Understanding Machine Learning Testing in Practice

Alfonso Cannavale,[1] Valeria Pontillo,[2] Andrea De Lucia,[1] Fabio Palomba[1]

[1] *Software Engineering (SeSa) Lab - Department of Computer Science, University of Salerno, Italy*
[2] *Gran Sasso Science Institute (GSSI), L'Aquila, Italy*
acannavale@unisa.it, valeria.pontillo@gssi.it, adelucia@unisa.it, fpalomba@unisa.it

Abstract

Context. Machine Learning is increasingly embedded in critical software systems, making their quality assurance a matter of growing concern. While the research community has proposed several techniques for testing ML-enabled systems, there is limited empirical evidence on whether these techniques are adopted in practice or align with developers' testing workflows. **Objective.** This paper presents a two-step empirical investigation aimed at characterizing the current landscape of ML testing in real-world development. Our goal is to understand how developers approach testing, whether proposed techniques are adopted, and what barriers hinder their implementation. **Methods.** We designed a mixed-method study that triangulates insights from two complementary sources: (1) a mining study of 398 open-source repositories to analyze implemented testing strategies and tool usage; and (2) a survey of 100 practitioners to capture perceptions, motivations, and practical challenges. **Results.** Our findings reveal that developers rely heavily on foundational strategies like *Smoke Testing* and *Rule-Based Checking*, implemented through custom testing logic built on general-purpose libraries (e.g., PyTest, NumPy). Conversely, we identified a critical adoption gap in specialized tools and advanced techniques such as *Metamorphic Testing*, which are rarely implemented despite their academic prominence. Our survey indicates that this gap is driven by practical barriers, including high integration costs and a poor fit with existing developer workflows. **Conclusion.** These findings suggest that future research and tooling must prioritize usability, integration, and a clearer alignment with the pragmatic needs of developers.

Keywords: Testing ML-Enabled Systems; Empirical Software Engineering; Software Quality Assurance; Software Engineering for Artificial Intelligence.

1. Introduction

Machine Learning (ML) is reshaping the software industry, powering applications across domains such as healthcare, finance, autonomous systems, and creative industries [1–3]. With the increasing integration of ML components into software systems, ensuring their reliability, robustness, and ethical soundness has become a pressing concern [4]. Indeed, these systems are no longer peripheral, but central to decision-making processes and critical infrastructures, making their quality assurance a matter of societal importance [5].

Among the various quality assurance activities designed to verify and validate the correctness, security, and fairness of ML-enabled systems, *software testing* remains one of the most essential practices [6]. Traditionally, software testing involves executing a system under controlled conditions to detect faults and ensure that its behavior aligns with expected outcomes [7]. However, in the context of ML, testing practices need to account for the unique characteristics of learning algorithms, such as the stochastic nature of model predictions, the strong dependency on data quality and distribution, and the frequent absence of well-defined test oracles [8]. These issues are further compounded by the nature of many models (e.g., Deep Neural Networks) and the high cost of generating realistic and reproducible test scenarios [4]. To address these challenges, the software engineering research community has proposed a variety of methods and techniques to (semi-)automatically verify the behavior and reliability of ML-enabled components. These include, for instance, traditional functional testing of model APIs [9, 10], adversarial testing to evaluate robustness against input perturbations [11–13], metamorphic testing to address the oracle problem [14–17], and neuron coverage analysis to examine internal model behavior [18–21]. Collectively, these approaches aim to go beyond traditional performance evaluation and provide more systematic guarantees about model behavior.

To illustrate the practical challenges motivating this work, consider a developer building an image classification system. A common validation strategy consists of training the model and evaluating its performance on a held-out validation set, for instance by measuring accuracy. If the model achieves a satisfactory score, it is typically considered acceptable for deployment. While this form of validation is widely adopted and useful to estimate generalization performance, it does not systematically verify important behavioral properties of the model. For instance, it does not assess whether predictions remain stable under semantically equivalent transformations of the input (e.g., changes in brightness or rotation), nor whether the model is robust to perturbations that may lead to incorrect predictions.

To address these limitations, the research community has proposed complementary testing techniques. *Metamorphic*

Testing verifies whether a system behaves consistently under predefined input transformations; in the context of image classification, this involves generating transformed versions of an image and checking whether the predicted label remains unchanged. In contrast, *Adversarial Testing* evaluates robustness against intentionally crafted perturbations, often imperceptible to humans, that are designed to cause misclassifications. These approaches aim to complement traditional validation by enabling a more systematic exploration of model behavior.

Despite this growing body of research, it remains unclear to what extent such techniques are actually adopted in real-world development. In practice, developers may continue to rely primarily on performance-based validation and simple testing strategies, potentially overlooking more systematic approaches proposed in the literature. Without such empirical insights, it is difficult to determine whether current research directions effectively address real-world needs or whether existing solutions are practical to integrate into ML development pipelines.

As such, gaining this understanding is essential to ensure that research efforts align with the realities of development practice and address actual needs and limitations encountered in the field. This may indeed inform researchers seeking to ground their contributions in real-world needs and tool developers aiming to create usable and effective solutions.

> ◎ **Objective of the Study.** This study aims to empirically understand how ML testing is *implemented* and *perceived* in practice, with the goal of contrasting real-world adoption patterns against established state-of-the-art concepts, identifying factors that may support the broader adoption of ML testing techniques.

To address this gap, we present a two-step empirical investigation [22] that characterizes the current landscape of ML testing in practice. Our study aims to explore the prevalence of specific *testing practices* across implementations and developer perceptions; the adoption and perception of *testing tools*; and the *limitations or barriers* that hinder the effective testing of ML systems. We design a mixed-method methodology that triangulates insights from complementary data sources: (1) a *mining study of open-source repositories* to examine testing behaviors in real-world ML projects; and (2) a *survey targeting practitioners* involved in ML development and testing to capture perceptions, motivations, and experiences.

Our findings reveal a hierarchy of ML testing practices. Developers primarily rely on pragmatic approaches such as *Smoke Testing* and *Rule-Based Checking*, typically implemented with custom solutions that leverage general-purpose libraries such as PyTest and NumPy. In contrast, advanced techniques from academia such as *Metamorphic Testing* are rarely adopted, largely due to inadequate tooling. Practitioners perceive specialized tools as poorly aligned with their workflows and too costly to integrate, leading them to favor custom solutions instead. We conclude by summarizing these insights and outlining future research directions to improve ML testing practices.

To sum up, this paper provides the following contributions:

1. An empirical investigation into ML testing in practice, spanning real-world implementations (via a mining study) and practitioners' perception (via a survey);

2. Empirically grounded observations that highlight the distinction between academic proposals and real-world testing needs, offering considerations for further research on more practical and adoptable ML testing solutions;

3. A publicly available replication package [23] including all materials to support transparency, reproducibility, and future research on ML testing practices.

Structure of the paper. Section 2 reviews the literature related to ML testing. Section 3 presents the research questions that guide our investigation, while Section 4 details the design of our two-phase empirical methodology. Section 5 reports the findings from the mining study and the survey. Section 6 interprets the results, reflecting on their implications for research and practice. Section 7 discusses threats to validity. Finally, Section 8 summarizes the contributions and outlines directions for future work.

2. Related Work

To contextualize our study within the broader research on ML software testing, we first outline the challenges of testing ML-enabled systems and summarize the main technical solutions proposed in the literature. We then report the most recent empirical studies on ML testing practices, identify their limitations, and motivate our study.

