RESEARCH ARTICLE

Mapping Traditional Software Non-Functional Requirements into the Machine Learning Context

Timothy Elvira[1], Lynn Vonderhaar[1,*], Juan Couder[1], Tyler Thomas Procko[1], William Pate[1] and Omar Ochoa[1]

[1] Department of Electrical Engineering and Computer Science, Embry-Riddle Aeronautical University, Daytona Beach 32114, United States

Abstract

Non-Functional Requirements (NFRs) are quality-focused attributes of a system that impact functional components. There are 24 common NFR classes with some of the most used being performance, scalability, availability, reliability, and security. The implementations of these classes are ambiguous and describe the attributes on the behavior of software. Due to their nature, NFRs are typically realized through the specification and implementation of functional requirements. For traditional software systems, the NFR definitions are well known. However, in Machine Learning (ML), which has numerous applications across a multitude of domains, a current challenge is that traditional software non-functional class definitions do not consider stochasticity and black-box nature of ML, and thus, are not appropriately reflecting the ML context. This paper aims to address that challenge by mapping traditional software NFR definitions into ML-NFR definitions by redefining them with ML characteristics in consideration. This research separates the 24 common NFRs into system NFRs and ML NFRs to delineate which of the original 24 carries over to the ML context. Because of the proliferation of ML, requirement engineers need to be cognizant of the nuances of ML behavior as it pertains to NFRs. This paper presents a mapping of NFRs into the ML context including new definitions, as needed, for NFRs in ML systems.

Keywords: machine learning, non-functional requirements, requirements engineering, ML engineering, MLOps.

1 Introduction

Machine Learning (ML) systems have revolutionized various industries, from healthcare to finance, by leveraging algorithms that learn patterns and insights from data [1]. Currently, there is a growing domain and effort in exploring Software Engineering (SWE) principles and their effects on ML Engineering (MLE) and ML Operations (MLOps) [2]. Within this growing effort, there is a crucial yet overlooked aspect: Non-Functional Requirements (NFRs). Stemming from traditional methods of developing software systems, NFRs

Submitted: 25 March 2026
Accepted: 27 April 2026
Published: 09 May 2026

Vol. 2, **No.** 2, 2026.
10.62762/JSE.2026.908327

***Corresponding author:**
✉ Lynn Vonderhaar
vonderhl@my.erau.edu

Citation
Elvira, T., Vonderhaar, L., Couder, J., Procko, T. T., Pate, W., & Ochoa, O. (2026). Mapping Traditional Software Non-Functional Requirements into the Machine Learning Context. *ICCK Journal of Software Engineering*, 2(2), 102–120.

describe the quality attributes that define the desired system behavior and characteristics [3, 4]. Common NFRs include reliability, scalability, security, usability, maintainability, and performance. If developers ignore these non-functional qualities, the system might face detrimental consequences e.g., security vulnerabilities, inefficient resource utilization, and difficulties in system maintenance [5]. ML NFRs are difficult to define due to the engineer's lack of control over the model's learning and testing on the ML model [4]. Some have used quality models, e.g., Boehm's quality model, to approach the ML NFR problem [6, 7]. There are a series of quality models that hold certain NFRs as more important than others, creating new revisions or trends within software development [8]. Bridging these perspectives, accepted definitions of existing traditional software NFRs within these quality models can serve as a foundation in defining them in the context of ML.

The present paper examines the longstanding, traditional SWE definitions of NFRs and redefines them into the ML context. Ideally, identifying viable ML NFRs from SWE enables the transfer of existing knowledge and known NFRs into a novel space. This paper also aims to identify work within the ML space that implicitly use NFRs but do not explicitly identify them as such. By analyzing traditional NFRs and their translations to the ML context, this paper aims to shed light into potential ML NFR gaps for future research. These gaps may point to novel ML-specific NFRs that are not defined by any pre-existing definition. Additionally, traditional software NFRs have tradeoffs that are well-mapped, e.g., security is a negative tradeoff to performance [9]. Analyzing and redefining the existing NFRs might provide insight into potential tradeoffs in the ML NFR space. The scope of this paper is to incorporate the existing NFRs in traditional software engineering and provide examples of their existence in ML academia. Ideally, this work is an on-going effort to refine definitions. This paper answers the following research questions:

- RQ1: Which NFRs for traditional software systems map to ML-based systems?
- RQ2: What gaps are present in ML-based systems that require new NFRs and definitions?

The organization of this paper is as follows. Section 1 introduces NFRs and discusses how they can improve the development of ML systems, along with the current state-of-the-art and key challenges. Section 2 provides background information regarding NFRs and ML. Section 3 presents the mapping of traditional software NFRs into the ML context, including the challenges and necessary modifications to the traditional definitions. Section 4 discusses tradeoffs, the proposed taxonomy, and empirical literature mapping. Section 5 covers related work. Finally, Section 6 discusses potential future work, and Section 7 presents the authors' conclusions.

2 Background

This section covers NFRs and ML, both of which are critical in understanding the contents and efforts of this paper. Furthermore, these sections only briefly explain the concepts and further knowledge is beneficial to understanding the benefits of having multiple perspectives on NFRs for ML.

2.1 Non-Functional Requirements

Requirements are a key element for the design, structuring, and building of software systems. Requirement specifications are a large part of the software design lifecycle, most importantly Verification & Validation (V&V) [7]. A Software Requirement Specification (SRS) is a document containing both Functional Requirements (FRs) and NFRs, used as a form of contract with the customer [13] to ensure the development of the correct product. FRs correlate to constraints that specifically link to system behavior. Alternatively, NFRs are desired quality attributes of the system. They are more qualitative in nature and thus are more difficult to prove complete. Examples of NFRs include but are not limited to scalability, security, maintainability, reliability, usability, and interoperability [4]. FRs implement the system's NFRs [10]. Both types of requirements play a critical role in ensuring the product adheres to stakeholder needs and desires.

Within the software development lifecycle, one of the first phases is the requirement elicitation phase. In this phase, requirement engineers gather information from various stakeholders, and other artifacts to elicit raw requirements [11]. These raw requirements are analyzed through many iterations to create the SRS. Requirement specifications are not only used to ensure the customer's desired goals and needs are met; but, at their core, they are meant to help requirements engineers understand the context of the product or problem [12]. Requirements, encompassing both functional and non-functional, serve a critical role in V&V. Verification focuses on ensuring that the software product is correctly made while validation

involves ensuring that the software product meets the customer standards. During verification, NFRs are paramount, as they are essential in verifying non-functional parameters, e.g., measuring a system's response time [14]. Within validation, NFRs are equally essential, as they serve as a representation of the stakeholder's explicit expectations or needs of the system. NFRs assist software engineers in understanding the product to ultimately ensure the product meets technical specifications and stakeholder needs.

2.2 Machine Learning

ML is a subset of Artificial Intelligence (AI) in which a model learns patterns and probabilities from input datasets. ML differs from modern programming because the model learns iteratively from experience without explicit programming, a characteristic that makes some traditional SWE concepts, e.g., NFRs, difficult to map to the ML context [15]. The stages of an ML model include training, testing, and deployment. In training, the model iterates over the dataset with the goal of improving performance over time, with the exact procedure depending on the training paradigm [16]. The stochasticity of a model's starting weights, as well as the non-determinism of backpropagation during training makes the use of traditional NFRs difficult in the ML context because developers have less control over the model's behavior than in a traditional software system. Additionally, ML testing varies from traditional testing. ML Testing uses an assortment of metrics to understand model performance, for example, recall, precision, accuracy, and F1-score. However, the relative lack of code and the black-box nature of ML models makes verification of an ML model via requirements more difficult than traditional software systems. Finally, Deployment involves integrating a model into a system that uses the model's predictions or inferences for a downstream task, utilizing its learned knowledge to achieve system objectives [17]. The practices for deployment and maintenance of ML within systems are a key aspect of ML operations. ML Ops span a wide range of applications, from self-driving cars utilizing computer vision models to the use of sequence models for drug discovery [18]. Currently, ML is a rapidly growing field with the rise of Large Language Models (LLMs), which are models capable of generating language indistinguishable from human-like language [19]. Overall, ML is a paradigm where computers learn from data, utilizing a range of different learning types and stages from training to deployment.

