path prediction for pedestrian trajectories that do not conform to designated crossings are then underrepresented or omitted from analysis. Formal clarification of the ODD, including the range of pedestrian behaviors the system is expected to accommodate, the illumination conditions under which detection must be reliable, and the persistence of tracked objects across classification changes, enables consistent mapping among the perception function, its failure modes, and the system-level hazards of collision with a vulnerable road user.

Humanoid Robot Example: For a bipedal robot performing balance and locomotion tasks, clarity about the operational state distribution under which the control policy is valid is essential. If the balance policy is assumed to operate on a freestanding robot with an unrestrained head, the case of the robot suspended by a tether during testing or demonstration is not a variation of the design condition but a departure from it. Failure modes such as misinterpretation of persistent external forces as a fall condition, divergent stabilization commands in the presence of unmodeled physical constraints, and execution of whole-body policies while the feet are not in contact with the ground are then absent from the analysis, because the states under which they arise are absent from the problem definition. Formal clarification of the operational state distribution, including the physical support conditions the policy assumes, the sensor readings consistent with those conditions, and the states under which the policy must

not be executed, grounds severity assessments in the full range of testing, demonstration, and deployment configurations the robot will encounter.

Documenting the Pipeline

Teams used the word "pipeline" to mean different things, sometimes only training, sometimes the full lifecycle including deployment and feedback. To create a shared reference, teams defined the specific steps of their workflow using concise verb-plus-noun phrasing (for example, *Collect Data Requests*, *Validate Data*, *Train Model*). This clarified both the value added at each step and the interfaces between roles, and it produced a stable artifact for audits and process improvement, consistent with documentation practices in ISO/PAS 8800 and ISO/IEC TR 5469.

Autonomous Vehicle Example: For the perception system on a developmental automated driving system, the pipeline extends from raw sensor capture through detection, classification, tracking, and prediction, into the trajectory planner that acts on the results. When the pipeline is documented only through the classification stage, the tracking and prediction steps become implicit, and the failure of a reclassification event to preserve tracking history, the mechanism by which a repeatedly reclassified pedestrian loses her predicted path, has no step to attach to in the analysis. Explicit documentation of every step-in verb-plus-noun form, from *Ingest Sensor Frames* through *Predict Object Trajectories*, ensures that inter-step behaviors such as tracking-history persistence across classification changes are named as first-class pipeline outputs, and their failure modes are analyzed accordingly.

Humanoid Robot Example: For a bipedal robot performing balance and locomotion tasks, the pipeline extends from state estimation through the control policy that produces joint commands and includes the assumptions the policy embeds about the physical support condition of the robot. When the pipeline is documented as running end to end from sensor input to joint actuation, the implicit precondition that the feet are in contact with the ground and the head is unrestrained is not represented as a pipeline element and cannot be examined for failure. Explicit documentation of every step-in verb-plus-noun form, from *Estimate Body State* through *Command Joint Torques*, and explicit naming of the preconditions each step assumes, ensures that violations of those preconditions during testing, demonstration, or deployment surface as analyzable pipeline states rather than remaining hidden inside the policy.

Establishing a Common Vocabulary

Terminology mismatches are a subtle but frequent source of confusion. Terms such as "validation," "testing," and "performance" carry different meanings for data scientists and safety engineers. Creating a shared glossary early, linking ML concepts to established safety language from ISO 26262, ISO 21448, and PFMEA, can be one of the most effective low-cost ways to improve communication and review quality. Defining success metrics and acceptance criteria during Step Zero makes later discussions of "good enough" objective: mitigations can be evaluated against explicit goals rather than intuition.

Clarifying "Safety"

Even the word "safety" carries different meanings by discipline, and reconciling those disparate meanings is central to the method.

In the ML research community, safety often refers to an algorithm's ability to satisfy constraints or remain within safe bounds during learning or operation, for example, Lyapunov stability, probabilistic constraint satisfaction, or bounded risk under distribution uncertainty. Here, safety is a property of the learning or control process: an

internal guarantee of bounded behavior under explicit mathematical assumptions, verified through proofs, convergence bounds, or empirical validation.

In functional safety engineering, safety is freedom from unreasonable risk to people or property, as codified in IEC 61508, ISO 26262, ISO 21448, ISO/TS 5083, and UL 4600. This view is system-level, emphasizing hazard identification, risk reduction, and evidence supporting assurance arguments.

Algorithmic safety alone is therefore insufficient: a stable, constraint-satisfying model can still contribute to an unsafe system if used without appropriate context, redundancy, or supervision. The two interpretations are bridged by deriving model-level safety properties from required system-level hazard mitigations.

Defining vocabulary early prevents failure modes that result from differing definitions. At the intersection of ML and functional safety specifically, terms such as validation, verification, reliability, and uncertainty can carry markedly different meanings across disciplines.

Analyzing Hazards

To estimate severity, the safety-relevant, system-level effect of each pipeline failure mode must be identified in terms of potential unsafe behavior of the encompassing system in which the ML component operates. Scoring severity likewise requires understanding the ML component's role in the broader system. Step Zero therefore includes a preliminary hazard analysis that explicitly links data-quality issues or model performance degradation to system-level consequences. The activity involves:

1. Identifying the safety-critical function the ML component performs within the encompassing system.
2. Determining how the ML component's outputs are used to make system-level decisions.
3. Performing a hazard analysis of the encompassing system (for example, FMEA or Systems-Theoretic Process Analysis, STPA) to determine how undesirable ML outputs could lead to system-level hazards (for example, a collision).

Autonomous Vehicle Example: For the perception system on a developmental automated driving system, the system-level hazards derived from vehicle-level hazard analysis include collision with a vulnerable road user, unintended lane departure, and unsafe adjacent-lane merging. Each hazard traces to the perception function whose failure could cause it: unstable classification of a pedestrian crossing outside a designated crossing traces to collision with a vulnerable road user; missed or offset subject-lane boundary detection traces to unintended lane departure; under-critical adjacent-lane boundary detection traces to unsafe adjacent-lane merging. This mapping allows severity in the ML FMEA to be scored against a named system-level effect rather than against the model error alone, making the resulting ratings defensible in an assessment.

Humanoid Robot Example: For a bipedal robot performing balance and locomotion tasks, the system-level hazards derived from system-level hazard analysis include unintended robot-human contact, unsafe contact forces during a fall or recovery, and damage to the operating environment. Each hazard traces to the ML function whose failure could cause it: false-negative detection of a partially occluded human traces to unintended robot-human contact; divergent stabilization commands under unmodeled physical constraints trace to unsafe contact forces; execution of a whole-body policy while the feet are not in contact with the ground traces to environmental damage from uncontrolled limb motion. Tracing each hazard back to its originating function keeps severity scoring anchored to a system-level consequence rather than to the policy error in isolation, and that anchoring is what an assessor needs to defend the resulting ratings.

Reflections and Recommendations

Explicitly linking ML process failures, incomplete data, labeling errors, model drift, to potential unsafe system behaviors lets analysts assign severity rankings with greater consistency and defensibility. It also strengthens traceability between process-level failures and system-level hazards, allowing the ML FMEA to serve as supporting evidence in broader safety cases and hazard logs. Organizationally, Step Zero has proven to be a unifying step: it bridges disciplines, reduces rework, and improves audit readiness. Teams that treat it as the foundation for rigorous safety analysis, rather than as an administrative prelude to skip through, tend to get the most out of it.

Tips for an Effective Step Zero

1. Conduct Step Zero as a single facilitated, cross-disciplinary session before any risk scoring begins; align on objectives, assumptions, and acceptance criteria up front.
2. State the model's responsibilities and system boundaries explicitly. Most misattributed or overlooked hazards trace back to an unstated scope.
3. Phrase every pipeline step as verb plus noun and define what each step does and does not cover, so failure-mode ownership is unambiguous.
4. Build a shared glossary that maps ML terms to ISO 26262, ISO 21448, and PFMEA language; revisit it whenever a term causes confusion.
5. Anchor severity in the encompassing system: for each ML function, name the specific system-level hazard(s) its failure could cause.

Chapter 4: How to Do a Basic ML FMEA

This chapter is a step-by-step tutorial for performing an ML FMEA. It introduces the 13-step ML pipeline, then walks through each step's characteristic failure modes, their causes, and the best-practice mitigations that control them. Two scoping refinements that experience proved essential, Step Zero (Define Scope) and Step Five (Construct Ground Truth or Feedback Signal), are introduced briefly here and treated in depth in Chapter 3 and Chapter 4, respectively.[11]

Method Overview

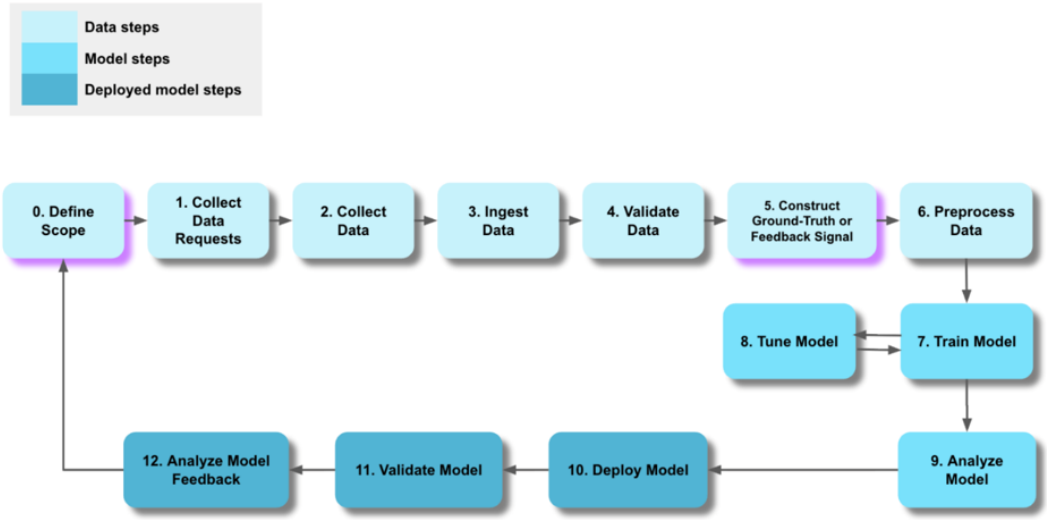
An ML FMEA proceeds in the same spirit as a PFMEA. A cross-functional team, typically including ML developers, data engineers, systems engineers, and safety practitioners, works through the pipeline one step at a time. For each step, the team asks: what can fail here, what causes that failure, what is the safety-relevant effect on the encompassing system, and what best practice or process control mitigates it? Each failure mode is rated for severity, occurrence, and detection, and the resulting Risk Priority Number focuses attention on the highest-risk areas of the pipeline.