2.1. Challenges and Approaches to ML Testing

Software testing is a fundamental discipline that ensures the quality, reliability, and correctness of software systems [24]. The emergence of ML–based systems, however, has challenged many of the assumptions underlying traditional testing practices, often rendering established techniques inadequate or insufficient [4, 24–26].

In this context, the literature studies proposed by Zhang et al. [4], and Riccio et al. [8] have become influential reference points for understanding this evolving research area. Both reviews characterize ML testing, providing a foundational understanding of techniques, tools, and challenges discussed by the research community.

The core challenge in testing ML-based systems lies in the development paradigm. Traditional software systems are typically deterministic and deductive: given an input, they produce a predictable output based on rules explicitly programmed by developers. In contrast, ML systems derive their behavior inductively by generalizing patterns from data [4, 25]. These systems approximate functions or decision boundaries by optimizing over training examples. As a result, their behavior is shaped by statistical correlations, noise, and biases in the data, rather than by explicitly defined rules. This data-driven and probabilistic nature makes their outputs inherently less predictable,

especially for inputs not well represented in the training distribution. One of the most cited consequences is the *test oracle problem* [4, 26]. In traditional systems, the oracle, i.e., the mechanism for determining whether an output is correct, is usually unambiguous. In ML systems, defining a *correct* oracle becomes difficult, as the output can be subjective or lack a single right answer [24].

Beyond the oracle issue, ML testing faces additional challenges. Non-determinism in training procedures, caused by randomness in initialization, data sampling, or hardware execution, can lead to different models being learned from the same data and code, complicating reproducibility and regression testing [27]. Data quality and representativeness also play a critical role: subtle shifts in the data distribution can degrade performance, necessitating testing that accounts for a wide range of inputs, including edge and out-of-distribution cases [28]. Moreover, model evolution through continuous retraining introduces a moving target, making it difficult to define a stable testing baseline [29]. Finally, ML systems often involve complex pipelines with multiple data transformations, pre- and post-processing steps, and third-party libraries, which increase the surface for faults and require holistic, system-level testing strategies [30].

The growing awareness of the unique challenges posed by ML testing has led to increased research activity, as documented in recent surveys and literature reviews [4, 8, 31]. From the analysis of these works, several established research areas emerge to address the specific issues of ML testing. One of the most widely studied approaches is *metamorphic testing* [4], which addresses the test oracle problem by verifying whether a system's outputs behave consistently under defined input transformations, known as *metamorphic relations*. Instead of checking correctness directly, it checks that specific changes to the input lead to predictable (or invariant) changes in the output. For example, if the brightness of an image is altered, an object classifier should still recognize the same object. Frameworks such as Amsterdam and Corduroy automate the generation and evaluation of metamorphic relations [4, 8].

Another area of interest is *robustness testing*, particularly through the generation of *adversarial examples* [32]. These inputs introduce subtle, often imperceptible perturbations designed to mislead models into making incorrect predictions, and are supported by libraries such as the Adversarial Robustness Toolbox (ART) [4]. While adversarial techniques focus on precision-crafted attacks, complementary approaches like *fuzzing* (e.g., TensorFuzz, DeepHunter) and *symbolic execution* (e.g., DeepConcolic) aim to systematically explore a broader input space, exposing edge cases and vulnerabilities through guided or random input generation [4].

To address the limited observability of internal model behavior, research has also introduced *structural coverage criteria* for Deep Learning models, such as *neuron coverage* [18], which tracks which parts of a neural network are activated by test inputs. Tools like DeepGauge extend classical testing concepts to deep learning by enabling finer-grained assessment of test completeness [4]. Finally, emerging research has begun to explore alternative approaches to ML testing techniques. These include the use of formal methods to verify model properties such as fairness or safety [33, 34], mutation testing adapted to neural networks to evaluate test suite effectiveness by injecting faults into models [35, 36], and runtime monitoring approaches that observe model behavior during execution to detect anomalies or distributional shifts [37, 38]. Additionally, some research efforts focus on embedding testing into MLOps pipelines, leveraging continuous integration/continuous deployment infrastructure to perform automated model validation during development cycles [39].

Our work contributes to the research areas by providing empirical evidence of how the ML testing techniques proposed are applied in real-world projects, grounding current research in practice. Additionally, we seek to expand the understanding of ML testing by uncovering additional practices and strategies that existing academic classifications may not fully capture.

2.2. Empirical Studies on ML Testing in Practice and Positioning of Our Work

Recognizing the gap between academic theory and industrial reality, recent research has begun to investigate ML testing practices empirically from different angles.

One line of research focuses on capturing the practitioners' perspective through surveys and interviews. Li et al. [40] surveyed 87 practitioners to uncover industrial concerns and good practices. Their findings highlight major challenges in test data collection, test execution, and result analysis, revealing a significant disconnect between the problems industry faces and the solutions proposed by academic research. Similarly, Song et al. [41] conducted an interactive rapid review with practitioners at Axis Communications, identifying key challenges in data testing and highlighting the need for context-specific solutions in industrial settings. A complementary line of research analyzes source code artifacts to understand the practices implemented. Openja et al. [42] performed an in-depth manual analysis of test files from 11 open-source projects. Using open coding, they derived the first empirically grounded taxonomy of ML testing strategies and properties, finding a predominance of grey-box and white-box techniques. Other studies have focused on more specific aspects, such as the adoption of MLOps and best practices [39, 43, 44].

While existing studies have begun to explore how ML testing is approached in practice, they focus on specific aspects and lack a comprehensive understanding of broader adoption. For example, Li et al. [40] investigated practitioners' experiences through interviews and surveys, but their study does not examine in detail the specific barriers that may hinder the adoption of testing techniques. Similarly, Openja et al. [42] conducted an in-depth manual analysis of a small number of open-source projects. Yet, the limited sample size and qualitative scope make it challenging to assess the prevalence of these practices.

Our work provides a broader and more integrated view of ML testing practices. By combining two complementary methods, i.e., a mining and a survey study, we aim to capture different dimensions of the phenomenon. The former examines what developers actually implement across hundreds of repositories, whereas the latter captures practitioners' perceptions,

motivations, and practical challenges. This enables us to construct a more comprehensive understanding of how ML testing is applied and experienced in practice, complementing and extending prior research with a broader evidence base.

3. Research Objectives and Research Method Overview

The *goal* of this study is to understand how ML testing is performed in practice, with the *purpose* of identifying both emerging strategies in real-world development and gaps that may hinder their adoption. The *perspective* is that of both researchers and practitioners. The former aim to understand current ML testing practices, with the ultimate objective of grounding future research contributions in empirical evidence. The latter seek actionable insights into practical tools and prevailing challenges, aiming to improve the quality, reliability, and efficiency of their testing processes in real-world development contexts. We structured our study around three main research questions, designed to capture what practitioners collectively do (implementation) and perceive (experience) regarding ML testing.

> ***RQ$_1$***. *What ML testing practices are prevalent across implementations and developer perceptions?*

This first research question allows us to map the state of ML testing and categorize testing activities by scope (e.g., data, model, system-level) and nature (e.g., unit, black-box), providing a structured understanding of what gets tested and at what level compared to what practitioners claim to know. Given the central role of tools in enabling or constraining testing practices, we then focused on the technological support around ML testing, asking:

> ***RQ$_2$***. *Which ML testing tools are used in code and perceived by practitioners?*

Our second research question complements **RQ$_1$** by providing a tooling perspective, highlighting which libraries and frameworks are actually used in source code, rather than those that practitioners are familiar with or prefer.