2.3 Non-Functional Requirements for Machine Learning Models

There is some existing work in defining NFRs for ML-based systems [20–26]. Many authors focus solely on defining explainability as a new NFR that becomes important in ML-based systems [20–23]. Köhl et al. [20] identify explainability as an NFR for ML and discuss the definition of an explanation to better understand how to write explainability NFRs. Meanwhile Chazette and Schneider [21] link explainability and transparency by stating that transparency in ML directly leads to explainable ML because of the new understanding of the inner workings of the model. Sheh and Monteath [22] aim to define explainability and its methods as a means to justify meeting explainability requirements. Finally, Vogelsang et and Borg [23] note explainability and ethical behavior as new, ML-specific NFRs and indicate that, because of a model's behavior being built from data, the data requirements become important when developing ML-based systems.

Meanwhile other authors present full quality models for ML-based systems. Habibullah et al. [24, 25] have two papers towards this goal. The authors began the exploration by interviewing ML Subject Matter Experts (SMEs) to define their scope and determine how NFRs for ML differ from those for traditional software [24]. The authors then used these SME interviews to define NFRs that are relevant to ML and their relative importance in ML-based systems [25]. Meanwhile Bajraktari et al. [26] develop a separate quality model with 33 ML-specific NFRs based on a systematic literature review and mapping study. The authors provide a list of ML NFRs and offer guidance in using their quality model in practice. Though many of the NFRs between the two studies overlap, there are some differences, i.e., Bajraktari et al. [26] include "legal requirements" as an ML NFR. In contrast to these existing works, this paper bases its mapping on documented definitions, standards, and uses of traditional NFRs in the ML context to provide a new perspective on the issue.

3 Non-Functional Requirements Mapping

This work divides 24 traditional NFRs, from the Software Engineering Body of Knowledge (SWEBOK), ISO 25010, ISO 9126 and academic papers that define individual NFRs beyond the previously mentioned three, into two categories –those relevant to ML models and those relevant only to the surrounding system. This distinction is made by evaluating whether a

Table 1. Inclusion criteria for NFRs and the aggregated quality models.

NFR	Source	Inclusion	Rationale
Accessibility	ISO25010		
Accuracy	ISO9126		
Accountability	ISO25010		
Adaptability	ISO9126	X	C2
Affordability	ISO/IEC 15288	X	C2
Authenticity	ISO25010		
Availability	ISO25010	X	C3
Capacity	ISO25010		
Changeability	ISO9126		
Co-existence	ISO25010		
Compatibility	ISO25010		
Compliance	ISO9126		
Confidentiality	ISO25010		
Conformance	ISO9126		
Context Completeness	ISO25010		
Complexity/ Simplicity	McCall's Quality model	X	C3
Economic Risk Mitigation	ISO25010		
Efficiency	ISO9126		
Effectiveness	ISO25010		
Environmental Risk Mitigation	ISO25010		
Extensibility (Expandability)	McCall's Quality model		
Fault Tolerance	ISO9126		
Flexibility	ISO25010		
Functionality	ISO9126		
Functional Appropriateness	ISO25010		
Functional Completeness	ISO25010		
Functional Correctness	ISO25010		
Health and Safety Risk Mitigation	ISO25010		
Installability	ISO9126		
Interoperability	ISO9126	X	C3
Learnability	ISO9126		
Maintainability	ISO9126	X	C3
Maturity	ISO9126		
Modifiability	ISO25010	X	C3
Modularity	ISO25010	X	C3
Operability	ISO9126		
Performance	ISO25010	X	C1
Portability	ISO9126	X	C2
Recoverability	ISO9126	X	C2
Reliability	ISO9126	X	C3
Replaceability	ISO9126		
Resource Behavior	ISO9126		
Reusability	ISO25010	X	C2
Robustness	FURPS+	X	C1
Security	ISO9126	X	C1
Serviceability	FURPS+	X	C3
Stability	ISO9126	X	C3
Suitability	ISO9126		
Sustainability	Green Software	X	C4
Testability	ISO9126	X	C1
Time Behavior	ISO9126		
Trust	ISO25010		
Understandability	ISO9126		
Usability	ISO9126	X	C2
Usefulness	ISO25010		

Table 2. Prolific citations for NFRs in academia.

Title	NFR Definition Used	Citation Count
Representing NFRs and FRs: A goal oriented and use case driven approach	Serviceability, Supportability	67
Workshop on adaptable and adaptive software	Adaptability	12
Software Engineering Body of Knowledge (SWEBOK)	Affordability, Complexity	1289
Basic Concepts and taxonomy of dependable secure computing	Dependability	12
How architects see non-functional requirements: Beware of Modifiability	Modifiability	46
Mastering non-functional requirements: analysis, architecture, and assessment	Availability, Extensibility, Interoperability, Maintainability, Performance, Recovery, Reliability, Scalability, Security, Usability	16
Synergies and tradeoffs in Software reuse: a systematic study	Reusability	24
A systematic review of Software Robustness	Robustness	118
Prediction of Software Requirements stability based on complexity point measurement using multi-criteria fuzzy approach	Stability	15
Safety, Security, now sustainability: The nonfunctional requirement for the 21st century	Sustainability	184
A survey on Software Testability	Testability	69
Compatibility, Consistency, Interoperability	Compatibility	2
Product Modularity: Definitions and benefits	Modularity	804
A standards-based reference framework for system portability requirements	Portability	37

developer can affect the quality attribute specifically through model training and development. If not, then the NFR is a system-level NFR. It is important to note that the system-level NFRs are labeled as such because they are *only* system-level NFRs. Because the rest of the system is considered to be non-ML software, all listed NFRs can apply to the rest of the system. The system-only NFRs are listed so as to provide rationale for their classification.

Table 1 shows each individual source, the NFR used, and the citation count. Repetition between sources is ignored. This work identifies 16 ML Model NFRs related to the ML model and 6 (24 in total) System NFRs that are related to the system and do not map to ML-specific functionality. Furthermore, these NFRs relate to the coordination or integration of ML models within a system, directly influencing the NFR of the overall system. These 24 system NFRs might be able to fit just within the ML context but would need further analysis beyond existing literature and are therefore outside the scope of this work. These 24 NFRs are shown in Tables 2 and 3 for ML and system requirements, respectively. The following section overviews the SWE definitions, the new ML definitions, and the rationale for the ML definition change as it pertains to the ML context, if necessary.

The total list of NFRs is shown in Table 1. There is an inclusion criteria based on analyzing the current definition of the NFR and the degree it can be changed to fit into the ML context. An NFR must meet at

Table 3. Model NFRs and their new definitions.