The recommended sequence is:

1. Define scope and perform a preliminary hazard analysis of the encompassing system (Step Zero).
2. Walk the pipeline step by step, populating failure modes, causes, effects, and mitigations.
3. Rate severity, occurrence, detection, and compute RPNs.
4. Prioritize and assign mitigations, looking for single mitigations that resolve multiple failure modes.
5. Maintain the analysis as a living document, feeding it with field evidence after deployment.

The 13-Step ML Pipeline

To make the value added by each step explicit, the pipeline phrases each stage as a function using a concise verb-plus-noun form. The reference pipeline is shown below, grouped into data steps, model steps, and deployed model steps.



Step Number	Pipeline Step (Verb + Noun)	Group	Purpose
0	Define Scope	Scoping	Define the problem, system boundaries, terminology, and perform preliminary hazard analysis (Chapter 3).
1	Collect Data Requests	Data	Specify the data needed for training and evaluation; define problem scope, features, and sources.
2	Collect Data	Data	Gather raw data from sensors, databases, or other sources with the right diversity and volume.
3	Ingest Data	Data	Import and store data reliably from multiple sources, forming the pipeline's foundation.
4	Validate Data	Data	Verify the data meets quality standards and is accurate, complete, and error-free.
5	Construct Ground Truth / Feedback Signal	Data	Generate labels (supervised) or specify reward (RL); control annotation quality (Chapter 4).
6	Preprocess Data	Data	Clean and transform raw data: handle missing values, normalize, encode, engineer features.
7	Train Model	Model	Select algorithms, configure parameters, and fit the model to training data.
8	Tune Model	Model	Fine-tune hyperparameters and features to improve performance.
9	Analyze Model	Model	Evaluate performance with a range of metrics; conduct error analysis and visualization.
10	Deploy Model	Deployed	Integrate the trained model into production with monitoring, rollback, and thresholds.
11	Validate Model	Deployed	Confirm performance on unseen, independent data against required standards.
12	Analyze Model Feedback	Deployed	Collect field performance, monitor for drift, and feed insight back into the pipeline.

Table 5.1: *The reference ML pipeline. Step Zero and Step Five were added to the original 11-step pipeline based on lessons from applying the method.*

The numbering above reflects the updated 13-step pipeline (Steps 0–12) with Step Zero prepended and Step 5 inserted. Teams should adopt whatever step set matches their own workflow and document it explicitly.

The ML FMEA Template

The analysis is recorded in the ML FMEA template, which frames the pipeline within a familiar tabular format. Most columns are common with a standard PFMEA, so safety assessors can read the artifact without retraining. Four columns are modified to carry ML-specific language. The open-source template is an 18-column workbook; the full column sequence, left to right, is reproduced below so the analysis can be set up exactly.

Step Number	Step	Type	Purpose
1	Machine Learning Pipeline Step	Modified	The pipeline step under analysis (maps to PFMEA Process Step / Function).

Step Number	Step	Type	Purpose
2	Failure Mode Guide Words	Common	HAZOP-style prompt: Incorrect or missing, Missing, Incorrect, Too late, Too little, Too much, Not provided.
3	Potential Failure Mode of the ML Model of Interest	Modified	How that step can fail (maps to PFMEA Potential Failure Mode).
4	Potential Effect on Higher-Level System or Customer	Common	System-level consequence; basis for severity.
5	Sev	Common	Severity rating (see Chapter 6).
6	Potential ML Pipeline Causes	Modified	Why the failure mode occurs (maps to PFMEA Potential Causes).
7	Occ	Common	Occurrence rating (see Chapter 6).
8	Current ML Pipeline Best Practices & Process Controls	Modified	The mitigation in place (maps to PFMEA Current Controls).
9	Det	Common	Detection rating (see Chapter 6).
10	ML RPN	Common	Risk Priority Number = Severity × Occurrence × Detection.
11	Actions Recommended	Common	Corrective actions to reduce risk.
12	Owner, Target and Dates	Common	Accountability and schedule.
13	Actions Taken	Common	What was actually done.
14 → 17	Sev / Occ / Det (Rerated)	Common	Post-mitigation rerating of each dimension.
18	Future ML RPN	Common	Recomputed RPN after mitigation, evidencing risk reduction.

***Table 5.2**: The full 18-column ML FMEA template. The four "Modified" columns carry ML-specific content; the rest follow standard PFMEA practice unchanged.*

Two design choices deserve emphasis. First, the Failure Mode Guide Words column drives systematic discovery: the seven guide words in use, Incorrect or missing, Missing, Incorrect, Too late, Too little, Too much, and Not provided, are applied HAZOP-style to each step so that failure modes are surfaced deliberately rather than from memory. Second, the template carries two sets of severity, occurrence, and detection columns: an initial rating (columns 5 → 10) and a rerating after mitigation (columns 14 → 18). The drop from the initial ML RPN to the Future ML RPN is the quantified evidence of risk reduction that a safety case can cite.

The Risk Priority Number (RPN = Severity × Occurrence × Detection) prioritizes which rows demand corrective action; high RPNs are addressed first, and the rerating columns demonstrate that the action worked.

Walking the Pipeline: Failure Modes and Mitigations

The remainder of this chapter walks through the data, model, and deployed model steps, listing characteristic failure modes with their causes and best-practice mitigations. These are the prepopulated rows of the open-source template; teams should tailor and extend them for their own application.

Step 0: Defining Scope

Step Zero precedes data collection and grounds every subsequent step in a shared understanding of what the ML component is intended to do, how its output influences the encompassing system, and which system-level hazards its failure could cause. The key steps in this phase are

Failure Mode (Guide Word)	Cause	Best-Practice Mitigation
Unclear Problem Definition (Missing)	Model purpose and system role are undocumented.	Formal problem statement covering intended function, output interpretation, and ODD; sign-off before Step 1.
Divergent Vocabulary (Incorrect)	Terms such as safety, validation, and pipeline carry different meanings across ML and safety teams.	Joint project glossary adopting the ML FMEA conventions; preserved as a Step Zero artifact.
Missing Hazard Analysis (Not Provided)	System-level hazards caused by ML failure are not identified before scoring begins.	Preliminary hazard analysis (FMEA or STPA) of the encompassing system; per-function hazard mapping traced to severity scoring.

Step 1: Collect Data Requests

Data collection requests initiate the pipeline by specifying what data is needed. Their quality directly shapes model performance and safety.

Failure Mode (Guide Word)	Cause	Best-Practice Mitigation
Insufficient Data Requested (Missing)	Incomplete or insufficient data requested for collection.	Clear problem definition and goal alignment, so requests reflect the learning task and reduce noise or irrelevant data.
Biased Request Prioritization (Incorrect)	Incorrect prioritization of collection requests.	Cross-functional input: collaborate with domain experts, so requests reflect operational realities.
Outdated Request (Too Late)	Late or lengthy requests cause training on outdated information (for example, seasonal change).	Constraints for collection: geographic, seasonal, time-of-day, or weather constraints on the request.

Step 2: Collect Data

Data collection gathers raw data from identified sources. Quality, volume, and diversity strongly influence generalization.

Failure Mode (Guide Word)	Cause	Best-Practice Mitigation
Insufficient Data Collected	Incomplete or insufficient data collected.	Diverse and representative sampling capturing edge cases and rare events across the operational domain.
Collection Does Not Match Intent of the Collection Request	Manual data collected is incorrect or mismatched to the request.	Automated data collection with monitoring to minimize human error and detect anomalies or drift.
Outdated or Incorrect Collection	Late or lengthy collection causes stale or incorrect data.	Constraints for collection as in Step 1 (geography, season, time of day, precipitation).

Step 3: Ingest Data
Ingestion imports data from various sources into storage for use. In multi-sensor systems, synchronization, communication, and format consistency are common pain points.

Failure Mode (Guide Word)	Cause	Best-Practice Mitigation
Biased / Incorrect Ingestion	Incomplete or inaccurate data collection during ingestion.	Automate ingestion with robust extract, transform, load (ETL) tooling to ensure consistent, error-free import.
Security Breach	Sensitive data exposed during ingestion.	Maintain data security: encryption, access controls, audit logs, and regulatory compliance.
Outdated Ingestion	Delays cause training on outdated information.	Ensure data quality with integrity checks (for example, schema or quality tooling) to detect and rectify anomalies.

Multi-Sensor Notes: Sensor synchronization issues, dropped messages, and data format inconsistencies (point clouds, images, radar readings) are recurring ingestion hazards. Timestamping, real-time synchronization, buffering and retry, multimodal redundancy, and standardized formats with consistency checks mitigate these.

Step 4: Validate Data
Validation verifies that collected data meets quality standards before progressing. It catches structural inconsistencies, projection errors, and anomalies.

Failure Mode (Guide Word)	Cause	Best-Practice Mitigation
Invalid / Corrupt Data	Erroneous data leads to faulty training and prediction.	Schema validation: enforce structure and type constraints; automate with schema tooling.
Undetected Anomalies	Outliers, missing values, and duplicates introduce bias.	Anomaly detection via automated checks; offline digital-twin replay for comparison.
Incompatible / Irrelevant Data Used	Lack of validation admits unsuitable data.	Consistent monitoring of quality metrics with alerts; cross-reference multimodal data; calibration and dirty-sensor checks.

Step 5: Construct Ground Truth or Feedback Signal
This step, inserted between data validation and preprocessing, makes explicit the construction of labels (supervised learning) or reward (RL). Because many downstream failures originate here, it is outlined in full in Chapter 4.

Its six characteristic failure modes are annotation bias, label noise, misaligned objective functions, reward hacking, insufficient statistical analysis over time, and misaligned evaluation metrics.

Failure Mode (Guide Word)	Cause	Best-Practice Mitigation
Annotation Bias	Human annotators or labeling tools apply subjective or inconsistent criteria, resulting in biased or unreliable ground truth.	Document dataset motivation, composition, and intended use. Use inter-rater agreement metrics to verify labeling consistency. Apply bias-auditing techniques and structured labeling interfaces.
Label Noise	Errors introduced during automated labeling, data ingestion, or the integration of third-party datasets.	Apply automated label-quality estimation to flag mislabeled samples. Perform random spot checks and human audits. Maintain dataset lineage and provenance metadata. Use cross-modal verification before training.
Misaligned Objective Functions	The metrics optimized during training (for example, accuracy or cumulative reward) fail to represent system-level performance or safety intent.	Apply inverse reward design to capture designer intent under uncertainty. Define explicit safety-oriented key performance indicators (KPIs) aligned to AI hazard catalogs. Validate optimization metrics against SOTIF functional-insufficiency criteria.
Reward Hacking	In reinforcement learning systems, the reward function does not align with the true system objectives, leading to exploitation of loopholes.	Incorporate safety constraints and tamper-resistant reward design. Use safe-RL techniques that penalize unsafe transitions. Apply adversarial testing to identify reward-tampering pathways. Cross-verify learned behaviors against human-preference models.
Insufficient Statistical Analysis of the Dataset Over Time	Ground-truth dataset not routinely analyzed for distribution shifts across time, geography, or sensor configuration, masking key differences.	Use configuration-managed repositories with structured metadata ("Datasheets for Datasets"). [13] Integrate dataset-lineage tracking into continuous-integration pipelines. Establish versioned baselines with inter-annotator sign-off criteria.
Misaligned Evaluation Metrics	Evaluation benchmarks or test data fail to capture the ODD or safety-critical scenarios.	Use dataset-slice analysis and coverage-guided scenario generation (per ISO/PAS 8800). Apply out-of-distribution detection in evaluation pipelines. Tie post-deployment drift detection to severity-ranked ML FMEA updates.