Finally, to fully assess the maturity and adoption of ML testing practices, we examined the challenges that affect the uptake and effectiveness of tools. Therefore, we asked:

> ***RQ$_3$***. *What limitations and barriers emerge across implementations and perceptions of ML testing?*

RQ$_3$ investigates whether technical constraints, observable in code, align with the practical barriers reported by developers. This provides a critical lens for identifying where current technical solutions and methodological proposals fail to meet developers' concrete needs.

From a methodological standpoint, we adopt a *mixed-method design* [22] that combines qualitative and quantitative analyses to triangulate insights from complementary data sources. As shown in Figure 1, the study begins with a *mining study* [45] that investigates the state of practice in open-source repositories. This phase focuses on identifying relevant test files, classifying testing strategies, and analyzing the adoption of specific tools. We then complement the findings with a *survey study* [46] targeting practitioners who develop and test ML-enabled systems. The survey captures perceptions, experiences, and self-reported practices, reinforcing and contextualizing the findings from the mining phase.

In terms of reporting, we follow the guidelines of Wohlin et al. [47] for designing and describing empirical software engineering studies. We also adhere to the principles of the *ACM/SIGSOFT Empirical Standards*,[1] specifically aligning with the *'Mixed-Methods Study'* standard.

4. Data Collection and Analysis

This section outlines the data collection and analysis methods used in the study. The detailed analyses are available in our online appendix [23].

4.1. Phase 1: Mining Study

This phase aims to build a large-scale, data-driven understanding of how ML testing is implemented in practice by analyzing open-source repositories. We analyze test files and project metadata to identify commonly used testing practices (**RQ$_1$**), the tools and frameworks adopted (**RQ$_2$**), and indirect signs of underuse that may indicate practical challenges (**RQ$_3$**).

4.1.1. Dataset Construction and Filtering

We leveraged *MARK*, a dataset recently proposed by De Martino et al. [48]. This contains 3,257 GitHub projects classified into four categories: *'ML Libraries & Toolkits'* (infrastructure and utilities for ML development), *'ML-Model Consumers'* (projects using pre-trained models), *'ML-Model Producers'* (projects training models from scratch), and *'ML-Model Producers & Consumers'* (projects that produce and integrate their own models). We selected *MARK* because it is specifically curated to distinguish among different types of ML-related development activities using fine-grained classification criteria. Additionally, it provides a reliable, practical foundation for studying ML testing practices as they naturally emerge in real-world software development.

Our investigation focuses on projects where developers actively build and train models, as these are most likely to involve model-centric testing. Consequently, we filtered the *MARK* dataset to include only *'ML-Model Producers'* and *'ML-Model Producers & Consumers'*, resulting in a subset of 1,852 projects. To ensure that the selected repositories were relevant, active, and representative of *mature and substantial* projects, we applied an additional filtering step using the criteria proposed by Widyasari et al. [49], designed to systematically exclude inactive, toy, or non-substantive repositories. To

[1] *ACM/SIGSOFT Empirical Standards*: `https://github.com/acmsigsoft/EmpiricalStandards`.

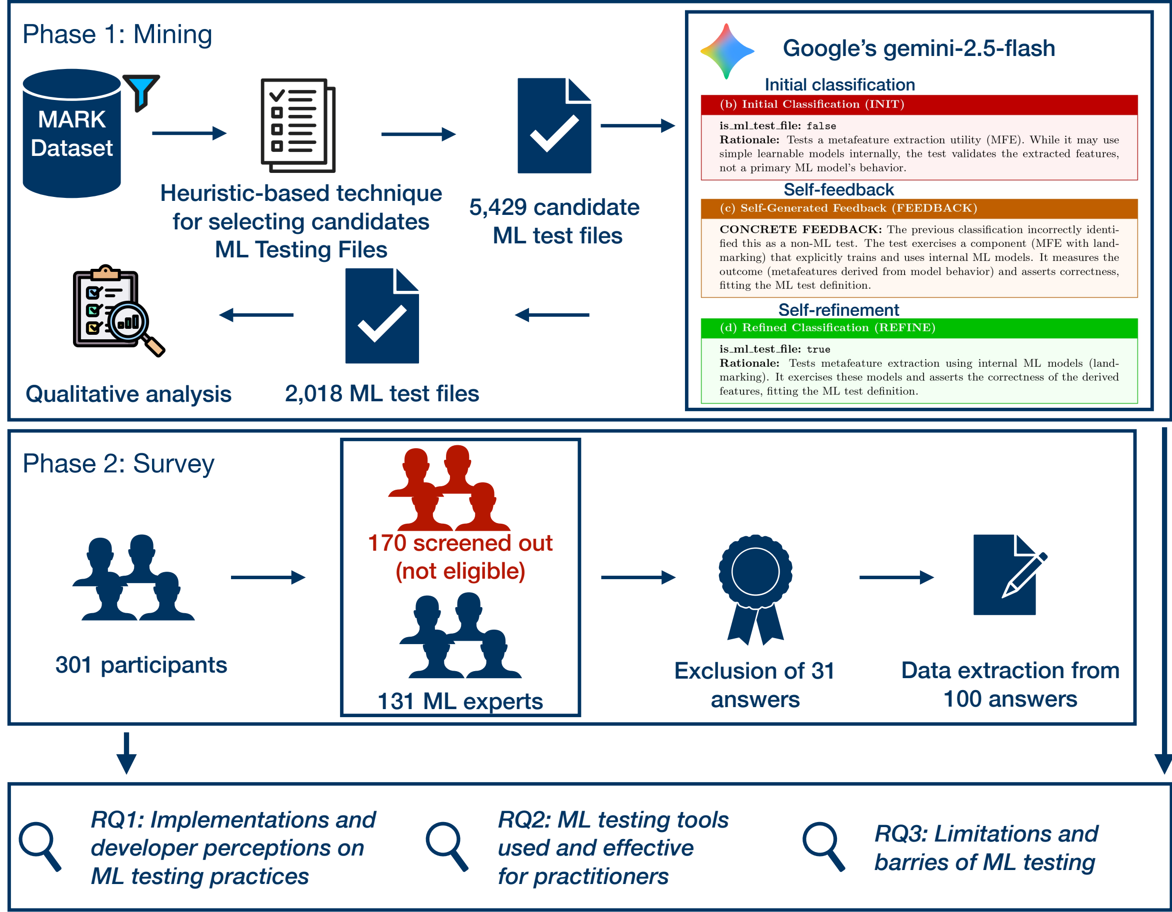


Figure 1: Overview of the two-phase mixed-method experimental design. Phase 1 (Mining) details the pipeline for identifying ML testing files, leveraging the MARK dataset, heuristic filtering, and an LLM-based classification with self-refinement. Phase 2 (Survey) depicts the practitioner recruitment process, including screening and data cleaning. The findings from both phases are triangulated to address the research questions.

this aim, we evaluated all $1,852$ repositories using the GitHub API. Each project was required to have at least 100 stars (as a proxy for community interest), at least 100 commits (to ensure development depth), and at least one commit after May 1, 2020. This date was selected as a baseline to ensure that the analyzed repositories were not abandoned projects, while still preserving a sufficiently large pool of candidate repositories for empirical analysis. This criterion does not imply that the selected projects were created within the last five years, but rather that they have been active at least once since May 2020. In other words, our dataset may include projects that were originally created well before 2020, but that have remained actively maintained in recent years. We also excluded repositories that were private or inaccessible at the time of analysis in June 2025. The decision to adopt validated filtering criteria rather than introduce arbitrary thresholds maintains consistency with established practices in prior empirical ML studies, ensuring that the selected repositories represent mature, actively maintained projects. Moreover, the purpose of these thresholds is not to restrict the dataset to recent projects, but rather to exclude clearly abandoned repositories. Given that our analysis focuses on current testing practices implemented in mature ML projects, repositories that have remained active after 2020 still provide meaningful insights into current development practices. After this step, we retained 398 repositories: 226 *'ML-Model Producers'* and 172 *'ML-Model Producers & Consumers'*.