NFR	SWE Definition	ML Challenges	ML Definition	Example
Adaptability	A system is adaptable based on how easy it is to change the behavior to meet the new needs [31].	Changing to meet new needs requires retraining of the model to some extent.	The degree to which a model accepts modifications in its architecture without degrading the performance.	Reducing model size to meet new computational restraints without losing performance.
Affordability	The system meets cost allocations in accordance with the contract [9].	Cost requirements for ML are difficult to decompose because modifications require new data collection model retraining.	Data compilation, model training, and model stability meet respective cost allocations.	Model maintenance and scalability can be expensive [32].
Complexity	A metric of the system's interdependency, modularity, and code readability [9].	The code structure of ML varies from traditional software. Complexity in ML is built during training, making it difficult to control [33]. A more complex model is not guaranteed to have a better performance than a simpler one [34].	A metric of the model's node interdependency and connectivity.	High connectivity and interdependency of nodes may cause the model to overfit to training data [34].
Maintainability	Requirements that describe the capability of the software product to be modified that may include correcting a defect or making an improvement or change in the software with a degree of ease [35].	Maintenance requires fine-tuning or retraining, resulting in a different model.	The ability to add more training data without degrading performance on existing classes and with the goal of improving performance.	Adding more training data of the same classes without tanking the performance [36].
Modifiability	Considers how the system can accommodate anticipated and unanticipated changes and is largely a measure of how changes can be made locally, with little ripple effect on the system at large [37].	Maintenance requires fine-tuning or retraining, resulting in a different model.	The ability to accommodate changes without compromising performance.	Adapting behavior to concept drift.
Performance	Responsiveness of the application to perform specific actions in a given time span. Measured in throughput or latency [35].	The time predictability of a model's response can vary [38].	The percentage of correct inferences.	Accuracy, precision, recall, F1-score [39].
Reliability	The ability of a system to garner trust given its demonstration of other quality attributes, including availability, integrity, safety, and maintainability [40].	Behavior in any scenario does not necessarily indicate a model's behavior in another scenario.	The ability of a system to garner trust given its demonstration of other quality attributes including availability, security, and performance.	Good performance metrics for a model may garner trust in its behavior.
Reusability	The ability of a system to be used for a variety of purposes different than its original one [41].	Models tend to be created for a specific purpose and if used for a different one, the performance will very likely decrease.	The ability of a model to be fine-tuned to a new application.	Transfer learning allows the hidden layers of a neural network to be reused [42].
Robustness	Ability of a system to maintain its regular behavior when it encounters errors during its execution and when receiving incorrect input [43].	Since ML is data-driven, there is no guarantee that a model can handle all possible scenarios, e.g., corner cases [38].	The ability of the model to maintain the same performance when it encounters unexpected transformations in the input data.	Handling natural noise in ML Data sets that could cause misclassification [44].
Modularity	Ability to handle an increase in the workload without impacting the performance, or the ability to quickly expand the architecture [35].	Larger/more capable models require more testing, which is already difficult.	The ability to add more training data with new classes without tanking a margin of performance on existing classes.	Adding new classes without decreasing the performance within a specified margin of existing classes [45].
Security	Requirements that concern preventing unauthorized access to the system, programs, and data [35].	N/A – This definition maps directly.	Requirements that concern the prevention of a threat actor attempting or succeeding in acquiring unauthorized access to a model and its components.	Artificial noise enables threat actors to influence a model's behavior [46, 47].
Serviceability/ Supportability	How easy it is to perform service on the component or the whole system when needed [10].	Maintenance requires fine-tuning or retraining, resulting in a different model.	How easy it is to retrain or fine-tune a model when needed.	The ease of retraining for concept drift.
Stability	A system that will not require changes over a period of time. The more stable, the fewer changes it requires [48].	This NFR depends on the data the system uses and depends on the performance of the model to determine when it must be retrained. The cost of retraining must also be considered. There must be a balance between performance and cost of retraining.	The ability of a model to keep up with current trends in the data i.e., the shelf life of a model.	Models that are used in more volatile environments will require frequent retraining while models in a more constant environment will not [19, 32].
Sustainability	Ability to meet the needs of the present without compromising the ability of future generations to meet their own needs [49].	N/A – This definition maps directly.	Ability of a model to not surpass the acceptable energy consumption.	Energy-efficient models [50].
Testability	Degree to which a system under test supports its testing in a given test context. If testability is high, the system will allow for more defects to be found during testing [51].	Testing methods developed for traditional software don't work for ML [52]. For instance, coverage testing is not feasible with millions of parameters and neurons. Additionally, corner cases prevent completeness of the test suite [52].	The amount of information that you can extract from a test case.	More complex models, like LLMs have more nodes, making testing take more time [53].

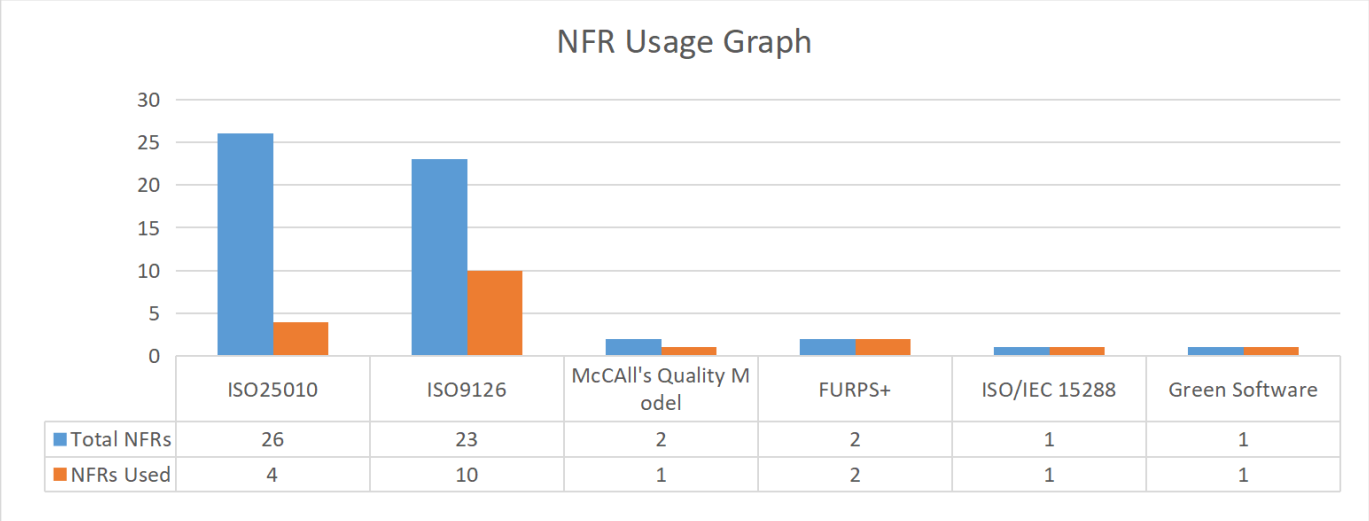


Figure 1. The usage of the ML Model NFRs compared to the total amount of analyzed NFRs, aggregated from 6 quality models.

least one inclusion criterion to be considered in this work. From these criteria, the NFRs are split into ML either ML Model NFRs or System NFRs as defined previously. Figure 1 illustrates the usage of the ML Model NFRs compared to the total amount of analyzed NFRs, aggregated from 6 quality models.

Inclusion criteria:

- Criteria 1 (C1): The NFR is currently in use in the ML academic space, i.e. security, performance, and robustness.
- Criteria 2 (C2): The NFR is analyzed for feasibility within the ML academic space using established definitions or similar words, i.e. sustainability and green ML.
- Criteria 3 (C3): The NFR is analyzed to be shifted into the ML context with reasonable similarities between traditional NFR and ML components/workflows/architectures.

The core contributions of this work are to shift the perspective of SWE-based NFRs into the ML context to reuse existing terms with only a slight redefinition of the SWE definitions. Furthermore, Habibullah et al. [25] also establishes a taxonomy of NFRs with slight changes in definitions. Therefore, another core contribution is that this paper provides a traditional SWE perspective on ML NFRs, or a novel quality model, to be used in exchange or supplementary to Habibullah et al. [25]'s general quality model.

3.1 Adaptability

Traditional Definition: A system is adaptable based on how easily its behavior can be changed to meet new needs.

ML Definition: The degree to which a model accepts modifications to its architecture without degrading performance.

Rationale: Although the traditional definition is broader in scope, its core intent is preserved when applied to ML models, whether simplistic or based on deep learning (DL) architectures. In the ML context, adaptability focuses specifically on architectural changes—such as the addition of network layers or functional components—and assesses whether such modifications cause performance regression after the modified architecture is retrained.

3.2 Affordability

Traditional Definition: The degree to which a system meets the outlined cost allocation.

ML Definition: The degree to which data compilation, model training, and model deployment collectively satisfy their respective cost allocations.

Rationale: Training and deploying ML models entail substantial computational expenditure. The original definition remains applicable but must be extended to cover three distinct cost phases in the ML pipeline: data acquisition and preparation, model training, and ongoing model stability during deployment. Affordability in this context thus evaluates whether the end-to-end ML workflow adheres to customer-defined

cost constraints.

3.3 Complexity

Traditional Definition: A metric of a system's interdependency, modularity, and code readability.

ML Definition: A metric of a model's layer interdependency and connectivity.

Rationale: This NFR undergoes relatively limited change in the ML context, as the ML community has already established complexity in terms of network depth and overall architectural size. Accordingly, the ML definition preserves the spirit of the SWE notion of interdependency while reframing it around the structural properties of neural network architectures.

3.4 Maintainability

Traditional Definition: Characteristics related to the effort needed to make modifications, including corrections, improvements, or adaptations to software changes in environment, requirements, and functional specifications [27].

ML Definition: The degree to which additional training data of learned classes can be incorporated without degrading performance on existing classes, with the goal of improving overall model performance.

Rationale: This definition shifts noticeably from its SWE counterpart. While adaptability addresses architectural changes, maintainability in ML becomes data-centric: it captures the quality by which the introduction of supplementary data advances model performance without inducing overfitting or negative interference with previously learned behaviors.

3.5 Modifiability

Traditional Definition: The degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.

ML Definition: The ability to accommodate changes—whether architectural, data-related, or task-related—without compromising model performance.

Rationale: In the ML context, modifiability serves as an umbrella NFR that subsumes three interconnected quality dimensions: (1) the degree of change when altering the model architecture (adaptability), (2) the impact of adding data for existing learned classes (maintainability), and (3) the capacity to incorporate entirely new data categories (modularity). A composite metric capturing variance across these three sub-dimensions would be appropriate; for instance, a model may demonstrate high architectural adaptability while exhibiting poor data-level maintainability.