Step 6: Preprocess Data

Preprocessing cleans and transforms raw data for training: handling missing values, normalizing, encoding, and feature engineering.

Failure Mode (Guide Word)	Cause	Best-Practice Mitigation
Insufficient Processing (Missing)	Poor handling of missing values introduces bias.	Standardize data-cleaning procedures for consistency and reliability.
Distorted Relationships (Incorrect)	Incorrect normalization or scaling distorts relationships.	Automate feature engineering to avoid overlooking critical transformations.
Suboptimal Features (Too Little)	Inadequate feature engineering degrades performance.	Document transformations for reproducibility and debugging.

Step 7: Train Model
Training fits the model to preprocessed data; tuning refines it. Overfitting and underfitting are the central hazards.

Failure Mode (Guide Word)	Cause	Best-Practice Mitigation
Over / Underfitting	Model not trained properly.	Use cross-validation for a reliable estimate of generalization.
Poor Algorithm Choice	Incorrect algorithm selection.	Hyperparameter optimization via systematic search (grid, random, Bayesian).
Ineffective Learning	Poor parameter configuration.	Monitor the training process; track loss and accuracy in real time.

Step 8: Tune Model
Model tuning fine-tunes the trained model to improve its performance.

Failure Mode (Guide Word)	Cause	Best-Practice Mitigation
Over-Tuned Model	Irrelevant or redundant features added.	Feature selection (for example, recursive feature elimination or LASSO).
Poor Generalization After Tuning	Lack of proper evaluation.	Evaluate on a separate validation set.

Step 9: Analyze Model
Model analysis evaluates performance and identifies areas for improvement.

Failure Mode (Guide Word)	Cause	Best-Practice Mitigation
Incomplete Performance Picture	Overreliance on a single metric.	Use comprehensive metrics covering multiple performance facets.
Critical Issues Unaddressed	No thorough error analysis.	Conduct error analysis of misclassification patterns for targeted refinement.
Hard-to-Interpret Results	Lack of clear visualizations.	Use confusion matrices, receiver operating characteristic (ROC) curves, and precision-recall curves.

Step 10: Deploy Model

Deployment integrates the model into production, where infrastructure, monitoring, and rollback determine reliability.

Failure Mode (Guide Word)	Cause	Best-Practice Mitigation
Performance Bottlenecks	Inadequate infrastructure.	Continuous integration / continuous deployment (CI/CD) pipelines to automate and de-risk deployment.
Undetected Degradation	Poor monitoring.	Monitoring and logging with alerts on the deployed model.
Hard to Recover	No rollback mechanism.	Implement software rollback to a known release with known limitations.
Masked / Integration Failures	Integration hides model failures.	Define and enforce performance thresholds before deployment.
Unsafe Online Learning	Model learns incorrectly online without validation.	Do not enable online learning; prefer validated offline retraining cycles.

Step 11: Validate Model
Validation confirms performance on independent, unseen data.

Failure Mode (Guide Word)	Cause	Best-Practice Mitigation
Poor Generalization	Overfitting training data.	Holdout validation or cross-validation.
False Sense of Reliability	Unrepresentative or derivative test set.	Real-world testing on an independent, curated dataset.

Step 12: Analyze Model Feedback
Feedback analysis closes the loop, detecting drift and degradation in the field.

Failure Mode (Guide Word)	Cause	Best-Practice Mitigation
Undetected Drift	Lack of feedback in the field.	Feedback loops, drift monitoring, and safety performance indicators (SPIs) with established ranges.
Obsolescence	Delayed updates.	Scheduled retraining and updates to incorporate new data.

Rating, Risk Priority Number (RPN), and Prioritization

Each failure-mode row is rated for severity (impact of the effect), occurrence (likelihood, reframed for ML as the maturity of requirements and quality controls), and detection (how and when the failure would be caught). The RPN is their product. High RPNs flag rows needing corrective action; the safety engineer then prescribes a strategy so that the system fails safe or fails operational. ML-tailored rating criteria are given in Chapter 6; they are essential because traditional PFMEA tables assume deterministic cause and effect that does not hold for ML.

Look for Cross-Cutting Mitigation: A recurring benefit reported by practitioners is that, after prioritizing by RPN, one mitigation often addresses several failure modes at once. For example, an automated data-quality gate introduced at Validate Data (Step 4) both catches invalid or corrupt records before they reach later stages and

reduces the label noise those records would otherwise introduce during Construct Ground Truth (Step 5), resolving two distinct failure modes across two pipeline steps with a single control.
Analyzing correlations among failure modes surfaces these high-leverage actions and reduces overall engineering effort while increasing assurance coverage.

Output and Traceability

A completed ML FMEA produces a traceable chain from pipeline step to failure mode, to cause, to system-level effect, to mitigation and owner. These outputs can be used in many ways, including:

- Identifying the pipeline steps with the highest impact on risk so they receive proportionate attention and diligence.
- Serving as a checklist to confirm no critical step was missed during development.
- Acting as a quick reference when a field performance limitation occurs, to review whether a pipeline gap contributed.
- Documenting functional insufficiencies and mitigations systematically.
- Providing evidence artifacts that support a safety-case claim that the ML pipeline was developed with rigor.

Because the artifact follows the PFMEA flow, it is transparent and familiar to reviewers, and that familiarity is what allows it to bridge ML development documentation and formal safety-case evidence.

Tips and Best Practices

1. Do not start at Step 1. Without Step Zero, a clear problem definition, documented pipeline, shared vocabulary, and preliminary hazard analysis, teams waste sessions debating scope and terminology, and severity ratings become indefensible.
2. Define the system-level effect, not only the model error. The effect of a pipeline failure must be expressed as a potential unsafe behavior of the encompassing system (for example, unintended lane departure, robot-human contact), or severity cannot be scored consistently.
3. Separate the two meanings of “safety.” Algorithmic safety (constraint satisfaction, stability guarantees) is different from functional safety (freedom from unreasonable risk at the system level). Bridge them by deriving model-level properties from system-level hazard mitigations.
4. Make ground truth and reward construction explicit (Step 5). Downstream failures such as bias amplification, reward hacking, and misread metrics often originate in supervision-signal construction that is otherwise hidden between data validation and preprocessing.
5. Use the right experts for the right rating. Safety engineers naturally lead severity (system-level hazard knowledge); ML developers lead occurrence and detection (pipeline knowledge). Pairing them improves both accuracy and efficiency.
6. Treat the ML FMEA as a living document. Feed it post-deployment evidence, drift, field failures, monitoring results, and update severity rankings and mitigations accordingly.

Chapter 5: Tailored Risk Assessment Criteria

Traditional PFMEA rating tables assume deterministic cause and effect and physically observable effects. ML pipelines involve probabilistic behaviors, data dependencies, and model-driven uncertainty, so direct application of those tables proved difficult. The criteria below retain PFMEA's familiar structure while introducing ML-appropriate definitions of severity, occurrence, and detection. Practitioners found the resulting tables intuitive and directly relevant to daily work, and the tables provided a shared risk vocabulary across technical and safety teams.

Severity Rating Criteria

The severity table distinguishes two perspectives, customer or vehicle safety impact and ML pipeline or process impact, enabling dual-perspective assessment. Refinements include explicit perception-failure criteria (for example, "perception blind spot," "low recall of pedestrians"), graduated safety-margin degradation levels, clear separation of primary safety functions from comfort features, and pipeline-specific disruption metrics.

Rank	Severity on Product (Customer / Vehicle Safety)	Severity on ML Process / Pipeline
10	Failure to meet safety or regulatory requirements. ML failure directly affects safe operation, causes missed detection or false actuation, or noncompliance without warning.	Pipeline error or model failure leads to safety-monitor bypass or catastrophic behavior; undetected until after incident.
9	Failure to meet safety, with warning. Hazard somewhat mitigated (monitor catches, but with degraded time-to-collision); regulatory risk with warning.	Issue forces partial stop of training, validation, or fleet deployment; may endanger operator with warning.
8	Loss of primary functions. Loss of ML-driven safety function (perception blind spot, planner failure); vehicle inoperable or needs takeover; safe state not guaranteed.	Major disruption: retraining halt, large data asset scrapped, or automated pipeline line shutdown.
7	Heavily degraded primary function. Function available but safety margins reduced (low recall, mis-planned maneuvers).	Significant disruption: retraining required, pipeline deviation, throughput drop, heavy rework.
6	Noticeable degradation (for example, 10% → 20% error on a critical slice). Hazards mitigated by redundancy.	Moderate disruption: subset of data reprocessed or relabeled; partial rollback.
5	Loss of secondary function (for example, Level 2 (L2) traffic-light misclassification). No direct safety impact.	100% of a secondary data product must be regenerated offline.
4	Degraded secondary function (rare-class mislabels). Customer notices degraded user experience, no safety risk.	Portion of pipeline or dataset rework; validation delays.
3	Annoyance or noise: false positives or non-safety misclassifications (frequent alerts, false stops).	Minor disruption: some relabeling or tuning needed.
2	Minor issue, not customer-visible; visible only to expert reviewers.	Slight inconvenience in pipeline or training; low-effort fix.
1	No effect on safety, performance, or compliance.	No effect on pipeline or product.

***Table 6.1**: Severity criteria, product versus ML process perspective.*

Occurrence Rating Criteria

In ML, a given rate of erroneous annotations may have unpredictable effects depending on data distribution and model architecture, so the usual assumption that cause probability maps directly to failure probability breaks down. The tailored occurrence criteria instead focus on the maturity of requirement definitions and quality controls, shifting discussion from probabilistic speculation to actionable quality management.

Level (Rank)	Criteria: Availability of Requirements and Checks
High (8 → 10)	Model performance requirements undefined and/or checks unavailable, or data-quality requirements undefined and/or checks unavailable.
Medium (4 → 7)	Requirements partially known and/or checks partially available, or requirements known and data-quality checks find x% of data failing (refer to severity table).
Low (1 → 3)	Model performance requirements known and checks available, or data-quality requirements known and checks available.

***Table 6.2**: Occurrence criteria based on maturity of requirements and quality controls.*

Detection Rating Criteria

Rather than likelihood-based detection ratings, the tailored criteria emphasize detection timing and automation across three pipeline segments: data pipeline detection (producer versus consumer), model development detection (before versus after training), and deployment detection (runtime monitoring and rollback). Three factors drive the rank: timing (before or after consumption or deployment), method (automated versus human-in-the-loop), and ownership (producer versus consumer).