4.1.2. Heuristic-based Identification of Candidate ML Testing Files

Given the large number of projects and the infeasibility of manual inspection, we developed a heuristic to narrow the search space and extract a relevant set of candidate files. Specifically, for each file, we assigned a relevance score and later used it as the basis for an additional filtering procedure. The process involved the following steps:

Location- and Name-based Filtering. The first stage builds an initial pool of candidate files by filtering based on typical project structures and test naming conventions. A file is considered a candidate if located in test directories (e.g., `tests`, `validation`) or matching naming patterns (e.g., `test_*.py`, `*_test.py`).

Content-based Analysis. Candidate files were analyzed for ML framework imports (TensorFlow, PyTorch, scikit-learn), testing constructs (`pytest`, `unittest`, `assert`), and evaluation keywords (*"accuracy"*, *"precision"*, *"loss"*).

Relevance Scoring. The goal of this scoring system was to prioritize files more likely to involve ML testing while filtering out irrelevant ones without requiring a full manual inspection. Each file received a score from 0 to 9, combining structural and content-based indicators. Up to 3 points were assigned based on structural heuristics: +1 point if the file was in a test-related directory (e.g., `tests/`), +1 if it appeared in an ML-related directory (e.g., `ml/`), and +1 if its filename contained the term `'test'`. Up to 6 additional points were assigned based on file content. Files received 1 point if they used a standard testing framework (e.g., `pytest`). To capture model-specific testing behavior, we further assigned up to 2 points for importing major ML libraries (e.g., TensorFlow, PyTorch) and up to 3 points for the presence of evaluation-related keywords (e.g., *"accuracy"*, *"precision"*). This weighting prioritizes content-based evidence, which more reliably captures ML testing behavior than structural patterns alone. This schema allowed us to obtain a subset of 16, 307 Python files with a non-zero score.

Refining for Precision and Recall. The relevance scoring mechanism was designed to maximize recall by capturing a broad set of Python files potentially related to ML testing. However, structural and content-based heuristics can also flag unrelated files, resulting in false positives. At the same time, we acknowledge the possibility of false negatives, i.e., relevant files that may have been missed because they did not follow common naming conventions, lacked typical directory structures, or used unconventional terminology. To ensure accuracy, we adopted a two-stage filtering strategy to further balance recall and precision, thereby retaining truly relevant files while minimizing noise in the final dataset. As such, we first identified a *recall-oriented threshold* to the heuristic scores to preserve as many true positives as possible; then, we refined the candidate set using a *high-precision LLM-based classifier*.

To determine the threshold, we manually validated a statistically significant sample of 376 candidate files (95% confidence level and 5% margin of error). The first two authors independently classified each file, resolving any disagreements through discussion. This process yielded a ground truth of 48 true ML test files within the sample. We then evaluated the performance of our heuristic at each score. The analysis shows that a threshold of `score` $\geq$ `4` yields a high recall of 87.5%, identifying 42 of the 48 true ML test files in our sample. While the precision at this stage was modest (33.3%), this reflects an intentional design choice driven by two factors: (1) the need to minimize false negatives (avoiding the premature exclusion of relevant test files), and (2) the necessity to provide the most comprehensive candidate set possible to the subsequent, more precise (but computationally expensive) LLM-based filtering step. Choosing a higher threshold (e.g., `score` $\geq$ `6`) would have discarded over a third of the relevant files, reducing recall to 62.5%. These factors were relevant to ensure that potentially relevant ML testing files were not discarded during the initial heuristic filtering stage, as the subsequent LLM-based classification step was specifically designed to refine the candidate set by removing false positives. In this sense, prioritizing recall at this stage therefore allowed us to preserve a broad set of candidates while delegating precision improvements to the later, more accurate classification phase. Applying the `score` $\geq$ `4` threshold, we retained 5, 429 candidate files for the next phase.

As for precision, we employed a state-of-the-art approach based on LLMs, leveraging the *SELF-REFINE* technique introduced by Madaan et al. [50], which iteratively refines LLM classifications using self-generated feedback. This approach allows the model to correct its own reasoning errors without requiring supervised fine-tuning, and its effectiveness for classification tasks in software engineering has been demonstrated in recent literature [51]. We implemented this process using `gemini-2.5-flash-preview-05-20`, for its cost-effectiveness and ability to process full source files without truncation. The process, illustrated with an example in Figure 2, consists of three steps. We first prompted the model with the full source code of each file. The system prompt was designed using a combination of role-play prompting [52] and few-shot learning [53]. We first assigned the model the persona of an *expert in ML-model testing*. Then, we provided a definition of an *ML-testing file* and a set of canonical examples of both positive (`ML_TEST`) and negative (`NON_ML_TEST`) cases. The model's task was to classify the file and provide its rationale in a structured JSON format. Then, this JSON classification was passed back to the same LLM, which was again guided by a role-play prompt to assume the persona of a *meticulous reviewer*. We asked to discuss its previous output and either respond with *VALID* if the classification was correct or provide concrete, actionable feedback for improvement. If feedback was generated, it was used as input for another prompt, instructing the LLM to produce a new, corrected JSON classification.

This loop was repeated up to three times or until the model returned `VALID` in the feedback stage, indicating that the classification had stabilized. In the example shown in Figure 2, the refined classification (d) was subsequently validated by the model, terminating the process.

To evaluate the effectiveness of the self-refinement process, we used a stratified sample of 126 files from our ground truth dataset with a `threshold` $\geq$ `4`. This sample, which reflects the actual distribution encountered during classification, included 42 true positives and 84 true negatives. We then compared the LLM final classifications against the manual annotations. The classifier achieved an overall accuracy of 81.7%, with a precision of 69.4% and a recall of 81.0% for the positive class, yielding an F1-score of 74.7%. Based on this performance, we deemed the classifier sufficiently accurate and applied the self-refinement process to the 5, 429 candidate files identified in the previous step. While some false positives or negatives may remain, the observed precision and recall provide strong confidence in the quality of the resulting dataset. Therefore, we proceed under the methodological assumption that the impact of residual noise is limited. Specifically, because our subsequent analysis seeks to identify aggregated, high-level trends across a large-scale dataset, the statistical impact of isolated individual misclassifications is inherently mitigated. We thus proceeded with the final set of 2, 018 files classified as `ML_TEST`, which we used for the final analysis phase.

(a) Source Code Snippet from `test_relative_landmarking.py`

```
import pytest
from pymfe.mfe import MFE
...
class TestRelativeLandmarking:
    def test_ft_method_relative(self, ...):
        X, y = load_xy(dt_id)
        mfe = MFE(groups=[GNAME], ...)
        mfe.fit(X.values, y.values, ...)
        _, vals = mfe.extract()
        assert np.allclose(vals, exp_value)
```

(b) Initial Classification (INIT)

is_ml_test_file: `false`
Rationale: Tests a metafeature extraction utility (MFE). While it may use simple learnable models internally, the test validates the extracted features, not a primary ML model's behavior.