3.6 Modularity

Traditional Definition: The ability to handle an increase in workload without impacting performance, or to quickly expand the architecture.

ML Definition: The degree to which a model can learn new classes from additional training data without degrading performance on existing classes.

Rationale: Modularity in the ML context is closely related to scalability, reframed around the model's capacity to extend its learned knowledge base. A modular ML model should incorporate new class representations with relative ease while sustaining performance on previously learned classes. This NFR intersects with architectural design, learning task formulation, and hyperparameter configuration. A suitable metric for this property may be the number of training epochs required to achieve convergence on new data without regression on prior data.

3.7 Performance

Traditional Definition: The degree to which the response time, processing time, and throughput rates of a system meet specified requirements.

ML Definition: The ratio of correct inferences to the total number of possible inferences.

Rationale: Although the definition changes substantially, the ML counterpart aligns with established evaluation metrics in ML and DL research, including precision, recall, accuracy, and F1 score [28]. This definition reflects the standard by which predictive model quality is assessed in academic and applied settings.

3.8 Reliability

Traditional Definition: The ability of a system to garner trust given its demonstration of dependability, availability, integrity, safety, and maintainability.

ML Definition: The ability of a model to garner trust through consistent demonstration of availability, security, and performance.

Rationale: Analogous to its SWE counterpart, ML reliability functions as an overarching quality attribute whose realization depends on multiple constituent NFRs. Ensuring reliability in an ML model requires

addressing availability, security, and sustained performance. These attributes collectively guarantee that the model behaves consistently and fulfills its intended purpose, mirroring the compositional role of reliability in traditional software engineering.

3.9 Reusability

Traditional Definition: The degree to which an asset can be used in more than one system or in building other assets.

ML Definition: The degree to which a pre-trained model can be fine-tuned for a new application domain.

Rationale: This NFR transfers naturally to the ML context. Pre-trained DL models, such as the YOLO architecture trained on the COCO dataset, can be fine-tuned for specialized recognition tasks—identifying specific vehicle types or animal species—and achieve satisfactory performance with limited additional training. Reusability thus measures the transferability of a model and the quality of its performance on target downstream tasks following fine-tuning.

3.10 Robustness

Traditional Definition: The degree to which a system maintains its regular behavior when encountering execution errors or receiving incorrect input.

ML Definition: The ability of a model to maintain consistent performance when exposed to unexpected transformations in input data. This NFR is subsumed under reliability and security.

Rationale: The ML interpretation of robustness is well-aligned with accepted usage in the research community, where it is primarily associated with model resilience to adversarially perturbed or corrupted inputs. It bears conceptual proximity to performance and complexity, as more complex models may exhibit different robustness profiles depending on the nature of input perturbations.

3.11 Security

Traditional Definition: The degree to which a system, product, or component prevents unauthorized access to or modification of computer programs or data [29].

ML Definition: The degree to which a model is protected against threat actors attempting to gain unauthorized access to the model or its components. This NFR is subsumed under reliability and intersects with robustness.

Rationale: Security is an increasingly recognized NFR in the ML community. In the ML domain, security threats primarily manifest as adversarial perturbation attacks, in which a threat actor crafts inputs designed to disrupt model inference or compromise the surrounding system [28, 30]. While historically considered lower priority than in traditional SWE contexts, ongoing research highlights the critical importance of adversarial robustness as a component of ML security.

3.12 Serviceability / Supportability

Traditional Definition: The ease with which service can be performed on a component or system when needed.

ML Definition: The ease with which a model can be retrained or fine-tuned when needed.

Rationale: This NFR parallels maintainability but emphasizes the model's ability to integrate within and adapt to production systems. It captures practical considerations such as model complexity versus deployability: while a deep neural network may learn highly complex patterns, its operational complexity may make it less suitable for production environments compared to a smaller, lighter-weight model. Serviceability thus quantifies the trade-off between model capability and the practicality of ongoing model management.

3.13 Stability

Traditional Definition: The risk of unexpected effects resulting from system modifications [27].

ML Definition: The degree to which a model maintains sufficient performance as data distributions evolve over time; i.e., the model's relevance across its operational lifespan.

Rationale: In the ML context, stability is reoriented from modification-induced side effects to temporal performance consistency. A stable model sustains adequate predictive quality over its deployment lifespan. In domains where data distributions shift gradually, model stability is more readily achieved; however, stability is not precluded in rapidly evolving domains, provided the model demonstrates sufficient generalization when confronted with emerging data trends.

3.14 Sustainability

Traditional Definition: The ability to meet present needs without compromising the ability of future

generations to meet their own needs.

ML Definition: The degree to which a model meets its energy consumption allocation during both training and deployment stages.

Rationale: Sustainable ML parallels sustainable SWE but must account for two distinct energy-intensive phases: model training and model inference at deployment. Large-scale models, such as large language models (LLMs), impose significant energy costs at both stages. This NFR ensures that total energy consumption across training and deployment remains within the boundaries defined by the customer or operational constraints.

3.15 Testability

Traditional Definition: The degree of effectiveness and efficiency with which test criteria can be established and tests can be performed to determine whether those criteria have been met [29].

ML Definition: The amount of actionable information that can be extracted from a single test case.

Rationale: Testability undergoes the most substantial transformation of all NFRs considered here. In the ML context, testability is closely coupled with model complexity: more complex models require more extensive testing to verify adequate training and convergence. For example, evaluating an LLM requires substantial information gain per test case to confirm appropriate behavior, whereas a support vector machine (SVM) requires far fewer test-case observations to establish convergence confidence. Testability thus serves as a measure of test case informativeness scaled to model complexity.

3.16 Summary

Table 3 provides a consolidated overview of all redefined ML Model NFRs, mapping each from its traditional SWE definition to its ML-specific counterpart. The six System NFRs, which are determined by the surrounding system rather than the model itself, are presented in Table 4.

4 Discussion

There are two subsets of ML NFRs: System-related NFRs, which indirectly affect the ML model's quality by impacting the surrounding system, and model-related NFRs, which directly influence the model or its lifecycle. The six System NFRs are summarized in Table 4. For instance, availability is a system NFR, determined by the system's ability to make a model accessible, either locally or online, rather than by the model itself. These are the NFRs from traditional software NFRs that can easily translate into the ML context. One key criterion in separating the ML and system NFRs in Table 2 is to find an example of the NFRs existence in academia. While System NFRs may evolve into ML NFRs, such transformation could necessitate substantial redefinition, potentially leading to new NFRs distinct from traditional SWE definitions. This paper's scope is to simply convert the traditional NFR definition into ML and identify examples of their existence. NFRs that are only found within the ML context and present within academia, e.g., explainability, fairness, and retainability, are also out of this paper's scope [25]. It is important to note that this mapping reviewed 25 academic papers. It would be beneficial to expand this search in the future, especially as the field of MLE gains traction and more researchers add to the discussion.

In adapting definitions to the ML context, some hierarchical and tradeoff relationships among NFRs emerged, highlighting the importance of understanding these tradeoffs for accurate NFR mapping. This effort initiates the exploration of how tradeoffs affect ML NFRs. Further experimental and empirical analysis is necessary to firmly establish the NFR tradeoffs.

4.1 Tradeoffs

Traditional NFRs involve tradeoffs where enhancing one, such as security, often compromises another, like performance. Understanding these tradeoffs is crucial to prevent inadvertently impairing the system while addressing specific NFRs. While exploring the novel context of the ML definitions, the analysis and work of this paper also examines the potential tradeoff space between the ML NFRs. This paper finds that some tradeoffs also hold when transferred into the ML domain, e.g., security hindering performance [56]. There are several discovered tradeoffs that logically conflict given the definitions and rationale in Table 1. The most apparent tradeoffs are in the areas of ML that are the easiest to translate: testability, scalability, maintainability, and performance. These NFRs particularly deal with the typical stages of the ML lifecycle: testing, training, and updating data/architecture. Testability, using our definition, has a tradeoff with scalability due to larger models becoming harder to test. One example of this is testing LLMs where the tests

Table 4. System NFRs.