Rank	Detection Timing (Containment)	Method and Ownership
10	After Consumption (Data / Model / Deployed Model)	High: human-in-the-loop check by data analyst or ML engineer.
9	After Consumption	Medium: automated quality check.
7 → 8	Before Consumption	High: data consumer performs human-in-the-loop check.
5 → 6	Before Consumption	Medium: data consumer uses automated quality check.
3 → 4	Before Consumption	Medium: data producer performs human-in-the-loop check.
1 → 2	Before Consumption	Low: data producer uses automated quality check.

***Table 6.3**: Detection criteria for data, model, and deployed model steps. Earlier detection by the producer scores lower (better) risk.*

Using the Tables Together

The three ratings multiply to the Risk Priority Number. Because the severity table is dual-perspective, a single failure mode can carry both a product-safety severity and a process-disruption severity; teams should be explicit about which they are scoring. The tables resonated with both ML developers and safety engineers, prompting cross-disciplinary discussion and guiding teams toward the right questions for each pipeline stage while maintaining continuity with traditional PFMEA conventions.

Chapter 6: Aligning ML FMEA with Safety Standards

Safety standards call for inductive, deductive, and exploratory analysis methods to generate systematic safety requirements, and this chapter shows where the ML FMEA fits alongside the latest machine learning and AI safety guidance. The ML FMEA is one such tool, approaching requirement generation from an ML process perspective; it is not a standalone method but a contributor within a broader ecosystem, and its artifacts map cleanly onto the expectations of five international standards.

ISO/PAS 8800, Safety and Artificial Intelligence

ISO/PAS 8800:2024 describes systematic safety analysis for AI elements across design, implementation, and operation, with specific focus on data.[4] It outlines techniques but offers limited guidance on specific tools. The ML FMEA addresses this gap by providing the recommended "safety analysis at the process level" (PFMEA in clauses such as 11.4.3.1, 12.3.5, 14.8.1, and 15.4), establishing traceability between ML development activities and safety requirements, documenting engineering rigor through systematic risk assessment, and creating auditable evidence of best-practice implementation.

UL 4600, Evaluation of Autonomous Products

UL 4600, Edition 3-2023, requires comprehensive safety arguments for autonomous systems.[5] ML FMEA artifacts support three argument patterns:

1. **Data Quality Arguments:** Rows addressing data collection, validation, and preprocessing provide evidence for Section 8.5.3.
2. **Robustness Arguments:** Failure modes for distribution shift and environmental variation support Section 8.5.2.2.
3. **V&V Process Arguments:** The complete ML FMEA demonstrates a systematic hazard analysis as required by Section 8.5.2.1.

The method also informs evidentiary processes supporting the Open Autonomy Safety Case (OASC), a non-normative safety case that supports assessment of UL 4600 conformance and the reporting of core autonomous-vehicle (AV) safety information.[12]

ISO 21448 (SOTIF)

The ML FMEA directly addresses the requirements of ISO 21448:2022 for identifying functional insufficiencies by systematically analyzing "specification of machine learning" insufficiencies, triggering conditions in the data used for training and testing, and measurement data-quality issues, and by supporting the offline training process analysis specified in Annex D.[6]

ISO/IEC TR 5469

ISO/IEC TR 5469:2024 is an industry-agnostic standard mapping AI lifecycle stages to applicable international standards.[7] It proposes a PFMEA approach to analyze and eliminate sources of bias and limitation in the offline training process, exactly the analysis the ML FMEA performs. Future standards such as ISO/IEC TS 22440 may apply this method more prescriptively.

MIL-STD-882E, System Safety (Aerospace and Defense)

FMEA is a recognized technique for MIL-STD-882E system safety programs, but practitioners rarely have the AI expertise to evaluate ML-relevant failure modes.[8] ML-specific failure modes trace directly into a MIL-STD-882E program: they seed the Preliminary Hazard List (Task 201), inform the Functional Hazard Analysis (Task 208), and carry forward into the System Hazard Analysis (Task 205). ML FMEA severity and occurrence ratings convert into the standard's severity categories and probability levels, and each hazard, causal factor, mitigation, V&V method, and risk-acceptance decision is recorded in the closed-loop Hazard Tracking System (Task 106). The method is referenced in the System Safety Program Plan (Task 102) and Hazard Management Plan (Task 103).

Taken together, compatibility with ISO/PAS 8800, ISO 21448 (SOTIF), UL 4600, ISO/IEC TR 5469, and MIL-STD-882E establishes a clear pathway for integrating ML FMEA results into formal safety cases.

Chapter 7: Applicability Across Industries

The ML FMEA was built with cross-industry use in mind, and this chapter surveys characteristic ML safety challenges across four domains along with how the method addresses each.

Automotive and Mobility

The number of autonomous vehicles and assistive driving features depending on ML is growing, spanning perception, prediction, planning, localization, equipment monitoring, and fleet routing, as well as micro-mobility platforms for delivery, sanitation, and security. These systems carry significant risk profiles, including potential fatalities. Two characteristic challenges:

1. **Diversity in Operating Domains:** Automotive operation ranges from rural winter roads to nighttime urban centers. Perception models must train on extremely diverse data. This is addressed by failure modes around Request Data (Step 1) and Collect Data (Step 2), for example, incorrect prioritization producing biased models, mitigated by cross-functional involvement in data requesting.
2. **Evolving Operating Domains:** Conditions change over time; even new reflective cycling apparel can impair camera perception. This is addressed at Ingest Data (Step 3) via the failure mode of ingestion delays producing out-of-date models, supported by Steps 1 and 2.

Aerospace and Defense

Chapter 7 details how ML FMEA failure modes map onto MIL-STD-882E's task structure; this section covers the domain's characteristic ML safety challenge. System safety practitioners on defense programs rarely have the AI expertise to evaluate ML failure modes, and ML engineers rarely know the MIL-STD-882E process. The ML FMEA bridges the two by giving both groups a shared artifact. In practice, the method's outputs are being deployed to multiple uncrewed ground-vehicle programs for the United States Army and used to inform test and evaluation centers on safety assessment reports and risk-acceptance decisions.[11]

Humanoid and Industrial Robotics

As robotics shifts from fenced, preprogrammed manipulators toward collaborative robots, mobile platforms, and general-purpose humanoids, learning-enabled systems introduce probabilistic behavior, opaque policies, and heavy data reliance. This presents three characteristic challenges:

1. **Robust Human Detection in Unstructured Environments:** Robots often operate within arm's reach of people, using vision or light detection and ranging (LiDAR) for dynamic speed and separation monitoring (ISO 10218, ISO/TS 15066). Detectors may fail under low light, occlusion, or unusual clothing and posture. This is addressed via Collect Data (Step 2) and Construct Ground Truth (Step 5). Other failure modes, such as insufficient worker diversity or annotation bias (labeling seated workers as background), are mitigated by targeted augmentation, synthetic data, and stricter labeling quality-assurance policies.
2. **Unsafe Emergent Behavior in RL Control Policies:** Policies learned in simulation and transferred via sim-to-real methods can produce brittle behavior, such as high-speed joint movements, unsafe contact forces, and reward hacking. These unsafe behaviors are mitigated by leveraging Step 5 (Feedback Signal) and Step 7 (Train Model), using test-time safety filters (joint-velocity caps, safety shields) and reward-validation workflows, with severity drawn from the robot's physical envelope.

3. **Integrating Learned Policies into Legacy Safety Envelopes:** Industrial robots run within safety-certified frameworks (programmable logic controllers, fieldbuses, configurable monitors). Introducing learned components requires preserving deterministic control paths and existing safety cases. Step Zero is critical here, ensuring the roles, interfaces, and boundaries of ML components are documented so teams can trace how learned behaviors affect legacy safe states or require new monitoring channels.

Across these domains, the ML FMEA method supports proactive risk mitigation, cross-disciplinary alignment, and compliance with functional safety standards such as ISO 13849 and IEC 62061.

Chapter 8: Why ML FMEA Is Open Source, and Who Benefits

From its first publication, the ML FMEA method, its template, and its toolkit were released as open source, and an ML FMEA Collaborative was formed to steward them. This was a deliberate choice, not an afterthought. Safety methods earn trust only when they are scrutinized, reused, and refined in the open, and the problem the ML FMEA addresses, the safety of learning-enabled systems, is moving quickly and is too consequential for any single organization to solve alone. This chapter explains the reasoning and what each kind of participant gains.

The Rationale

Three properties of the ML safety problem make an open, collaborative model the right fit.

1. **The Risk is Shared and Precompetitive:** A pedestrian struck by one company's autonomous vehicle damages public trust in the entire category. Safety is not where firms should compete; it is the floor everyone must clear. Pooling failure modes and mitigations raises that floor faster than siloed effort.
2. **The Standards Point to the Method But Do Not Prescribe the Tool:** ISO/PAS 8800, ISO/IEC TR 5469, ISO 21448 (SOTIF), and UL 4600 call for process-level analysis and engineering rigor without specifying how. An open, freely available method that satisfies those clauses gives the industry a concrete, auditable answer rather than incompatible proprietary ones.
3. **The Field is Young And Evolving:** Step Zero and Step Five, the two most significant refinements in this handbook, did not exist in the first ML FMEA publication. These steps emerged because multiple organizations applied the method, encountered the same gaps, and fed the lessons back. A closed method would have had to rediscover each gap independently; the open model surfaced and addressed them once, for everyone.

How Open Source Strengthens the Method Itself

Because the template and rating tables are public, they are stress-tested across automotive, robotics, aerospace, and defense applications simultaneously. Cross-industry validation exposes failure modes and ambiguities that a single domain would not encounter on its own. Transparency also makes the method auditable, since a safety assessor can inspect exactly how a rating table was constructed, itself a form of engineering rigor.

Open documentation practices reinforce this. The verb-plus-noun pipeline, the shared glossary, and the prepopulated rows are all public reference points, so a new team does not start from a blank page; the team starts from a peer-reviewed baseline and tailors it. This lowers the barrier to adoption, which widens the contributor base, which further improves the method.

Who Benefits, and How

The open model creates distinct, concrete value for every participant in the ecosystem.

Stakeholder	What They Gain
ML Developers	A free, ready-to-use template that frames safety in familiar pipeline terms, plus a shared vocabulary for talking to safety teams; no licensing, no locking in, no reinventing failure-mode catalogues.
Safety Engineers	A standard-aligned, auditable analysis method that extends FMEA discipline to ML, with transparent rating criteria they can defend in an assessment.