(c) Self-Generated Feedback (FEEDBACK)

CONCRETE FEEDBACK: The previous classification incorrectly identified this as a non-ML test. The test exercises a component (MFE with landmarking) that explicitly trains and uses internal ML models. It measures the outcome (metafeatures derived from model behavior) and asserts correctness, fitting the ML test definition.

(d) Refined Classification (REFINE)

is_ml_test_file: `true`
Rationale: Tests metafeature extraction using internal ML models (landmarking). It exercises these models and asserts the correctness of the derived features, fitting the ML test definition.

Figure 2: An example of the SELF-REFINE classification process. The LLM initially misclassifies the file as a non-ML test (b), then generates feedback on its own mistake (c), and uses it to produce the correct, refined classification (d). A subsequent feedback step on this refined classification yielded a "VALID" response, terminating the iterative process.

4.1.3. Data Analysis

Upon selecting the final set of 2,018 files, we analyzed the content and associated metadata to address our research questions. We combined a top-down semantic analysis to uncover the high-level testing intent with a bottom-up structural analysis of the code to identify concrete tool usage patterns.

To investigate the types of tests and their intent, we conducted a qualitative analysis powered by LLM. For this complex reasoning task, we selected `gemini-2.5-pro` model (accessed on *2025-08*), a more powerful model than the one used for the initial binary classification, which enabled a more in-depth analysis. The choice of `gemini-2.5-pro` was motivated by both methodological and practical considerations. First, Gemini supports a substantially larger context window (up to 1M tokens) than other models (which typically support 128K–200K tokens), allowing us to process entire source files without splitting them into smaller chunks. This point is particularly important for our task, because understanding and determining which ML-specific testing approach is implemented in a file often requires reasoning over the full code context, whereas chunking could have led to loss of context and inconsistent classifications. Second, from a practical standpoint, `Google` provides initial research credits for new accounts, which enabled us to conduct the large-scale analysis cost-effectively while maintaining reproducibility and consistency in model usage.

We configured the model with a *temperature* of 0.2 and a *top_p* of 0.95 to balance creativity and determinism in structured generation tasks [54]. Each file was processed independently to prevent context contamination. We configured the model to generate a structured JSON output, enforcing a consistent response format. The core of our method was a detailed system prompt that instructed the LLM to act as a *testing expert* and follow a strict analysis protocol. In particular, the model was instructed to assign the most dominant testing strategy observed in each file. The taxonomy of strategies was defined in accordance with established academic literature. Specifically, we selected 15 testing methodologies as the basis for our classification: *Smoke*, *Metamorphic*, *Mutation*, *Black-box*, *Data-box*, *White-box*, *Domain-specific Test Cases*, *Input*, *Fuzz*, *Symbolic Execution*, *Data Linter*, *Adversarial Examples*, *Perturbed Model Validation*, *Rule-Based Checking*, and *Neuron Coverage-based Testing*. The selection began by reviewing the academic literature on ML testing (e.g., [4, 8, 15]) to identify commonly discussed techniques. Table 1 provides an overview of the operational definitions adopted in our classification process. To ensure coverage of practices not yet formally codi-

Table 1: Testing methodologies included in our study and their definition.

Testing Methodology	Definition
Smoke Testing	A shallow and broad test to ensure that critical system components execute without crashing and are minimally operational. It relies on sanity checks to confirm basic functionality is available, rather than verifying implementation correctness.
Metamorphic Testing	Verifies the model by checking expected relationships (metamorphic relations) between the outputs of multiple, transformed inputs. For instance, a slightly rotated image should still be classified the same.
Mutation Testing	Introduces small, artificial defects (mutants) into the model's source code, architecture, or training data to check if the test suite is effective enough to detect them. It primarily serves as a test adequacy criterion.
Black-box Testing	Testing conducted without any knowledge of the model's internal structure (architecture, weights). Analysis is based solely on the input-output relationship.
Data-box Testing	An intermediate approach where the tester has access to inputs, outputs, and the training/test data, but not the internal model architecture. Allows for analysis of the influence of the training data.
White-box Testing	Testing that leverages knowledge of the model's internal structure, such as its architecture, weights, or gradients. Neuron Coverage is a prime example.
Domain-specific Test Cases	Creating tests based on expert knowledge of the application domain, targeting specific, realistic scenarios or edge cases that are known to be challenging (e.g., testing an autonomous vehicle with a rare traffic scenario).
Input Testing	Focuses on analyzing the properties and quality of the input data itself (e.g., training, test sets) to identify issues like imbalance, distribution shifts, or corruption that could compromise model performance.
Fuzz Testing	Generates random or semi-random inputs (fuzzing) to find unexpected crashes, errors, or security vulnerabilities in the model or its data processing pipeline.
Symbolic Execution	Uses symbolic variables as inputs instead of concrete data to explore multiple execution paths simultaneously, aiming to verify properties or find inputs that satisfy specific conditions formally.
Data Linter	Employs automated, rule-based checks to inspect datasets for common issues like incorrect encoding, outliers, data duplication, or schema mismatches, much like a code linter checks source code.
Adversarial Testing	Generates maliciously crafted inputs (adversarial examples), often with imperceptible changes, designed to cause misclassifications, thereby testing its robustness.
Perturbed Model Validation	Tests model stability by injecting noise into the training data, retraining the model, and observing how its performance and behavior change in response to the perturbation.
Rule-Based Checking	Verifies a system against predefined rules or quality criteria. This can involve comparing outputs with a reference implementation ("golden master") to ensure consistency, or checking compliance with a checklist of readiness requirements (e.g., reproducibility, robustness, feature usefulness).
Neuron Coverage-based Testing	A white-box technique that measures the percentage of neurons activated by a test suite, to ensure that test inputs exercise a significant portion of the model's internal logic.

fied in the literature, we included a fallback category (`Other`) to capture emerging or unconventional testing approaches encountered in real-world code. This design choice reflects the exploratory nature of our study, acknowledging that developers may adopt novel strategies that evolve faster than academic taxonomies. It is important to emphasize that in our study, it was necessary to assign a single dominant testing strategy to each file to enable a large-scale quantitative analysis across repositories. Because testing strategies may overlap, we adopted a specificity-based classification rule, instructing the LLM to assign the most specific label that reflects the developer's testing intent. For example, tests implementing metamorphic relations are classified as *Metamorphic Testing*, even though they can also be interpreted as a form of black-box testing. This approach allows us to explicitly capture specialized ML testing techniques, which are central to our research questions. In contrast, the *Black-box Testing* category is used only for tests that verify system behavior through input–output observations without exhibiting the characteristics of more specialized strategies. In any case, it is worth remarking that this classification choice does not materially affect the interpretation of our results. The purpose of the rule is only to decide how a test should be labeled when multiple strategies could apply, not to change which tests are included in the analysis. Since our results are based on aggregate trends across a large set of test files, the overall conclusions depend on the relative prevalence of testing practices rather than on the classification of any individual file. The specificity-based rule simply ensures that specialized techniques (e.g., *Metamorphic Testing*) are explicitly identified rather than grouped under broader categories, such as black-box testing. This improves the interpretability of the results by making the presence of advanced techniques visible, without altering the overall patterns observed in the dataset.