NFR	SWE Definition	Rationale
Availability	Time frame in which the system functions normally and without failures. Measured as the percentage of total application downtime over a defined period [35].	The availability of a model is affected by the system that contains the model. If the system is accessible, the model will be accessible.
Compatibility	How well does a system function when there are other applications running at the same time [54].	The compatibility of a model will be determined by the system. The interaction of the system with the model should always be the same, but it is the interaction of the system with its environment that might change.
Interoperability	Ability to exchange information and communicate with internal and external applications and systems [35, 54].	The interoperability of the model will depend on the system. If the system is interoperable with other systems, the model contained within the system, will be too.
Portability	Capability of the system to be transferred from one environment to another [55].	The portability of the model will depend on the system. If the system is portable to other platforms, the model contained within the system will be too.
Recoverability/ Recovery	Ability of a system to recover key information in the event some disaster takes place that causes the loss of information. Not only does it relate to the capability to recover information but the speed at which it does so [35].	The recoverability of a model is the result of the recoverability of the system. The amount of information recoverable and speed will be dependent on the system.
Usability	Specify the end-user-interactions with the system and the effort required to learn, operate, prepare input, and interpret the output of the system [35].	The usability of the model will always be the same, but it is the system's responsibility to make it as usable as possible for the user.

require difficult, human-like cognitive benchmarks, rather than traditional curated test sets for simpler models [57]. Additionally, testability and security have a negative tradeoff due to security practices in ML requiring training on perturbed datasets and testing, in some regard, it requires different benchmarks to test the network [30]. Performance is another NFR with discovered tradeoffs against security and scalability. Security and scalability require benchmarks, perturbed data, and have architectural complexity, which can impact performance and complexity NFR. Further exploration is needed to determine if traditional NFR tradeoffs apply within the ML context. It is important to note that while these suggested NFR tradeoffs are sound based on the analysis of the above NFR mapping, they are preliminary observations and should be evaluated experimentally in future work.

4.2 Model NFR Taxonomy

A taxonomy is a method of organizing concepts or information into groups. Our defined NFRs, which are derived from existing SWE NFRs, can be resolved into 4 categories: *ML evaluability*, *ML flexibility*, *ML resilience*, and *ML sustainability*. Figure 2 shows the visualization of our taxonomy of Model NFRs. By adding this structure to our NFRs, these groupings not only clarify the important aspects of ML but also help reveal potential new NFRs or tradeoffs. Furthermore, this taxonomy facilitates analysis by identifying patterns and measuring coverage of NFRs.

- *ML Evaluability:* Encompasses the testing and performance aspects of ML. Complexity is included in this category because complexity affects the ability to test an ML model. ML testing aims to ensure correctness and efficiency.

- *ML Flexibility:* Encompasses the ability to retrain, adapt, or configure a model's classifications through the adaptation of its dataset or architecture. ML flexibility prioritizes adaptation and evolution.

- *ML Resilience:* Encompasses the ML model's ability to be resilient against changes in data trends (both short term and long term), erroneous input, and perturbed input from threat actors. ML resilience focuses on stability and dependability in situations where the model might produce

Table 5. A table of ML testing publications which contain NFR(s) derived in the above taxonomy.

ID	Paper Title	NFRs Mentioned
1	*DeepGauge: multi-granularity testing criteria for deep learning systems*	Performance, Robustness
2	*Aries: Efficient Testing of Deep Neural Networks via Labeling-Free Accuracy Estimation*	Performance, Robustness
3	*Robust Test Selection for Deep Neural Networks*	Performance, Reliability, Robustness, Complexity
4	*DeepCT: Tomographic Combinatorial Testing for Deep Learning Systems*	Robustness, Security
5	*DeepGini: prioritizing massive tests to enhance the robustness of deep neural networks*	Reliability, Performance, Complexity, Security, Robustness
6	*Structural Test Coverage Criteria for Deep Neural Networks*	Reliability, Robustness, Complexity
7	*Prioritizing Test Inputs for DNNs Using Training Dynamics*	Reliability, Testability
8	*FuzzGAN: A Generation-Based Fuzzing Framework for Testing Deep Neural Networks*	Performance, Security, Robustness
9	*DeepStellar: model-based quantitative analysis of stateful deep learning systems*	Reliability, Security
10	*Revisiting Neuron Coverage Metrics and Quality of Deep Neural Networks*	Reliability, Robustness
11	*Effective white-box testing of deep neural networks with adaptive neuron-selection strategy*	Performance, Reliability, Robustness

Table 6. A table of MLOps publications which contain NFR(s) derived in the above taxonomy.

ID	Paper Title	NFRs Mentioned
12	*DNN-Powered MLOps Pipeline Optimization for Large Language Models*	Performance, Affordability, Reliability, Adaptability
13	*Towards Modular Machine Learning Pipelines*	Modularity, Complexity
14	*MLOps Maturity Assessment: A Case Study and a Guide for Improvement*	Performance, Reliability, Security, Maintainability
15	*Model Reliability and Performance through MLOps: Tools and Methodologies*	Performance, Reliability, Complexity, Robustness
16	*Visualizing Secure MLOps (MLSecOps): A Practical Guide for Building Robust AI/ML Pipeline Security*	Security, Robustness
17	*Pipeline Probe: A Tool for Automated Quality Assessment in MLOps*	Performance, Reliability, Testability
18	*Decoding MLOps: Bridging the Gap Between Data Science and Operations for Scalable AI Systems*	Performance, Complexity, Sustainability
19	*Scalable Kubernetes-Based Data Engineering Pipelines with Airflow and Kubeflow for Industrial IoT Analytics*	Performance, Reliability, Modularity, Adaptability, Maintainability

incorrect input.

- *ML Sustainability:* Encompasses the support for an ML model that might affect the model's architecture or hyperparameter configuration. ML sustainability highlights NFRs which consider long-term viability and efficiency in using ML models.

There are a few impacts and major contributions from aligning model NFRs into this taxonomy: ensuring correctness and efficiency through *ML Evaluability*, promoting adaptability and evolution via *ML Flexibility*, improving long term viability through *ML Sustainability*, and deploying a stable and dependable model with *ML Resilience*. This taxonomy aids in aligning technical goals with strategic needs, enabling more informed decisions during the development of ML applications. Ideally, developers use these NFRs to plan and make design decisions at the model level. Through iterative refinement and contributions from other researchers and academia, this taxonomy can evolve and expand, becoming more comprehensive and adaptable to emerging NFRs in the field of ML. Refining this preliminary taxonomy leads to a deeper understanding of ML NFRs, improved prioritization of trade-offs, and the identification of emerging gaps in current ML development. Finally, once refined, a structured and well-understood quality model can be used to deploy and develop well-engineered ML applications.

4.3 Empirical Literature Mapping

To validate the relevance of the mapped NFRs, this work conducted a targeted literature mapping of

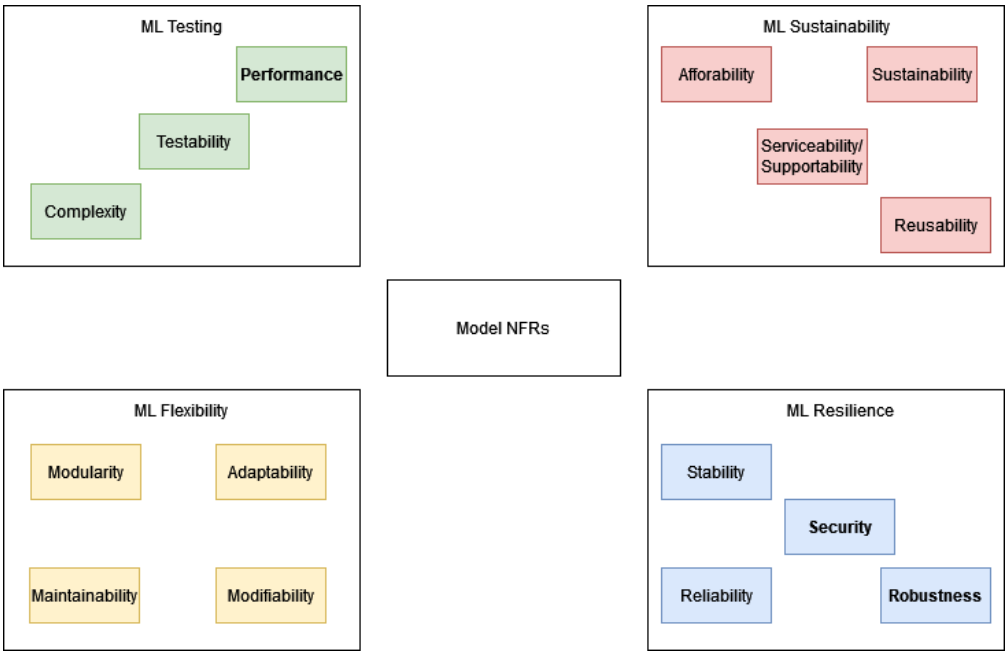


Figure 2. Shows the taxonomy of our NFRs: ML testing, ML sustainability, ML flexibility, and ML resilience. Bolded NFRs are ones which are defined in ML academia.

peer-reviewed ML papers published between 2019 and 2025. Each paper was manually searched for implicit references to the 15 Model NFR taxonomy. Because testing is inherently tied to the fulfillment of requirements, the selected papers are based on ML testing, from top conferences. The following Tables 5, 6 and 7 show a selection of the sources and their found NFR(s). The criteria are to find out if the NFRs are mentioned by name, showing their use in certain research areas. This is to show the importance of taxonomy, but also to identify the areas in which they are used. Table 5 is examining top ML testing papers.