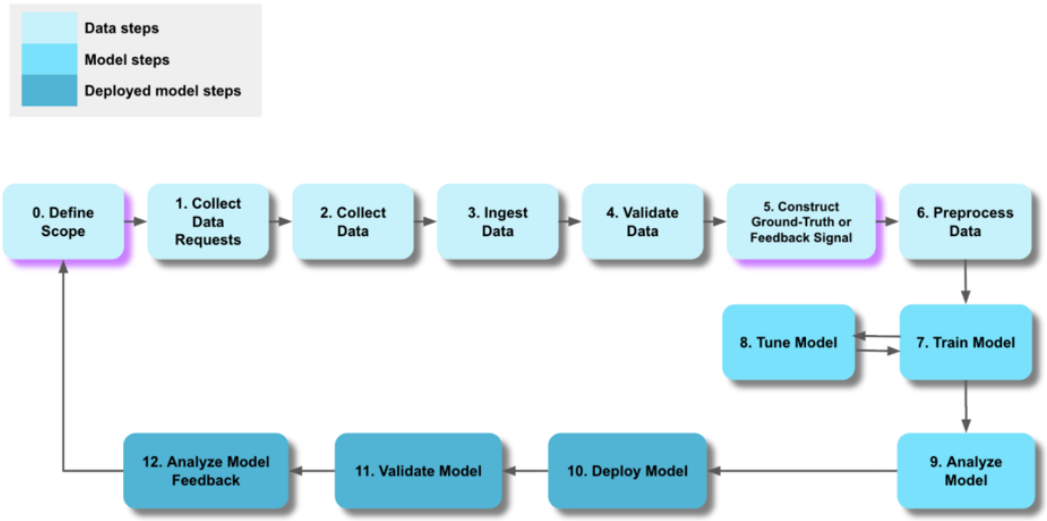
Stakeholder	What They Gain
Organizations	Lower cost of compliance with ISO/PAS 8800, SOTIF, UL 4600, and MIL-STD-882E; faster onboarding; and shared maintenance of a tool none of them must fund alone.
Regulators and Assessors	A common, inspectable reference method, making safety arguments across vendors easier to compare and evaluate.
Standards Bodies	Field-tested evidence of what process-level ML analysis looks like in practice, informing future standards such as ISO/IEC TS 22440.
The Public	A higher safety floor for learning-enabled systems across every industry that adopts the method.

The Collaborative Model in Practice

The ML FMEA Collaborative brings together developers, safety engineers, certification bodies, and researchers from across industry to share lessons, validate refinements, and keep the open template current. Knowledge gained in one domain propagates to the others, and improvements are reviewed in the open before they enter the reference artifact. Step Zero itself began this way: application teams repeatedly found themselves stalled before data collection could even begin, uncertain what problem the model was meant to solve or which pipeline stages were in scope, and that recurring gap became a fixture for the entire community once it was shared through the Collaborative.[11]

Appendix A: The Reference ML Pipeline at a Glance

Use this as a quick reference when scoping an analysis. Group each step into data, model, or deployed model phases and confirm that Step Zero (scoping) and Step Five (supervision signal) are explicitly covered.



□

Phase	Steps
Scoping	0. Define Scope
Data Steps	1. Collect Data Requests 2. Collect Data 3. Ingest Data 4. Validate Data 5. Construct Ground truth / Feedback Signal 6. Preprocess Data
Model Steps	7. Train Model 8. Tune Model 9. Analyze Model
Deployed Model Steps	10. Deploy Model 11. Validate Model 12. Analyze Model Feedback

Appendix B: The ML FMEA Template, Step by Step

This appendix reproduces the structure of the open-source ML FMEA template as published, the 13-step pipeline with its prepopulated guide words, failure modes, and example controls. (Step Zero and Step Five, from Chapter 3 and Chapter 4, are refinements layered onto this base; teams add them as additional blocks.) The severity, occurrence, detection, RPN, actions, owner, and post-mitigation rerating columns are completed per application during the analysis.

Row Count per Step

The published template carries a deliberately uneven distribution of prepopulated rows; Deploy Model has the most, reflecting how many distinct things can go wrong once a model meets the real world. Use this as a coverage checklist.

Pipeline Step	Prepopulated Rows	Guide Words Used
Collect Data Requests	3	• Incorrect or Missing • Missing • Too Late
Collect Data	3	• Incorrect or Missing • Incorrect • Too Late
Ingest Data	3	• Incorrect or Missing (×2) • Too Late
Validate Data	3	• Incorrect or Missing (×3)
Preprocess Data	3	• Missing • Incorrect • Too Little
Train Model	3	• Incorrect (×2) • Too Little
Tune Model	3	• Too Little (×2) • Too Much
Analyze Model	3	• Too Little • Not Provided (×2)
Deploy Model	9	• Too Little (×3) • Incorrect (×3) • Too Late (×2) • Not Provided (×1)
Validate Model	4	• Not Provided • Too Little (×3)
Analyze Model Feedback	3	• Not Provided (×2) • Too Late

***Table B.1**: Distribution of prepopulated rows across the 11 steps of the published template.*

Representative Rows

The following rows are drawn directly from the published template to illustrate the guide-word-to-failure-mode-to-cause-to-control chain at each phase.

Step	Guide Word	Failure Mode	Cause	Best-Practice Mitigation
Collect Data Requests	Incorrect or missing	Insufficient data requests	Incomplete data requested for collection	Clear problem definition and goal alignment
Collect Data	Incorrect	Insufficient data collected	Manual data does not match collection intent	Automated data collection with monitoring
Ingest Data	Incorrect or missing	Insufficient data ingestion	Security breach exposes sensitive data	Encryption, access controls, audit logs
Validate Data	Incorrect or missing	Insufficient data validation	Undetected anomalies introduce bias	Anomaly detection; offline digital-twin replay
Preprocess Data	Missing	Insufficient data preprocessing	Poor handling of missing values	Standardize data-cleaning procedures
Train Model	Incorrect	Insufficient model training	Overfitting or underfitting	Cross-validation (for example, k-fold)
Tune Model	Too much	Model is over-tuned	Irrelevant / redundant features added	Feature selection (recursive feature elimination, LASSO)
Analyze Model	Not provided	Missing model analysis	No thorough error analysis	Error analysis of misclassification patterns
Deploy Model	Too little	Insufficient model deployment	Lack of rollback mechanism	Implement software rollback
Deploy Model	Incorrect	Insufficient model deployment	Unsafe online learning, unvalidated	Do not enable online learning
Validate Model	Too little	Insufficient model validation	Derivative test set inflates reliability	Independent, curated validation dataset
Analyze Model Feedback	Not provided	Insufficient feedback analysis	Lack of feedback causes undetected drift	Feedback loops; drift monitoring; safety indicators

The full living template is maintained as an open-source workbook by the ML FMEA Collaborative. Teams should copy it and tailor the failure modes, causes, and controls to their own application rather than treating the prepopulated rows as exhaustive; the rows are a starting catalogue, not a ceiling.

Appendix C: Example System-Level Hazards

Hazards are often similar within an industry. The examples below, drawn from the two running applications, can seed a Step Zero hazard analysis. Recall that severity is scored against the encompassing system hazard, not the model error alone.

Application	ML Function	Failure	Linked System-Level Hazard
Autonomous Vehicle	Identify subject-lane boundaries.	Missed / offset lane boundary.	Unintentional lane departure.
Autonomous Vehicle	Identify adjacent-lane boundaries.	Under-critical adjacent-lane detection.	Collision with vehicle in adjacent lane
Humanoid Robot	Detect nearby humans / free space.	False-negative human detection (occlusion).	Unintended contact between robot and person.
Humanoid Robot	RL locomotion / recovery policy.	Unstable recovery under perturbation.	Unsafe joint accelerations / contact forces.

Appendix D: Understanding the PFMEA

The ML FMEA is built on the foundation of the Process Failure Mode and Effects Analysis (PFMEA), a method that has been in continuous use across the automotive, aerospace, medical device, defense, and manufacturing industries for decades. This appendix is a self-contained reference for practitioners who want to understand the PFMEA in its original form before applying it to machine learning development via the ML FMEA.

No prior knowledge of PFMEA is assumed. After reading this appendix, the reader will understand what a PFMEA is, why it exists, how to perform one step by step, how to interpret risk scores, and where it fits relative to other FMEA variants, including the ML FMEA.

1. What Is PFMEA?

The Process Failure Mode and Effects Analysis (PFMEA) is a structured, team-based method for systematically identifying the ways in which a process can fail, assessing the consequences of those failures, and implementing controls to prevent or detect them before they reach the customer or cause a safety incident.

The operative word is *process*. The PFMEA does not analyze a product's design (that is the role of the Design FMEA, or DFMEA). It analyzes the steps taken to manufacture, assemble, or deliver a product: the welding operation, the torquing sequence, the coating application, the software build pipeline. Any process that can produce a nonconforming output or create a hazard is a candidate for a PFMEA.

The method answers three fundamental questions for every potential failure in a process:

- **What could go wrong?** The failure mode.
- **How bad would it be?** The effect and its severity.
- **How likely is it, and would it be caught?** Occurrence and detection.

Multiplying the scores for these three dimensions produces the RPN, a composite index that guides where the team's corrective attention should go.

2. A Brief History and Standards Landscape

FMEA was first formalized by the United States military in 1949 (MIL-P-1629) as a reliability analysis technique for aerospace and defense systems.[9] The National Aeronautics and Space Administration (NASA) adopted and refined the method for the Apollo program in the 1960s.[10] The automotive industry adopted FMEA beginning in the 1970s, and it later became a formal requirement under the Automotive Industry Action Group (AIAG) standards.[9]

Today, PFMEA is defined and governed by five standards, each with slightly different terminology and emphasis:

Standard	Issuing Body	Scope and Key Contribution
AIAG-VDA FMEA Handbook (2019)	AIAG and Verband der Automobilindustrie (VDA)	The current joint automotive standard; harmonizes United States and German FMEA practice. Introduces the 7-step approach and

		the Action Priority (AP) system alongside RPN.
SAE J1739 (2021)	SAE International	North American automotive FMEA standard; defines DFMEA, PFMEA, and the Supplemental FMEA for Monitoring and System Response (FMEA-MSR).
IEC 60812 (2018)	International Electrotechnical Commission (IEC)	Industry-agnostic FMEA standard; applicable to any product, process, or service.
MIL-STD-1629A (1980)	United States Department of Defense	The foundational defense and aerospace standard; defines FMECA (FMEA plus Criticality Analysis).
IEC 61508 (2010)	IEC	Functional safety of electrical, electronic, and programmable electronic (E/E/PE) systems; Annex C defines the FMEDA variant for quantifying diagnostic coverage.

3. Key Terminology

The following terms appear throughout the PFMEA worksheet and are used consistently across all FMEA variants.