The model was also instructed to take a conservative approach in uncertain cases, reporting a confidence level (high, medium, low) and using a dedicated label (`Not a Test/Unclassifiable`) when the file lacked sufficient evidence to support a reliable classification. The model was required to return a JSON object containing fields such as the *dominant testing strategy*, a confidence level, and structured *strategy evidence* (e.g., key functions, code patterns). These outputs directly support $\mathbf{RQ}_1$ by identifying the specific testing practices developers adopt, including their type and level (e.g., unit, integration), which can be inferred from the context.

To complement the semantic analysis, we conducted a bottom-up structural analysis to understand how developers implement ML tests. For each of the $2,018$ files, we parsed their Abstract Syntax Trees (ASTs) using Python's built-in `ast` module. We developed a static analyzer that systematically collects imported libraries and invoked functions by traversing the tree with dedicated visitor classes. This process allowed us to extract low-level evidence of tool usage and API calls within the test logic. To determine which tools to search for, we conducted a targeted preliminary landscape analysis to capture tools and frameworks widely discussed by practitioners but not yet reflected in academic publications. In particular, this preliminary review targeted practitioner-oriented blogs, technical reports, white papers, and online documentation. This process enabled us to identify five specialized tools (e.g., *Giskard*, *Deepchecks*) and five emerging testing topics. This comparison enabled us to determine which of these specialized tools are actually adopted in practice, reported in $\mathbf{RQ}_2$, and to identify gaps between community discussion and real-world usage, providing evidence for the challenges discussed in $\mathbf{RQ}_3$. We then cross-referenced the AST data against this list of tools.

4.1.4. Validation of the Strategy Analysis.

This classification is challenging because a single test file can often employ multiple strategies; choosing a single "dominant" strategy is subjective, even for a human expert. A traditional quantitative validation would require a large, manually-labeled gold standard. However, creating such an oracle is challenging and prone to low inter-rater agreement due to the inherent ambiguity of the task.

Therefore, we opted for a qualitative assessment to evaluate the plausibility and consistency of the LLM reasoning. The first two authors manually inspected a sample of 35 files classified by the LLM and compared the model's output with their own judgments. The goal was not to rigidly measure correctness, but to assess whether the chosen strategy and its rationale were logical and well-supported by the code.

The comparison revealed an exact match rate of 48.6% (17 out of 35 cases), confirming the task's inherent complexity. Analyzing the 18 disagreement cases, we found that they do not represent misclassifications by the LLM, but rather different yet plausible interpretations of the test file's primary intent. For instance, in the case of `test_soft_update.py` from the CATALYST project, our manual label was *White-box Testing* because the test directly asserts on the model's internal weights.

The LLM classified it as *Rule-Based Checking*, arguing that the primary goal was to verify that a component correctly manipulates the model's state according to a predefined mathematical rule. Importantly, *Rule-Based Checking* falls within the broader scope of *White-box Testing*, but the LLM's interpretation provides a more fine-grained and precise categorization. Moreover, in its extended rationale, the LLM acknowledged that *White-box Testing* remained a valid alternative perspective. This pattern—where the LLM recognized the human-preferred label as a higher-level category while offering a more detailed sub-classification—was recurrent across our analysis.

Given this high level of reasoning alignment with human expertise, even in cases of label disagreement, we concluded that the output was sufficiently reliable and consistent for our large-scale, exploratory analysis.

4.1.5. Data Extraction

From the outputs of the semantic and structural analyses, we systematically extracted quantitative metrics and qualitative samples to address our research questions.

Quantitative Data Extraction. To obtain a statistical overview of ML testing practices, we processed the structured outputs from our analysis pipeline as follows.

- **Testing Strategy Distribution.** From the JSON outputs of the LLM-based semantic analysis, we extracted the 'dominant_testing_strategy' field for each of the test files. We then computed the frequency distribution of these strategies to identify the most and least prevalent practices.

- **Tool and Library Usage.** From the AST-based structural analysis, we aggregated all unique `import` statements across the $2,018$ files. Our analysis proceeded on three levels. First, to measure the real-world impact of specialized tools, we computed the adoption frequency of the specific frameworks identified in our preliminary landscape analysis (more details are reported in Section 4.1.3). Second, to better understand the ecosystem of specialized tools, we calculated the overall frequency of all imported libraries and identified the most common dependencies in ML testing files. Finally, we conducted an exploratory analysis by manually inspecting the most frequent, unidentified libraries to discover any emerging tools not captured in our initial lists.

Qualitative Data Sampling for In-depth Analysis. To provide concrete examples and deeper insights into how testing strategies are implemented, we leveraged a manually inspected sample of 35 files used for validating our LLM-based analysis (as described in Section 4.1.3). For each of these files, we compiled a qualitative data record containing: (i) the source code, (ii) its project context, (iii) the LLM classification and rationale, and (iv) our own manual analysis notes. This set serves as the basis for the qualitative case studies presented later, allowing us to illustrate the quantitative findings with real-world code snippets.

4.2. Phase 2: Survey Study

This phase aims to complement our findings by gathering practitioners' perspectives on the adoption and challenges of ML testing in real-world settings. While the mining study offers valuable insights into actual testing behaviors and tool usage as reflected in code repositories, it provides limited visibility into the motivations, perceptions, and contextual factors that shape these practices. A survey enables us to explore the rationale behind observed behaviors, such as the choice of testing practices (**RQ**$_1$), the selection of tools and frameworks (**RQ**$_2$), and the perceived limitations and barriers to adoption (**RQ**$_3$).

4.2.1. Survey Recruitment and Dissemination

We followed the guidelines by Kitchenham and Pleeger [55] to design a survey that balanced the need to be short enough with the requirement to be effective in addressing our research questions. We provided participants with an informed consent document and deliberately avoided collecting personal information, such as gender, age, or email addresses. Our focus was solely on their experience with ML testing. Additionally, we incorporated attention checks to identify and later discard careless responses from participants.

Survey Design. We designed two questionnaires: a pre-screening survey that served as a preliminary screening tool to identify individuals matching our criteria, and our main survey.

The pre-screening survey comprised eight closed-ended questions designed to assess respondents' knowledge and experience with ML testing. Examples of these sentences include: *"In the last 6 months, I have worked with AI/ML models"* and *"I know the general concept of ML Testing, i.e., the systematic process of checking whether an ML system works as expected, both in terms of functionality and quality (e.g., reliability, fairness, robustness)"*. Participants with limited knowledge of ML testing were excluded from the main survey to ensure that responses were gathered from individuals with relevant expertise.

We intentionally adopted this broader definition because, in real-world ML development, practitioners may integrate model evaluation, data validation, and software testing practices within a single development pipeline [30, 39]. As a consequence, restricting the definition exclusively to traditional software testing activities would have therefore risked overlooking important verification practices commonly used in ML projects. Instead, our goal was to capture practitioners' perspectives on the full spectrum of verification activities applied to ML systems, including checks on data, models, and pipeline behavior. This approach is consistent with the broader perspective on ML testing adopted in recent literature on SE for AI and MLOps, where testing is often considered part of a wider validation and quality assurance process [39, 43].

Regarding the main survey, we first included a brief introductory section that introduced ourselves and the overall goals of the empirical study. We highlighted the growing importance of ML testing techniques alongside the challenges of applying them in practice, to inform participants of the required expertise and research objectives.

Furthermore, we provided information on the survey length, which was estimated at 10 minutes and subsequently empirically assessed in a pilot study (see details later in this section). We also explained that participation was voluntary, that participants could leave the survey at any time, and that all responses would remain anonymous to preserve privacy.