After examining the selection of the testing papers, the area of testing covers complexity, performance, reliability, robustness, security, testability. This is only a third of the taxonomy. This shows that the ML testing grouping is sensible. ML testing also covers most of ML Resilience outside of stability. ML Sustainability and ML Flexibility are left completely unmapped from identifying NFRs within ML testing. Now, to do this, the areas of MLOps & workflows are mapped with the full list of NFRs, collecting the top papers from those research areas.

After surveying top MLOps/Workflow publications, as shown in Table 6, more NFRs arose, such as adaptability, affordability, maintainability, modularity, and sustainability. Interestingly, complexity, security, and especially performance, are still commonly used outside of ML testing. Performance is a key NFR to prove any extant behavior learned during training. Table 7 shows the mapping between Green ML and the NFRs to identify the remaining NFRs, namely ML sustainability.

From this literature mapping, this paper finds that performance is mapped most across ML testing, MLOps & Workflows, and Green ML. The findings are that the quality model needs a more expansive search to identify papers that belong to the respective NFR that could not identify in a total of n=25 papers across three ML domains. Alternatively, the NFRs that were not mentioned in this literature mapping were modifiability and serviceability, which could possibly be in alternative domains. From Figure 3, a few trends are noticeable. ML Testing papers are predominantly focused on robustness and reliability which become less popular in Green ML. Performance NFRs are mentioned commonly amongst all three domains. Green ML uses sustainability and complexity in all it found sources. MLOps had nearly the same NFRs as ML testing; however, they had some mentions of affordability, adaptability, modularity, and maintainability. In conclusion, a more comprehensive literature mapping could identify the NFRs which are not mentioned.

5 Related Work

This paper is not the first to suggest the use of requirements to guide ML development. Others have proposed the necessity of applying SWE practices to ML development. For instance, Masuda et al. [58]

Table 7. A table of Green ML publications which contain NFR(s) derived in the above taxonomy.

ID	Paper Title	NFRs Mentioned
20	*Green Machine Learning*	Performance, Complexity, Sustainability
21	*Towards Sustainable Machine Learning: Energy-Efficient Algorithmic strategies for Environmental Sensor Data*	Performance, Sustainability, Complexity
22	*Energy-Efficient Machine Learning on the Edges*	Performance, Sustainability, Complexity, Reliability
23	*Green MLOps to Green GenOps: Energy consumption in discriminative and generative AI operations*	Modularity, Reusability, Sustainability, Complexity
24	*Automated Green Machine Learning for Condition -Based Maintenance*	Performance, Reliability, Sustainability, Complexity
25	*Decoding MLOps: Bridging the gap between Data Science and Operations for Scalable AI Systems*	Complexity, Performance, Sustainability

Paper ID	Performance	Testability	Complexity	Modularity	Adaptability	Maintainability	Modifiability	Affordability	Sustainability	Serviceability	Reusability	Stability	Security	Robustness	Reliability
1	X													X	
2	X													X	
3	X		X											X	X
4													X	X	
5	X		X										X	X	X
6			X											X	X
7		X													X
8	X												X	X	
9													X		X
10														X	X
11	X													X	X
12	X				X			X							X
13			X	X											
14	X					X							X		X
15	X		X											X	X
16													X	X	
17	X	X													X
18	X		X						X						
19	X			X	X	X									X
20	X		X						X						
21	X		X						X						
22			X	X					X		X				
23			X	X					X		X				
24	X		X						X						X
25	X		X						X						
Count	16	2	12	4	2	2	0	1	7	0	2	0	6	11	13

Figure 3. Shows the taxonomy of our NFRs: ML testing, ML sustainability, ML flexibility, and ML resilience. Bolded NFRs are ones which are defined in ML academia.

state that the use of requirements can better align ML development with traditional software development. However, their paper then pivots to the verification of requirements and model quality, leaving the definitions of ML requirements unexplored.

The quality of ML models is a highly discussed topic, but research into formally defining quality in ML is scarce [6]. Ali et al. [6] found that most often, authors seek privacy, fairness, accuracy, performance, and security as quality characteristics of their models. However, the work in this paper defines these quality characteristics in relation to the ML development phases.

Some authors specifically express the need for defining NFRs within the ML context [5, 24–26]. Habibullah et al. [24, 25] and Horkoff [5] explain that NFRs in the ML context may look quite different. For instance, they suggest that fairness and transparency

are two NFRs that are not as notable in traditional software because they both can be controlled within conventional code. However, in data-driven modeling, fairness cannot be controlled by the engineer and transparency is inherently difficult, so they take precedence. Conversely, other NFRs may become less important or even irrelevant, which this paper has also noted as system-centric NFRs. The work of Habibullah et al. [25] and Horkoff [5] forms a literature foundation for the challenges of utilizing standard NFRs for ML. The authors explore the scope of NFRs within the ML context and discuss how granularity could affect the applicability of NFRs to ML, i.e., system-centric and model-centric requirements [24]. Their work also begins the discussion on which NFRs are specifically important in ML development but leaves room for discussion of ML-specific definitions for some of the NFRs. Meanwhile Bajraktari et al. [26], a later publication in the field of NFRs for ML, survey existing work in the field and summarize their findings in their own quality model. Many of their proposed NFRs align with Habibullah et al., but with a few alterations to accommodate the other surveyed papers.

In contrast to Habibullah et al. [25], Horkoff [5] and Bajraktari et al. [26], our work starts from the list of traditional NFRs and analyzes which of these NFRs can map to an ML context. This approach ensures the completeness of the mapping from traditional to ML-based systems and maximizes the application of traditional definitions to ML-based definitions. Additionally, for NFR definitions that do not directly map to the ML context, this work surveys academic literature to find documented examples of NFRs in the ML context and adjust their definitions accordingly.

Some authors note the existence of new NFRs for ML that have not previously applied to software, e.g., explainability [20]. This NFR is not necessary in traditional software systems that function deterministically. However, explainability is important in complex data-driven models where the learned features are not guaranteed. Köhl et al. [20] present explainability as an NFR for ML systems and explore the scope of model explainability within the literature.

While previous research has explored the topic of quality models or NFRs for ML, this work is one of the first to specifically define NFRs in the ML context. This novelty includes the separation of traditional SWE NFRs into system and model NFRs, as well as the redefinition of some traditional NFRs to better match the development and capabilities of ML models.

6 Future Work

This work focuses on assessing and mapping traditional software NFRs to the ML context. This provides an important foundation for developing research and points to areas for future research. There are some aspects of ML development and deployment that diverge from traditional software that current NFRs cannot properly describe. Current NFR gaps stem from the unpredictable nature of data-driven training, affecting both feature learning and usage. Despite using curated data, convergence is not guaranteed. This leads to a few directions of future work including how new definitions change, establishing NFR tradeoffs, and implementations.

It also may be helpful to define ML NFR groupings based on similarities. Some authors have already provided potential groupings of ML NFRs [25]. As the traditional and new NFRs for ML solidify, further research should be dedicated to updating NFR groups. These categories could help ML engineers adapt to and apply the new NFR definitions cohesively for ML systems. Additionally, NFR categories in ML development could illuminate new and shifting tradeoffs, given the evolution of NFR definitions and the emergence of ML-specific NFRs. While some tradeoffs, like security versus performance, persist from traditional software to ML, others may become irrelevant or transform, introducing entirely new dynamics. Further research is needed to empirically detail how these tradeoffs evolve in the ML context.

Finally, the data-centricity of ML makes model NFRs particularly difficult to verify because many traditional testing methods do not map to the ML-context, e.g., branch coverage and other white-box testing methods. Therefore, future work will also research measurement and verification techniques for ML NFRs.