Term	Definition
Process Step / Function	A discrete activity in the process being analyzed, described as a verb plus noun (for example, “Torque Fastener,” “Apply Adhesive”). Each step is analyzed independently.
Failure Mode	The specific way in which a process step fails to perform its intended function. A failure mode is described in physical or functional terms, not as an effect or a cause (for example, “under-torqued fastener,” not “joint failure”).
Effect of Failure	The consequence of the failure mode on the next operation, on the final customer, or on the system. Effects are described from the customer’s or system’s perspective (for example, “joint separates under vibration load; vehicle unsafe”).
Severity (S)	A 1 → 10 rating of how serious the worst-case effect of the failure mode is. Severity is rated on the effect, not the cause or the likelihood. S = 9 → 10 indicates a safety or regulatory failure; S = 1 means no discernible effect.
Cause / Mechanism of Failure	The root cause or mechanism that produces the failure mode (for example, “torque wrench out of

Term	Definition
	calibration," "operator skipped step"). Each cause receives its own row in the PFMEA.
Occurrence (O)	A 1 → 10 rating of how likely the cause is to produce the failure mode given current controls. O = 10 means failure is near certain; O = 1 means failure is near impossible.
Current Process Controls	Prevention controls reduce the likelihood of the cause occurring. Detection controls identify the failure mode, or its effect, before it reaches the next process step or the customer.
Detection (D)	A 1 → 10 rating of how well current detection controls will identify the failure mode or its cause before it reaches the customer. D = 10 means no detection capability exists; D = 1 means detection is certain.
Risk Priority Number (RPN)	RPN = S × O × D. Ranges from 1 to 1,000. Used to prioritize corrective actions. A high-severity score (9 → 10) mandates action regardless of the resulting RPN.
Recommended Actions	Specific corrective or preventive actions that reduce S, O, or D. Actions should be specific, measurable, assignable, realistic, and time-bound.
Action Priority (AP)	An alternative to RPN introduced in the AIAG-VDA handbook. Rates items as high, medium, or low priority based on a combination of S, O, and D bands, mitigating known weaknesses of the multiplicative RPN approach.

4. How to Perform a PFMEA Step by Step

The following 11 steps describe how to complete a PFMEA from initiation to living-document maintenance. Steps 1 → 3 are preparation activities; Steps 4 → 8 constitute the core analysis; Steps 9 → 11 close the loop.

Step	Activity	Description
1	Form the Cross-Functional Team	Assemble engineers from process design, quality, manufacturing, safety, and maintenance. No single engineer should complete a PFMEA alone; the method's value comes from diverse expertise identifying failure modes that would be invisible to any one perspective.
2	Define the Scope	Agree on which process (or subprocess) is being analyzed, its boundaries, inputs, and outputs. Document the process using a flow diagram so every step is visible and

Step	Activity	Description
		agreed upon before analysis begins.
3	Break the Process into Steps	List each discrete process step or function. Phrase each step as verb plus noun (for example, "Weld Bracket," "Apply Adhesive"). This becomes the first column of the PFMEA worksheet.
4	Identify Failure Modes	For each process step, ask: "In what ways could this step fail to perform its intended function?" A failure mode is not an effect; it is the specific way the process goes wrong (for example, "under-tightened fastener," "contaminated surface").
5	Determine Effects of Each Failure	For each failure mode, determine its consequences at the next process step, at the final customer, and at the system level. Effects drive the severity (S) score.
6	Identify Causes	For each failure mode, identify the root causes or mechanisms that could produce it. Each cause receives its own row in the PFMEA worksheet. Causes drive the occurrence (O) score.
7	List Current Controls	Document what currently exists to prevent the cause from occurring (prevention controls) or to detect the failure mode or its effect before it reaches the next stage (detection controls). Controls drive the detection (D) score.
8	Score S, O, D, and Calculate RPN	Score each dimension on a 1 → 10 scale using the standard criteria tables. Multiply to obtain the Risk Priority Number (RPN = S × O × D). High RPN values, or a high-severity score regardless of RPN, signal the need for action.
9	Prioritize and Assign Corrective Actions	Focus first on any severity 9 → 10 items (potential safety or regulatory failures), then on the highest RPNs. Assign a specific

Step	Activity	Description
		person and a target completion date to each action.
10	Rescore After Actions Are Taken	Once actions are implemented and verified, rescore S, O, and D to calculate the revised RPN. Document what was actually done, which may differ from what was recommended.
11	Maintain as a Living Document	A PFMEA is not a one-time report. Update it when the process changes, when a field failure occurs, when new causes are identified, or at scheduled review intervals.

The Cross-Functional Team is Not Optional

Every authoritative PFMEA standard, AIAG-VDA, SAE J1739, and IEC 60812, requires a cross-functional team. This is not a formality. A manufacturing engineer sees machine capability; a quality engineer sees field-return data; a safety engineer sees regulatory exposure; a maintenance technician sees what breaks. No single person has all of these views. The ML FMEA extends this principle: ML engineers, data engineers, and safety engineers must all participate, because no single discipline sees the full failure landscape of an ML pipeline.

5. The PFMEA Worksheet, Column by Column

The PFMEA is typically documented in a structured worksheet. Below is an annotated example with one row filled in, followed by a description of each column. The worksheet below is condensed for illustration; full worksheets often include additional fields for revision history, team members, and part numbers.

Process Step / Function	Potential Failure Mode	Potential Effect(s) of Failure	S	Potential Cause(s) / Mechanism(s) of Failure	O	Current Process Controls	D	RPN	Recommended Action(s)	Responsibility & Target Date	Actions Taken	S	O	D	RPN (After)
Paint Application	Incorrect Film Thickness	Poor adhesion; corrosion within warranty.	7	Spray-gun pressure is out of calibration.	4	Pressure gauge checks every shift; statistical process control (SPC) chart on thickness.	3	84	Add automated inline thickness sensor	Engineering / Q3 2025	Sensor Installed; SPC Limits Tightened	7	4	2	56

Column Descriptions

- **Process Step / Function:** Identifies the step being analyzed. Sourced directly from the process flow diagram. Using a verb-plus-noun format (for example, “Apply Sealant”) keeps entries actionable and specific.
- **Potential Failure Mode:** How the step could fail to deliver its intended output. Multiple failure modes per step are normal and expected.
- **Potential Effect(s) of Failure:** The downstream consequence of the failure mode, at the next operation, at the final customer, or at the regulatory level.
- **Severity (S):** Rated 1 → 10. Reflects the worst-case effect listed. Assigned by the team, typically anchored by safety or quality engineers.
- **Potential Cause(s):** Root cause(s) or mechanism(s) that produce the failure mode. Each distinct cause receives its own row.

- **Occurrence (O):** Rated 1 → 10. Reflects how often the cause produces the failure mode given current prevention controls. Lower is better.
- **Current Process Controls:** Both prevention controls (reduce O) and detection controls (reduce D) are listed here, often labeled separately.
- **Detection (D):** Rated 1 → 10. Reflects how reliably the current detection controls identify the failure before it leaves the process. Lower is better.
- **RPN:** S × O × D. A prioritization tool, not a pass/fail threshold.
- **Recommended Actions:** Specific, assigned, time-bound improvements targeting S, O, or D. Actions targeting severity require a design change; those targeting occurrence require a process change; those targeting detection require an inspection or monitoring improvement.
- **Actions Taken and Rescore:** After implementation, document what was actually done and rescore to confirm risk reduction. The revised RPN closes the loop.

6. Further Reading

The following sources are the authoritative references for PFMEA and related FMEA methods.

- **AIAG-VDA FMEA Handbook (2019).** Automotive Industry Action Group and Verband der Automobilindustrie. The current joint standard for automotive DFMEA and PFMEA. Available from AIAG at aiag.org.
- **SAE J1739 (2021).** *Potential Failure Mode and Effects Analysis (FMEA) Including Design FMEA, Supplemental FMEA-MSR, and Process FMEA.* SAE International. Available at sae.org.
- **IEC 60812:2018.** *Analysis Techniques for System Reliability, Procedure for Failure Mode and Effects Analysis (FMEA).* International Electrotechnical Commission. Available at iec.ch.
- **MIL-STD-1629A (1980).** *Procedures for Performing a Failure Mode, Effects, and Criticality Analysis.* United States Department of Defense. The foundational defense and aerospace FMEA standard.
- **IEC 61508:2010, Annex C.** *Failure Modes, Effects and Diagnostic Analysis (FMEDA).* International Electrotechnical Commission. The normative reference for FMEDA applied to safety-related electronic systems.
- **Rausand, M., and Høyland, A. (2004).** *System Reliability Theory: Models, Statistical Methods, and Applications* (2nd ed.). Wiley. A comprehensive academic treatment of FMEA, FMECA, and related reliability methods.

The ML FMEA papers that build on this PFMEA foundation, and the open-source template itself, are listed in Appendix F.

Appendix E: Glossary

A shared glossary linking ML concepts to safety language is one of the highest-leverage Step Zero activities. The terms below recur throughout this handbook.

Term	Definition
ML FMEA	Machine Learning Failure Mode and Effects Analysis: a PFMEA-derived method that analyzes the ML pipeline for failure modes, causes, effects, and mitigations.
PFMEA	Process Failure Mode and Effects Analysis: the established automotive process-analysis method the ML FMEA adapts.
ML pipeline	The sequence of data, model, and deployed-model activities (Steps 0–12) treated as a process to be analyzed.
Step Zero	Preliminary scoping activity: problem definition, pipeline documentation, vocabulary, safety clarification, and hazard analysis.
Step Five	Construct Ground Truth (supervised) or Feedback Signal (RL): the supervision-signal construction step.
RPN	Risk Priority Number = Severity × Occurrence × Detection; used to prioritize mitigation effort.
Encompassing system	The larger system (vehicle, robot) in which the ML component operates; hazards and severity are defined at this level.
Functional safety	Freedom from unreasonable risk to people or property (IEC 61508, ISO 26262, ISO 21448, UL 4600).
Algorithmic safety	Constraint satisfaction or bounded behavior of the learning or control process (for example, Lyapunov stability); necessary but not sufficient for functional safety.
SOTIF	Safety of the Intended Functionality (ISO 21448): addresses hazards from functional insufficiencies and triggering conditions.
Reward hacking	An RL agent achieving high reward by exploiting loopholes in a misaligned reward function while behaving unsafely.
Drift	Change over time in data or concept distribution that degrades model performance if undetected.

Appendix F: Source Material and Further Reading

This handbook is a synthesis for instructional use. It draws on two primary sources from the ML FMEA Collaborative and references the standards and works those papers cite.

- **Schmitt, P., et al.** "Introducing the ML FMEA: A Safe Machine Learning Approach." SAE World Congress, 2025. Introduces the method, the 11-step pipeline, and the open-source template.
- **Wadhvana, N., et al.** "The ML FMEA in Action: Lessons from Applications of Machine Learning Safety." SAE 2026-01-0079 (preprint). Contributes Step Zero, Step Five, tailored rating criteria, standards alignment, and cross-industry perspectives.
- **Open-source template.** The living ML FMEA template is maintained on GitHub by the ML FMEA Collaborative and is the authoritative source for the latest prepopulated rows.

Referenced standards. IEC 61508; ISO 26262; ISO 21448 (SOTIF); ISO/TS 5083; ISO/PAS 8800; ISO/IEC TR 5469; ISO/IEC 42001; ISO/IEC 23053; ISO/IEC TR 29119-11; UL 4600; MIL-STD-882E; ISO 10218 and ISO/TS 15066 (robotics); ISO 13849 and IEC 62061 (machinery functional safety).