We concluded this part by asking for explicit consent to use the collected data for research purposes and authorization to proceed with the survey.

Table 2: List of questions for the background section in the survey with the type of response provided. The asterisk next to some questions indicates that an answer is required. Each question is mapped with the corresponding **RQs**.

Section 1: Participants' Background		Type
#1	How many years of professional experience do you have in software development?*	Multiple choice
#2	What is your highest level of education?*	Multiple choice
#3	What is your current role?*	Checkboxes
#4	What kind of organization are you currently working for?*	Multiple choice
#5	Do you have a background in Software Engineering?*	Multiple choice
#6	How familiar are you with general software testing practices (e.g., unit testing, integration testing, test automation)?*	5-point Likert scale
#7	How frequently do you write or maintain software tests (in any context)?*	5-point Likert scale
#8	How long have you been working with ML systems?*	Multiple choice
#9	What is your primary area of expertise related to AI?*	Checkboxes
#10	How familiar are you with the general concept of ML Testing, i.e., the systematic process of checking whether an ML system works as expected, both in terms of functionality and quality (e.g., reliability, fairness, robustness)?*	5-point Likert scale

Participant's background. The first section was related to the participants' background. This was required to (1) characterize the developers answering the survey and (2) understand whether and which responses could be removed because of the poor experience/expertise. As shown in Table 2, we inquired information on the years of software development experience, the current role they hold (e.g., Software Tester or QA Engineer), whether they mainly developed in open-source or other contexts, and their primary ML expertise. When answering these questions, they could select one or more options from a predefined list of the most common areas where ML techniques are used. Additionally, we asked whether they have a background in Software Engineering, their familiarity with software testing practices, and how frequently they write and maintain software tests. Finally, we asked about their familiarity with ML testing.

Table 3: List of questions related to the practical use of ML Testing, with the type of response provided. The asterisk next to some questions indicates that an answer is required. Each question is mapped with the corresponding **RQs**.

Section 2: ML Testing in Practice		Type	Mapping to RQs
#11	Indicate your level of familiarity and usage for each of the following ML testing techniques in your work:*	Multiple-choice grid	**RQ$_1$**
#12	Would you like to provide further information about the ML testing techniques you used?	Paragraph	**RQ$_1$**
#13	Indicate your level of familiarity and usage for each of the following ML testing tools/frameworks in your work:*	Multiple-choice grid	**RQ$_2$**
#14	In the previous question, you indicated your familiarity and usage of ML testing tools. If you rarely or never use specialized tools, what best explains why?	Check-boxes	**RQ$_2$**

Awareness and Practical Use of ML Testing. Participants then moved to the second section, which proposed questions concerned with the familiarity and practical use of techniques and tools for testing ML systems (Table 3). Each ML technique listed in the questionnaire was accompanied by a short description to ensure that participants had a consistent understanding of the concepts when reporting their familiarity and usage.

Regarding ML testing tools, to ensure completeness, we included `TextAttack` [56], the only specialized ML testing library identified through our AST code analysis (see Section 5.2). We combined this with the five specialized tools identified in our preliminary landscape analysis (as described in Section 4.1.3 and detailed in the technical report [23]).

For each strategy and tool, we used a 6-point scale designed to capture two dimensions: awareness and frequency of use. The first point on the scale, *"Not familiar"*, allowed participants to indicate a complete lack of awareness. The subsequent five points formed an ordinal scale of usage frequency, ranging from *"Never (know it, but never used it)"* to *"Always"*.

Table 4: List of questions related to the challenges in applying ML Testing, with the type of response provided. The asterisk next to some questions indicates that an answer is required.

Section 3: Challenges in applying ML Testing		Type	Mapping to RQs
#15	How much do you agree with each of the following statements regarding the main challenges in testing ML systems?*	Multiple-choice grid	**RQ$_3$**
#16	Are there other challenges or issues that you frequently encounter when testing ML systems that have not been mentioned above?	Paragraph	**RQ$_3$**

Challenges in applying ML testing. Afterward, the survey included a set of questions focusing on the main challenges practitioners face when testing ML systems. As shown in Table 4, we asked to indicate the extent to which each preliminary challenge identified in previous academic studies (e.g., [4, 8]) represents an actual challenge from their own perspective. We also included a free-text answer to elaborate and provide information on additional challenges encountered in their experience.

Survey closure. Before thanking the participants, we allowed them to enter their email addresses to (1) receive a summary of our results and/or (2) participate in a future follow-up study.

Survey Validation. Before disseminating the survey, we evaluated its quality and the time required to complete it. We conducted a pilot study [57], consisting of an experiment with trusted participants who provided feedback on the length, clarity, and structure of the survey. The pilot study was conducted with two software engineers and ML experts with experience in software development and testing, who were selected through purposive sampling within our contact network. We directly contacted these two experts because they share the profile of our target population (with extensive experience in both SE and ML) and, crucially, have prior experience designing and conducting empirical survey studies. This background allowed them to evaluate the questionnaire both on technical grounds and in terms of survey flow, clarity, and potential biases.

The first author contacted them via email, providing instructions on completing the survey. The experts were also provided with a document in which they could report their observations on the length, clarity, and structure of the survey. We also asked them to clock their progress to have a precise estimate of the time required to complete the survey. They had one week to complete the task. The two pilots sent their notes back upon completion. These were jointly analyzed by the first two authors, who identified and addressed clarity issues in the various sections. Additionally, the experts suggested integrating back-

ground information by including details on participants' experience with traditional software testing and their expertise in software engineering. In terms of timing, the pilots completed the survey within 10 minutes. We finally double-checked our changes with the pilots, who confirmed that all of their feedback had been addressed.

4.2.2. Survey Recruitment and Dissemination

A critical aspect of survey design is selecting an appropriate and relevant target population [58]. We deliberately avoided disseminating our survey via social networks because we could not control the responses we received [58]. We used PROLIFIC[2], a research-oriented web-based platform to facilitate participant recruitment for scientific studies.

We adhered to established best practices for using PROLIFIC, following the set of recommendations provided in the literature [59–64]. In particular:

- We used a pre-screened method to enhance the selection of participants.
- We included validation questions to assess participants' ML testing expertise.
- The survey incorporated attention checks to identify and exclude participants who provided careless responses. These checks were embedded within longer question matrices to ensure participants remained engaged. For instance, we asked about familiarity with different ML testing techniques, with one question explicitly stating: *"For this question only, please select Never to show that you are paying attention"*.
- We employed a layered payment plan, offering a £8 ("good") reward for pre-screening and a £9 ("good") reward for completing the main survey.[3]
- We used a tool to detect AI-generated responses (specifically to open-ended questions) and identify any inappropriate behavior among participants.[4]

Participants were selected based on criteria related to their ML expertise. Those who indicated *'never worked with ML'*, or '*demonstrated no knowledge of ML testing'*, as per their responses in the pre-screening questionnaire, were excluded from the study. Moreover, PROLIFIC provides historical metadata for each participant, including the number of completed surveys and the participant's approval rate. To ensure reliability, we restricted participation to individuals who had completed at least 50 previous tasks and maintained an approval rate of 100%. This strategy increased our confidence that we had selected individuals with relevant experience, credibility, and interest in the subject matter.