7 Conclusion

The purpose of the present paper is to map existing NFRs into the ML context as a call for consensus towards developing requirements engineering quality models for ML. Defining requirements for a software system enables concrete communication between the customer and developer over what the system will do. More specifically, NFRs describe the quality attributes of the software. As the necessity of ML to handle big data continues to grow, NFR definitions need to expand to encompass software that utilizes ML models. However, many current NFR definitions do not apply to ML because many traditional NFRs are

system-centric and are therefore beyond the scope of a single ML model. Some of the traditional NFRs that do apply to ML need to be redefined because of the variation in how ML is developed and deployed versus traditional software.

This paper mapped 24 NFRs commonly used in traditional software development to ML development. Of the 24 traditional NFRs, this paper found 16 NFRs are pertinent to ML models within applications whereas the remaining NFRs are more system-centric, focusing on qualities around the model. Specifically, ML model NFRs include adaptability, affordability, complexity, maintainability, modifiability, performance, reliability, reusability, robustness, modularity, security, serviceability/supportability, stability, sustainability, and testability. This mapping found that only security and sustainability definitions map directly from traditional software to ML, whereas the rest of the NFR definitions had to be adjusted to the ML context. This paper included citations of ML works which implicitly use our NFR definitions to show the strength of the proposed definitions and taxonomy, however, future work should empirically evaluate the mapping presented in this work. This paper briefly discusses some discovered high-level tradeoffs and taxonomies. Future work will focus on experimental evaluation of the proposed tradeoffs, as well as discussion of NFR verification, which becomes difficult in ML due to data-centricity.

Ultimately, the authors hope that this paper not only promotes thought-provoking ideas for novel ML-NFRs but also energizes fellow researchers to contribute to our work and engage in debate on the topic. Furthermore, the authors invite future research to expand upon or refine the definitions presented in this paper, fostering a dynamic evolution of understanding in the field.

Data Availability Statement

Data will be made available on request.

Funding

This work was supported without any funding.

Conflicts of Interest

The authors declare no conflicts of interest.

AI Use Statement

The authors declare that no generative AI was used in the preparation of this manuscript.

Ethical Approval and Consent to Participate

Not applicable.

References

[1] Shinde, P. P., & Shah, S. (2018, August). A review of machine learning and deep learning applications. In *2018 Fourth international conference on computing communication control and automation (ICCUBEA)* (pp. 1-6). IEEE. [CrossRef]

[2] Isbell, C., Littman, M. L., & Norvig, P. (2023). Software engineering of machine learning systems. *Communications of the ACM, 66*(2), 35-37. [CrossRef]

[3] Chung, L., Nixon, B. A., Yu, E., & Mylopoulos, J. (2012). *Non-functional requirements in software engineering* (Vol. 5). Springer Science & Business Media.

[4] Glinz, M. (2007, October). On non-functional requirements. In *15th IEEE international requirements engineering conference (RE 2007)* (pp. 21-26). IEEE. [CrossRef]

[5] Horkoff, J. (2019, September). Non-functional requirements for machine learning: Challenges and new directions. In *2019 IEEE 27th international requirements engineering conference (RE)* (pp. 386-391). IEEE. [CrossRef]

[6] Ali, M. A., Yap, N. K., Ghani, A. A. A., Zulzalil, H., Admodisastro, N. I., & Najafabadi, A. A. (2022). A systematic mapping of quality models for AI systems, software and components. *Applied Sciences, 12*(17), 8700. [CrossRef]

[7] Boehm, B. W., Brown, J. R., & Lipow, M. (1976). Quantitative evaluation of software quality. *Software Engineering: Barry W. Boehm's Lifetime Contributions to Software Development, Management, and Research*, 25.

[8] Miguel, J. P., Mauricio, D., & Rodríguez, G. (2014). A review of software quality models for the evaluation of software products. *arXiv preprint arXiv:1412.2977*. [arXiv]

[9] Washizaki, H. (Ed.). (2025). *SWEBOK: Guide to the software engineering body of knowledge (v4.0a)*. IEEE Computer Society. https://www.laobu.com/ai/book_se_swebok_4_files/swebok4a.pdf

[10] Chung, L., & Supakkul, S. (2004, May). Representing nfrs and frs: A goal-oriented and use case driven approach. In *International Conference on Software Engineering Research and Applications* (pp. 29-41). Berlin, Heidelberg: Springer Berlin Heidelberg. [CrossRef]

[11] Ruparelia, N. B. (2010). Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes, 35*(3), 8-13. [CrossRef]

[12] Macaulay, L. A. (2012). *Requirements engineering*. Springer Science & Business Media.

[13] Agarwal, B. B., & Tayal, S. P. (2009). *Software Engineering*. New Age International.

[14] Luo, L. (n.d.). *Software testing techniques: Technology maturation and research strategy* [Class report, Carnegie Mellon University]. Carnegie Mellon University. https://www.cs.cmu.edu/~luluo/Courses/17939Report.pdf

[15] Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media. https://dl.acm.org/doi/abs/10.5555/3153997

[16] Kim, K. G. (2016). Book review: Deep learning. *Healthcare Informatics Research, 22*(4), 351–354. [CrossRef]

[17] Paleyes, A., Urma, R. G., & Lawrence, N. D. (2022). Challenges in deploying machine learning: a survey of case studies. *ACM computing surveys, 55*(6), 1-29. [CrossRef]

[18] Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., ... & Dennison, D. (2015). Hidden technical debt in machine learning systems. *Advances in neural information processing systems*, 28.

[19] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems, 33*, 1877-1901.

[20] Köhl, M. A., Baum, K., Langer, M., Oster, D., Speith, T., & Bohlender, D. (2019, September). Explainability as a non-functional requirement. In *2019 IEEE 27th international requirements engineering conference (RE)* (pp. 363-368). IEEE. [CrossRef]

[21] Chazette, L., & Schneider, K. (2020). Explainability as a non-functional requirement: challenges and recommendations. *Requirements Engineering, 25*(4), 493-514. [CrossRef]

[22] Sheh, R., & Monteath, I. (2018). Defining explainable AI for requirements analysis. *KI-Künstliche Intelligenz, 32*(4), 261-266. [CrossRef]

[23] Vogelsang, A., & Borg, M. (2019, September). Requirements engineering for machine learning: Perspectives from data scientists. In *2019 IEEE 27th international requirements engineering conference workshops (REW)* (pp. 245-251). IEEE. [CrossRef]

[24] Habibullah, K. M., Gay, G., & Horkoff, J. (2023). Non-functional requirements for machine learning: Understanding current use and challenges among practitioners. *Requirements Engineering, 28*(2), 283-316. [CrossRef]

[25] Habibullah, K. M., Gay, G., & Horkoff, J. (2022, May). Non-functional requirements for machine learning: An exploration of system scope and interest. In *Proceedings of the 1st Workshop on Software Engineering for Responsible AI* (pp. 29-36). [CrossRef]

[26] Bajraktari, E., Krause, T., & Kücherer, C. (2024). Documentation of non-functional requirements for systems with machine learning components. In *Joint Proceedings of REFSQ-2024 Workshops, Doctoral Symposium, Posters & Tools Track, and Education and Training Track; co-located with REFSQ 2024: Winterthur, Switzerland, 8-11 April 2024* (Vol. 3672). RWTH Aachen.

[27] ISO/IEC. (2001). *ISO/IEC 9126-1:2001 Software engineering — Product quality — Part 1: Quality model* (ISO/IEC 9126-1:2001). https://www.iso.org/standard/22749.html

[28] Zhang, J. M., Harman, M., Ma, L., & Liu, Y. (2020). Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering, 48*(1), 1-36. [CrossRef]

[29] International Organization for Standardization, & Technical Committee ISO/IEC JTC 1, Information technology. Subcommittee SC 7, Software and systems engineering. (2011). *Systems and Software Engineering: Systems and Software Quality Requirements and Evaluation (SQuaRE): System and Software Quality Models*. ISO. https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en

[30] Zhang, X., Zheng, X., & Mao, W. (2021). Adversarial perturbation defense on deep neural networks. *ACM Computing Surveys (CSUR), 54*(8), 1-36. [CrossRef]

[31] Lieberherr, K. (1995, October). Workshop on adaptable and adaptive software. In Addendum to the proceedings of the 10th annual conference on Object-oriented programming systems, languages, and applications (pp. 149-154).