Prepared as an instructional handbook modeled on the structure of the *MIT STPA Handbook* (Leveson and Thomas, 2018). All ML FMEA technical content is attributable to the two source papers above and the open-source ML FMEA framework.

Appendix G: Worked Examples

Worked Example 1: AMR Perception, Driving Speed, and Safety Bubble

System context. An autonomous mobile robot (AMR) operates in a shared human-robot workspace and uses AI-based perception to detect obstacles (humans, forklifts, static obstructions, other AMRs) and to modulate its commanded speed and dynamic safety envelope accordingly. Faster speeds are permitted only when the perception system detects a clear envelope; slower speeds and a larger safety bubble are commanded as obstacles enter the envelope. Load state (empty, partial, full) modifies stopping distance and therefore the safety-bubble geometry required at any given speed.

Step 0: Define Scope

Failure Mode	Cause	Mitigation
Unclear Perception-Control Boundary (Missing)	Perception role and downstream control role are not documented separately.	Formal problem statement naming perception output as input to speed selection; sign-off before Step 1.
Missing Hazard Analysis (Not Provided)	System-level hazards from perception failure are not identified before scoring.	Preliminary hazard analysis (FMEA or STPA) mapping each perception function to collision and stopping-distance hazards.
Undocumented ODD (Missing)	Human presentations, obstacle types, load states, and lighting bands are not enumerated.	ODD document listing obstacle categories, human presentations, load states, and environmental bands; reviewed by ML, safety, and operations.

Step 1: Collect Data Requests

Failure Mode	Cause	Mitigation
Insufficient Coverage Specification (Missing)	Requests do not stipulate load-state or human-presentation variation.	Requests enumerate obstacle types, human presentations, and load states with coverage targets per ODD slice.
Biased Request Prioritization (Incorrect)	Requests focus on common obstacles at nominal load.	Cross-functional review with domain experts to prioritize safety-critical slices.
Outdated Request (Too Late)	Requests issued after collection is under way.	Requests front-loaded and phase-gated before Step 2.

Step 2: Collect Data

Failure Mode	Cause	Mitigation
Under-Collection of Rare Cases (Incorrect or Missing)	Passive collection oversamples common cases and under samples children, wheelchair users, and reflective forklifts.	Active-learning triggers for underrepresented slices; targeted supplementary campaigns; coverage matrices tracked against ODD.

Failure Mode	Cause	Mitigation
Collection Mismatches Intent (Incorrect)	Manual collection drifts from specification without monitoring.	Automated collection with sensor-verified logging; real-time anomaly and drift monitoring.
Late Collection (Too Late)	Extended campaigns cause data to no longer represent current site conditions.	Time-boxed windows; refresh cadence; version releases per ISO/PAS 8800.

Step 3: Ingest Data

Failure Mode	Cause	Mitigation
Time-Synchronization Loss (Incorrect or Missing)	Camera, LiDAR, and load-cell streams commingled without timestamp reconciliation.	End-to-end timestamp preservation with reconciliation checks; alerts on synchronization drift.
Security Breach at Ingestion (Incorrect or Missing)	Sensitive operational data exposed by weak controls.	Encryption in transit and at rest; role-based access; immutable audit logs.
Late Data Ingestion (Too Late)	Batch delays cause training on outdated snapshots.	Streaming or micro-batch pipelines with service-level agreement (SLA) monitoring; latency alerting.

Step 4: Validate Data

Failure Mode	Cause	Mitigation
Invalid or Corrupt Data (Incorrect or Missing)	Schema mismatches or truncated records introduce silent corruption.	Schema validation; per-feature range checks; sensor-firmware continuous integration.
Undetected Calibration Drift (Incorrect or Missing)	Dirty lens, LiDAR recalibration after firmware update, or load-cell drift passes validation.	Automated calibration checks against reference targets; digital-twin replay; validation stratified by sensor generation.
Incompatible Data Commingled (Incorrect or Missing)	Datasets from different sites or campaigns combined without validation.	Dataset lineage per “Datasheets for Datasets”; explicit ODD tagging per sample.

Step 5: Construct Ground Truth

Failure Mode	Cause	Mitigation
Annotation Bias in Obstacle Envelopes (Incorrect)	Guidelines do not standardize envelope size for occlusion, deformable objects, or human upper-body sway.	Explicit guidelines; inter-rater agreement (Cohen’s kappa); “Datasheets for Datasets”; slice audits.
Label Noise on Rare Presentations (Incorrect)	Automated labels or third-party datasets carry errors for wheelchair users, children, and seated workers.	Label-quality estimation (Confident Learning);[14] random spot checks; cross-modal verification.
Missing Coverage of Transition Scenarios (Missing)	Ground truth focuses on steady-state obstacle presence rather than	Coverage-guided scenario generation; explicit transition

Failure Mode	Cause	Mitigation
	obstacles entering or exiting the envelope.	annotation; out-of-distribution (OOD) detection per ISO/PAS 8800.

Step 6: Preprocess Data

Failure Mode	Cause	Mitigation
Load-State Signal Discarded (Missing)	Perception and load-cell channels normalized independently, losing correlation with required bubble geometry.	Joint preprocessing preserving cross-channel relationships; explicit load-state features.
Distorted Feature Relationships (Incorrect)	Incorrect normalization distorts sensor scale.	Automated feature engineering with defined transforms; validation of distribution shape before and after transform.
Suboptimal Feature Representation (Too Little)	Missing derived features for approach velocity or motion history.	Documented transformations for reproducibility; domain-specific augmentation.

Step 7: Train Model

Failure Mode	Cause	Mitigation
Overfitting to Nominal Cases (Incorrect)	Class imbalance rewards good performance on common scenes.	Class-weighted or focal loss; per-slice validation during training; augmentation for rare presentations.
Algorithm Mismatch (Incorrect)	Architecture chosen without empirical comparison for the perception task.	Baseline model portfolio as sanity check; nested cross-validation to avoid selection bias.
Ineffective Learning (Too Little)	Poor learning-rate or batch-size configuration.	Real-time monitoring of loss and gradient statistics; reproducibility harness.

Step 8: Tune Model

Failure Mode	Cause	Mitigation
Aggregate-Metric Tuning (Too Little)	Hyperparameter search scored against aggregate mean average precision (mAP), ignoring per-slice degradation.	Multi-objective tuning with worst-slice performance as a constraint; per-slice validation during tuning.
Over-Tuning to Spurious Features (Too Much)	Feature selection admits site-specific correlations.	Recursive feature elimination (RFE), LASSO, or mutual-information selection; permutation-importance validation.
Poor Generalization After Tuning (Too Little)	Tuning set reused as test set.	Strict three-way split; nested cross-validation where data is scarce.

Step 9: Analyze Model

Failure Mode	Cause	Mitigation
Aggregate-Only Reporting (Too Little)	Reports omit per-slice detection performance.	Comprehensive slice reporting per Model Cards; per-slice release criteria.
No Error Analysis (Not Provided)	Errors treated as an aggregate rate rather than analyzed for pattern.	Structured error analysis; clustering of high-loss examples; attribution methods (SHapley Additive exPlanations, or SHAP).[15]
Uninterpretable Results (Not Provided)	No visualizations or explainability output.	Confusion matrices, ROC and precision-recall curves; explainability applied to low-confidence detections.

Step 10: Deploy Model

Failure Mode	Cause	Mitigation
Missing Containment Logic (Too Little)	Perception output drives speed selection without an independent safety monitor.	Independent monitor per ISO 13849 or IEC 62061; conservative-default speed on low confidence; load-tied speed limits; emergency stop on safety-rated hardware.
No Rollback Mechanism (Too Little)	Deployment lacks a version-controlled fallback.	Blue-green or canary deployment; software rollback to a known release; immutable model artifacts.
Downstream Integration Masks Failures (Incorrect)	Post-processing hides degradation.	Per-deployment thresholds at the model boundary; raw output logging upstream of post-processing.

Step 11: Validate Model

Failure Mode	Cause	Mitigation
Test Set Drawn From Same Campaign (Too Little)	Test data shares training-set blind spots, such as site features and seasonal personal protective equipment (PPE).	Independent dataset collected separately; adversarial evaluation on known edge cases; site-forward splits.
False Sense of Reliability (Too Little)	Test distribution matches training rather than deployment ODD.	Real-world testing on deployment conditions; ODD-coverage audit.
Poor Performance On New Data (Too Little)	Overfitting undetected during development.	Holdout validation on independently collected data; time-forward and site-forward splits.

Step 12: Analyze Model Feedback

Failure Mode	Cause	Mitigation
Missing Fleet-Expansion Drift Monitoring (Not Provided)	Monitoring focuses on the initial region; new-site drift is undetected.	Data-drift monitoring per region (Kullback-Leibler, or KL, divergence; Population Stability Index, or PSI; Kolmogorov-Smirnov, or KS, test);

Failure Mode	Cause	Mitigation
		safety performance indicators (SPIs) with per-site ranges; expansion gates.
User Feedback Ignored (Not Provided)	Operator reports discarded; model optimizes for offline metrics.	User-perspective dashboard; structured feedback intake with triage workflow.
Model Becomes Obsolete (Too Late)	Seasonal or site change without retraining.	Scheduled retraining tied to drift; new-data ingestion; deprecation criteria.

Worked Example 2: Humanoid Inspection and Tote Retrieval on Rough Terrain

System context. A bipedal humanoid performs inspection and tote-retrieval tasks across manufacturing terrain (machined floor, gratings, cable protectors, spill trays, wet areas). An AI-based stability controller, a reinforcement-learning policy trained primarily in simulation with domain randomization, modulates step placement, torso pitch, and joint impedance. System-level hazards derived at Step 0 include unsafe joint accelerations, uncontrolled falls onto personnel or equipment, and dropped-tote impacts.

Step 0: Define Scope

Failure Mode	Cause	Mitigation
Unstated Policy Preconditions (Missing)	Feet-in-contact and head-unrestrained conditions are implicit rather than stated.	Formal problem statement naming the operational state distribution under which the policy is valid; sign-off before Step 1.
Missing Hazard Analysis (Not Provided)	System-level hazards from stability failure not identified before scoring.	Preliminary hazard analysis (FMEA or STPA) mapping each ML function to fall, contact, and dropped-tote hazards.
Undocumented Terrain ODD (Missing)	Terrain classes, transitions, and payload states not enumerated.	ODD document listing terrain classes, transition patterns, payload states, and support configurations.

Step 1: Collect Data Requests

Failure Mode	Cause	Mitigation
Under-Specified Terrain and Payload Variation (Missing)	Requests focus on nominal locomotion at nominal payload.	Requests enumerate terrain classes, transitions, payloads, and support configurations with coverage targets.
Biased Request Prioritization (Incorrect)	Simulation-cheap terrain over requested; hardware-only cases under requested.	Cross-functional review with controls, ML, safety, and operations.
Outdated Request (Too Late)	Requests issued after policy training is under way.	Requests front-loaded and phase-gated before Step 2.

Step 2: Collect Data

Failure Mode	Cause	Mitigation
Overreliance on Simulation (Incorrect or Missing)	Simulation is inexpensive and produces nominal-terrain data; hardware is costly and under collected.	Targeted hardware campaigns; systematic sim-to-real discrepancy metrics; active learning where simulation and hardware diverge.
Collection Mismatches Intent (Incorrect)	Manual hardware collection deviates from specification.	Automated collection with instrumented logging; real-time anomaly and drift monitoring.
Late Data Collection (Too Late)	Extended campaigns cause data to no longer represent current terrain.	Time-boxed windows; refresh cadence; version releases per ISO/PAS 8800.

Step 3: Ingest Data

Failure Mode	Cause	Mitigation
Multi-Rate Stream Misalignment (Incorrect or Missing)	Proprioceptive, exteroceptive, and inertial streams treated as synchronous at ingest.	Timestamp-preserving ingest with reconciliation; documented resampling policy; drift alerts.
Security Breach At Ingestion (Incorrect or Missing)	Sensitive operational data exposed by weak controls.	Encryption in transit and at rest; role-based access; immutable audit logs.
Late Data Ingestion (Too Late)	Batch delays cause training on stale samples.	Streaming or micro-batch pipelines with SLA monitoring; latency alerting.

Step 4: Validate Data

Failure Mode	Cause	Mitigation
Undetected Sim-to-Real Gap (Incorrect or Missing)	Validation confirms internal consistency but not simulation-versus-hardware agreement.	Automated simulation-to-hardware distribution comparison on matched terrain; digital-twin replay.
Undetected Anomalies (Incorrect or Missing)	Outliers and distribution shifts pass through validation.	Automated anomaly detection; cross-modal consistency checks.
Incompatible Data Commingled (Incorrect or Missing)	Data from different hardware units combined without stratification.	Dataset lineage per “Datasheets for Datasets”; explicit unit tagging.

Step 5: Construct Feedback Signal

Failure Mode	Cause	Mitigation
Misaligned Reward Specification (Incorrect)	Reward prioritizes efficiency and task success without penalizing divergent stabilization or precondition violation.	Reward shaping with safety constraints (joint bounds, near-fall penalty, precondition-violation termination); safe-RL techniques; adversarial evaluation.
Reward Hacking (Incorrect)	Reward function admits loopholes rewarding aggressive corrections.	Tamper-resistant reward design; causal-influence analysis; simulation validation cross-checked with human-preference models.

Failure Mode	Cause	Mitigation
Missing Precondition Tagging (Missing)	Training episodes not labeled with support configuration and terrain class.	Explicit episode metadata; slice-based analysis of reward against precondition state.

Step 6: Preprocess Data

Failure Mode	Cause	Mitigation
Terrain-Transition Signal Attenuated (Missing)	Observation down sampling smooths short-duration transitions below detection threshold.	Preprocessing preserves high-frequency content around transitions; explicit transition features.
Distorted Feature Relationships (Incorrect)	Incorrect normalization across sensor modalities.	Automated feature engineering; validation of distribution shape before and after transform.
Suboptimal Feature Representation (Too Little)	Missing derived features for terrain history and payload state.	Documented transformations for reproducibility; domain-specific augmentation.

Step 7: Train Model

Failure Mode	Cause	Mitigation
Overfitting to Simulation Dynamics (Incorrect)	Training exclusively in simulation with fixed physics parameters produces a policy tuned to simulator characteristics.	Domain randomization across mass, friction, actuator dynamics, and terrain; sim-to-real transfer; on-hardware fine-tuning.
Algorithm Mismatch (Incorrect)	RL algorithm choice not empirically compared against task structure.	Baseline algorithm portfolio; nested cross-validation to avoid selection bias.
Ineffective Learning (Too Little)	Poor exploration or reward-scaling configuration.	Real-time monitoring; reproducibility harness; safe-exploration constraints.

Step 8: Tune Model

Failure Mode	Cause	Mitigation
Aggregate-Reward Tuning (Too Little)	Hyperparameter search scored against mean training reward.	Multi-objective tuning with worst-terrain-class performance as a constraint; per-terrain-class validation.
Over-Tuning to Nominal Terrain (Too Much)	Configurations favoring machined-floor performance dominate.	Explicit constraint on rare-terrain performance; permutation-importance validation.
Poor Generalization After Tuning (Too Little)	Tuning against training-terrain distribution alone.	Strict three-way split by terrain class; hardware-based validation.

Step 9: Analyze Model

Failure Mode	Cause	Mitigation
Aggregate-Only Stability Metrics (Too Little)	Reports omit per-terrain-class stability signals.	Per-slice reporting; per-terrain release criteria for safety-critical classes.
No Error Analysis (Not Provided)	Near-fall events treated as an aggregate rate.	Structured analysis of near-fall clusters; explainability of policy actions on borderline scenarios.
Uninterpretable Results (Not Provided)	Policy behavior remains opaque to safety reviewers.	Explainability tooling applied to borderline decisions; Model Cards documenting known limitations.

Step 10: Deploy Model

Failure Mode	Cause	Mitigation
Missing Containment Logic (Too Little)	Policy commands reach the joints without an independent safety monitor.	Runtime motion bounding; safety monitor per ISO 13849 or IEC 62061; precondition monitoring with safe-state transition; emergency stop on safety-rated hardware.
No Rollback Mechanism (Too Little)	Deployment lacks a known-good policy fallback.	Version-controlled release; software rollback; immutable policy artifacts.
Downstream Integration Masks Failures (Incorrect)	Task-layer logic compensates for control-layer errors.	Per-deployment thresholds at the policy boundary; raw command logging upstream of post-processing.

Step 11: Validate Model

Failure Mode	Cause	Mitigation
Simulation-Only Validation (Too Little)	Validation captures none of the sim-to-real gap.	Independent validation on multiple hardware units across the terrain distribution.
False Sense of Reliability (Too Little)	Test distribution matches training rather than deployment.	Real-world evaluation on deployment conditions; adversarial evaluation on precondition-violation scenarios.
Single-Unit Validation (Too Little)	Validation on one hardware unit misses unit-to-unit variation.	Multi-unit evaluation; unit-forward splits reported alongside random splits.

Step 12: Analyze Model Feedback

Failure Mode	Cause	Mitigation
Missing Stability-Signal Monitoring (Not Provided)	Post-deployment focus on task-success metrics; underlying stability signals unmonitored.	SPIs on stability signals with per-unit ranges and alerting; drift monitoring on encountered terrain.
User Feedback Ignored (Not Provided)	Operator reports discarded.	Structured feedback intake with triage; user-perspective dashboard.

Failure Mode	Cause	Mitigation
Model Becomes Obsolete (Too Late)	Hardware wear or environmental change without retraining.	Scheduled retraining tied to drift; deprecation criteria defined and enforced.

Bibliography

[1] National Transportation Safety Board. “Collision Between Vehicle Controlled by Developmental Automated Driving System and Pedestrian, Tempe, Arizona, March 18, 2018.” Highway Accident Report NTSB/HAR-19/03. Available at: ntsb.gov/investigations/accidentreports/reports/har1903.pdf.

[2] Roy, A. “How GM’s Cruise Robotaxi Tech Failures Led It to Drag Pedestrian 20 Feet.” Reuters, January 26, 2024. Available at: reuters.com/business/autos-transportation/how-gms-cruise-robotaxi-tech-failures-led-it-drag-pedestrian-20-feet-2024-01-26/.

[3] Koopman, P. “Lessons from the Cruise Robotaxi Pedestrian Dragging Mishap.” Available at: users.ece.cmu.edu/~koopman/pubs/Koopman2024_CruiseMishap_IEEEReliabilityMagazine.pdf.

[4] International Organization for Standardization. *ISO/PAS 8800:2024, Road Vehicles: Safety and Artificial Intelligence.* 1st ed. Geneva: ISO, December 2024. Available at: iso.org/standard/83303.html.

[5] UL Standards & Engagement. *ANSI/UL 4600, Standard for Safety for the Evaluation of Autonomous Products.* 3rd ed. Northbrook, IL: UL, March 17, 2023. Available at: shopulstandards.com/ProductDetail.aspx?productid=UL4600.

[6] International Organization for Standardization. *ISO 21448:2022, Road Vehicles: Safety of the Intended Functionality.* 1st ed. Geneva: ISO, June 2022. Available at: iso.org/standard/77490.html.

[7] International Organization for Standardization and International Electrotechnical Commission. *ISO/IEC TR 5469:2024, Artificial Intelligence: Functional Safety and AI Systems.* 1st ed. Geneva: ISO/IEC, January 2024. Available at: iso.org/standard/81283.html.

[8] United States Department of Defense. *MIL-STD-882E, Department of Defense Standard Practice: System Safety*, with Change 1. Washington, DC: DoD, September 27, 2023 (originally issued May 11, 2012). Available at: cto.mil/wp-content/uploads/2025/07/MIL-STD-882E-w_CHANGE-1.pdf.

[9] American Society for Quality. “What Is FMEA? Failure Mode & Effects Analysis.” Reviewed November 2024. Available at: asq.org/quality-resources/fmea.

[10] National Aeronautics and Space Administration. “Procedure for Failure Mode, Effects, and Criticality Analysis (FMECA).” NASA Technical Memorandum NASA-TM-X-65227, August 1966. Available via the NASA Technical Reports Server at: ntrs.nasa.gov/citations/19700076494.

[11] Wadhvana, N., Seifert, B., Chalana, A., Poh, J., Mohammed, M., Wagner, M., Diemert, S., Mannan, F., Lopez, J., Pennar, K., Shinde, C., Ward, D., and Schmitt, P. “The ML FMEA in Action: Lessons from Applications of Machine Learning Safety.” SAE International, Paper 2026-01-0079 (preprint).

[12] Wagner, M., and Carlan, C. “The Open Autonomy Safety Case Framework.” *Safety-Critical Systems eJournal*, Vol. 3, No. 1 (2024). Available at: arxiv.org/abs/2404.05444.

[13] Gebru, T., Morgenstern, J., Vecchione, B., Vaughan, J. W., Wallach, H., Daumé III, H., and Crawford, K. "Datasheets for Datasets." *Communications of the ACM*, Vol. 64, No. 12 (2021), pp. 86–92. Available at: doi.org/10.1145/3458723.

[14] Northcutt, C. G., Jiang, L., and Chuang, I. L. "Confident Learning: Estimating Uncertainty in Dataset Labels." *Journal of Artificial Intelligence Research*, Vol. 70 (2021), pp. 1373–1411. Available at: doi.org/10.1613/jair.1.12125.

[15] Lundberg, S. M., and Lee, S.-I. "A Unified Approach to Interpreting Model Predictions." *Advances in Neural Information Processing Systems (NeurIPS) 30* (2017), pp. 4765–4774. Available at: proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html.