[32] Kanda, H., Okamura, H., & Dohi, T. (2023, August). A Note on Optimal Retraining Strategy for ML Systems. In *2023 10th International Conference on Dependable Systems and Their Applications (DSA)* (pp. 730-733). IEEE. [CrossRef]

[33] Hu, X., Liu, W., Bian, J., & Pei, J. (2020, August). Measuring model complexity of neural networks with curve activation functions. In *Proceedings of the 26th ACM SIGKDD International Conference on knowledge discovery & data mining* (pp. 1521-1531). [CrossRef]

[34] Myung, I. J. (2000). The importance of complexity in model selection. *Journal of mathematical psychology, 44*(1), 190-204. [CrossRef]

[35] Paradkar, S. (2017). *Mastering non-functional requirements: analysis, architecture, and assessment*. Packt Publishing Ltd.

[36] Weiss, G. M., & Provost, F. (2001). *The effect of class distribution on classifier learning: an empirical study*. Rutgers University. [CrossRef]

[37] Poort, E. R., Martens, N., Van De Weerd, I., & Van Vliet, H. (2012, March). How architects see non-functional requirements: Beware of modifiability. In *International Working Conference on Requirements Engineering: Foundation for Software Quality* (pp. 37-51). Berlin, Heidelberg: Springer Berlin Heidelberg. [CrossRef]

[38] Biondi, A., Nesti, F., Cicero, G., Casini, D., & Buttazzo, G. (2019). A safe, secure, and predictable software architecture for deep learning in safety-critical systems. *IEEE Embedded Systems Letters, 12*(3), 78-82. [CrossRef]

[39] Sujon, K. M., Hassan, R., Choi, K., & Samad, M. A. (2025). Accuracy, precision, recall, f1-score, or MCC? empirical evidence from advanced statistics, ML, and XAI for evaluating business predictive models. *Journal of Big Data, 12*(1), 268. [CrossRef]

[40] Avizienis, A., Laprie, J. C., Randell, B., & Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing, 1*(1), 11-33. [CrossRef]

[41] Bombonatti, D., Goulão, M., & Moreira, A. (2017). Synergies and tradeoffs in software reuse – a systematic mapping study. *Software, Practice & Experience, 47*(7), 943-957. [CrossRef]

[42] Niu, S., Liu, Y., Wang, J., & Song, H. (2021). A decade survey of transfer learning (2010–2020). *IEEE Transactions on Artificial Intelligence, 1*(2), 151-166. [CrossRef]

[43] Shahrokni, A., & Feldt, R. (2013). A systematic review of software robustness. *Information and Software Technology, 55*(1), 1-17. [CrossRef]

[44] Gupta, S., & Gupta, A. (2019). Dealing with noise problem in machine learning data-sets: A systematic review. *Procedia Computer Science, 161*, 466-474. [CrossRef]

[45] Madden, M. G., & Munroe, D. T. (2005). Multi-class and single-class classification approaches to vehicle model recognition from images. [CrossRef]

[46] Bae, H., Jang, J., Jung, D., Jang, H., Ha, H., Lee, H., & Yoon, S. (2018). Security and privacy issues in deep learning. *arXiv preprint arXiv:1807.11655*. [arXiv]

[47] Khalid, F., Hanif, M. A., Rehman, S., Qadir, J., & Shafique, M. (2019, March). Fademl: Understanding the impact of pre-processing noise filtering on adversarial machine learning. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (pp. 902-907). IEEE. [CrossRef]

[48] Christopher, D. F. X., & Chandra, E. (2012). Prediction of software requirements stability based on complexity point measurement using multi-criteria fuzzy approach. *International Journal of Software Engineering & Applications, 3*(6), 101. [CrossRef]

[49] Penzenstadler, B., Raturi, A., Richardson, D., & Tomlinson, B. (2014). Safety, security, now sustainability: The nonfunctional requirement for the 21st century. *IEEE software, 31*(3), 40-47. [CrossRef]

[50] Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., & Peste, A. (2021). Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research, 22*(241), 1-124.

[51] Garousi, V., Felderer, M., & Kılıçaslan, F. N. (2019). A survey on software testability. *Information and Software Technology, 108*, 35-64. [CrossRef]

[52] Sun, Y., Huang, X., Kroening, D., Sharp, J., Hill, M., & Ashmore, R. (2018). Testing deep neural networks. *arXiv preprint arXiv:1803.04792*. [arXiv]

[53] Osisanwo, F. Y., Akinsola, J. E., Awodele, O., Hinmikaiye, J. O., Olakanmi, O., & Akinjobi, J. (2017). Supervised machine learning algorithms: classification and comparison. *International Journal of Computer Trends and Technology (IJCTT), 48*(3), 128-138.

[54] Adams, K. M. (2015). Compatibility, consistency, interoperability. In *Nonfunctional Requirements in Systems Analysis and Design* (pp. 131-153). Cham: Springer International Publishing. [CrossRef]

[55] Abran, A., Al-Sarayreh, K. T., & Cuadrado-Gallego, J. J. (2013). A standards-based reference framework for system portability requirements. *Computer Standards & Interfaces, 35*(4), 380-395. [CrossRef]

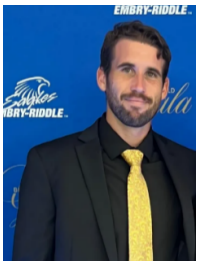
[56] Raghunathan, A., Xie, S. M., Yang, F., Duchi, J. C., & Liang, P. (2019). Adversarial training can hurt generalization. *arXiv preprint arXiv:1906.06032*. [arXiv]

[57] Valmeekam, K., Marquez, M., Olmo, A., Sreedharan, S., & Kambhampati, S. (2023). Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. *Advances in Neural Information Processing Systems, 36*, 38975-38987.

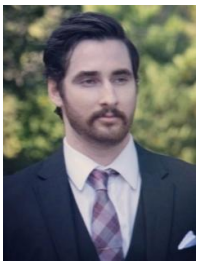
[58] Masuda, S., Ono, K., Yasue, T., & Hosokawa, N. (2018, April). A survey of software quality for machine learning applications. In *2018 IEEE International conference on software testing, verification and validation workshops (ICSTW)* (pp. 279-284). IEEE. [CrossRef]

Timothy Elvira received his Ph.D. in Electrical Engineering and Computer Science from Embry-Riddle Aeronautical University in 2025. His research centers on testing and evaluation methods for computer vision models. His other areas of research include software engineering for machine learning, deep learning, and large language models. (Email: elvirat@my.erau.edu)

Lynn Vonderhaar is a Ph.D. candidate at Embry-Riddle Aeronautical University, where he previously earned both his B.S. and M.S. degrees in Software Engineering. His research focuses on integrating Machine Learning into the field of Cybersecurity from a Software Engineering perspective. His additional research interests include Software Engineering education, particularly the incorporation of emerging AI trends into academia. (Email: vonderhl@my.erau.edu)



Juan Couder is a Ph.D. candidate at Embry-Riddle Aeronautical University, where he previously earned both his B.S. and M.S. degrees in Software Engineering. His research focuses on integrating Machine Learning into the field of Cybersecurity from a Software Engineering perspective. His additional research interests include Software Engineering education, particularly the incorporation of emerging AI trends into academia. (Email: ortizcoj@my.erau.edu)



Tyler Thomas Procko is an AI research scientist and knowledge graph engineer specializing in machine learning provenance, applied ontology, and explainable AI. He earned his Ph.D. in Computer Science from Embry-Riddle Aeronautical University, where his research examined the integration of knowledge graphs and large language models for software and ML traceability. He has conducted multi-year sponsored research with the U.S. Air Force Research Laboratory and has published on semantic technologies, NLP, and AI systems. (Email: prockot@my.erau.edu)

William C. Pate is a software engineer and recent graduate of Embry-Riddle Aeronautical University. He completed his master's degree in software engineering in May 2025 and has published research related to artificial intelligence and software requirements. William has also worked on projects for both Honeywell Aerospace and NASA. (Email: patew@my.erau.edu)

Omar Ochoa is an Associate Professor of Software Engineering and Computer Science at Embry-Riddle Aeronautical University, where he has served since Fall 2016. He earned his Ph.D. in Computer Science from the University of Texas at El Paso. With more than a decade of industry experience, Dr. Ochoa has contributed to projects at the Army Research Laboratory, IBM, and Hewlett Packard Enterprise. His research, supported by organizations including the NSF, FAA, U.S. Navy, and AFRL, focuses on integrating Software Engineering practices into Machine Learning. His academic interests also include Software Engineering education, particularly the incorporation of Agile methods into academia. (Email: ochoao@erau.edu)