Ethical Considerations. Before disseminating the survey, we requested approval from the Ethical Review Board of the University of Salerno. We followed responsible research practices to address potential ethical and privacy concerns [65]. The survey was designed to protect participants' anonymity, as no personally identifiable information was collected unless participants voluntarily provided it. When recruiting ML experts, we clearly communicated the study's purpose. We specified that the collected responses would be used solely for academic research, with no intention of disclosing sensitive data. Finally, we informed participants that aggregated results may be made public, but individual privacy would be strictly preserved.

Survey Dissemination. On August 27, 2025, the survey dissemination began with 331 participants recruited. 170 participants were screened out due to a lack of knowledge in ML testing and experience with ML systems. The remaining 131 participants completed the entire survey, and their answers were subject to quality assessment.

4.2.3. Data Cleaning and Analysis

Once we had collected the responses, we conducted a quality assessment. In particular, the first author reviewed the individual responses to validate them, possibly identifying cases in which participants failed the attention check or lacked sufficient experience to provide valuable insights. This procedure resulted in the exclusion of 31 responses: 21 failed one or more attention checks, and 10 were removed for providing clearly incomplete or inconsistent answers. In the other responses, we first interpret the data by analyzing the closed answers. Most questions were formulated so that participants could express their opinions using check-boxes or a Likert scale [66] with different response ranges. These answers were analyzed using statistical methods, with numerical distributions collected and presented in bar plots.

The open answers were subject to content analysis [67], a research method in which one or more inspectors review the data of interest and attempt to deduce their meaning and/or the concepts they elicit. The content analysis was conducted by applying a two-phase process. In particular we first, we performed an iterative codebook development phase. In this phase, the first two authors independently reviewed an initial subset of 25 open-ended responses, selected from the total of 80 open-ended responses to identify recurring concepts and themes related to ML testing practices and challenges. The authors then compared their observations and jointly defined an initial set of codes capturing the main ideas emerging from the responses. As a result of this phase, the authors established a preliminary codebook comprising 14 thematic codes and their corresponding descriptions, which served as the basis for the subsequent systematic coding of the remaining responses.

After the codebook development phase, the systematic coding phase was conducted. Using the agreed codebook, the first author applied the codes to the full set of responses, assigning one or more codes to each answer where appropriate. Throughout this phase, the second author remained closely involved in the process, with the two authors meeting weekly to review coding progress, discuss emerging themes, and refine the codebook

[2]`https://www.prolific.com/`

[3]PROLIFIC rates compensation in four levels: low, fair, good, and great.

[4]ZeroGPT: `https://www.zerogpt.com` (August 2025)

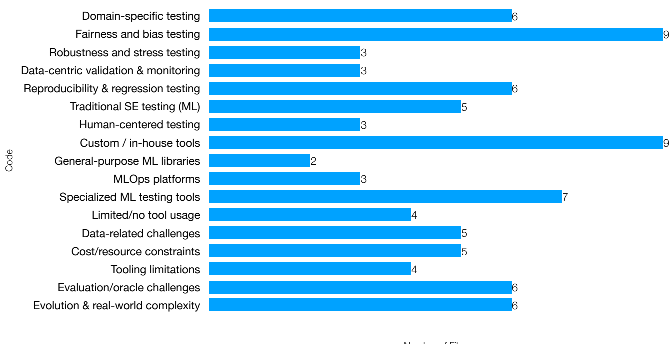


Figure 3: Overview of all codes identified during the content analysis of open-ended questions and their corresponding distributions.

as needed. During this process, the initial codebook was iteratively refined. In particular, three additional themes emerged during the systematic coding of the remaining responses. This iterative process required two rounds of refinement before no further relevant themes emerged, indicating that theoretical saturation had been reached. Whenever ambiguous or difficult-to-interpret responses were encountered, the first two authors jointly discussed them to reach consensus on the most appropriate interpretation and coding.

The entire coding process was deliberately performed manually (rather than using specialized qualitative analysis software), as this approach enabled us to more carefully interpret the domain-specific insights from practitioners in the ML testing context. Figure 3 shows all the codes detected during the process, along with their distributions.

5. Analysis of the Results

In the following, we first provide an overview of the two data sources that underpin our analysis. We then present the synthesized results, organized per research question.

Overview of the mining study. The dataset comprises 398 projects (2,018 test files) with significant heterogeneity across the ML landscape. The projects cover a wide spectrum of popularity and community engagement, ranging from niche tools to major frameworks (max 80 596 stars, max 34 592 forks), with a median of 752 stars and 142 forks. Development activity is similarly diverse, spanning from the minimum threshold of 100 to over 97 000 commits (median 447.5), driven by teams ranging from single developers to large collaborations (max 453 contributors; median 9). Codebases range in scale from median 158.5 files and 13 396 LOC to over 52 000 files and 7M LOC.

To provide more context, we further characterized the 146 unique repositories associated with the 2,018 test files in terms of underlying ML techniques and application domains. To identify the specific ML/DL techniques involved, we implemented an AST-based static analysis script that parses the Python test files and inspects import statements, class instantiations, and API calls associated with common ML libraries (e.g., TensorFlow, PyTorch, scikit-learn). Since individual test files often instantiate multiple algorithms concurrently, our file-level analysis identified a total of 4,004 distinct algorithm occurrences, which we refer to as *traces*, across the 2,018 test files. Based on these traces, we observe a strong presence of Deep Learning architectures, appearing in 71.2% of the projects. Among these, Deep Neural Networks (DNNs) are the most common (54.8%), followed by Convolutional Neural Networks (33.6%), Recurrent Neural Networks (20.5%), and Transformers (12.3%). Concurrently, classical ML algorithms appear in 40.4% of the projects, with tree-based models like Random Forests (18.5%) and Gradient Boosting (10.3%), alongside Linear Models (17.1%) and Support Vector Machines (10.3%).

Regarding the application context, our metadata analysis using GitHub Topics reveals that general-purpose Data Science and ML Frameworks are the most prevalent (accounting for 47.3% of projects). When looking at specific application domains, Natural Language Processing (15.8%) and Computer Vision (13.7%) are the most represented fields. We also highlight the presence of highly specialized and critical domains, including Bioinformatics and Drug Discovery (3.4%), Medical Imaging (3.4%), and Autonomous Driving (0.7%). This confirms that our dataset captures a diverse and comprehensive cross-section of the real-world ML landscape.

Table 5: Demographics and Experience of the 100 Survey Participants.

Category	Attribute	Count
SW Dev. Experience	10+ years	39%
	6–9 years	20%
	3–5 years	27%
	0–2 years	14%
ML Systems Experience	10+ years	2%
	6–9 years	7%
	3–5 years	44%
	0–2 years	47%
Role (Top 3)	Developer	50%
	Software Tester/QA	30%
	Project Manager	29%
Organization Type	Industry	64%
	Startup	15%
	Freelancer/Consultant	8%
ML Testing Familiarity	Familiar or Very Familiar	72%
	Somewhat Familiar	28%

Overview of the survey study. Table 5 reports demographics for the 100 surveyed practitioners, most hold senior roles (59% with 5+ years experience).[5] The most common roles are Developer (50%), Software Tester/QA Engineer (30%), and Project Manager (29%). The sample is highly educated, with 87% holding a Bachelor's degree or higher, and most participants (67%) have a formal background in Software Engineering.

Participants are actively involved in ML testing: 90% write tests weekly, and 53% have 3+ years ML/AI experience. Their

[5] Our survey involves exactly 100 participants. As such, the percentage values reported in the survey results can be interpreted directly as the corresponding numbers of participants.

expertise spans various domains, with Natural Language Processing (50%) and Autonomous Systems (37%) being the most common, and only a small minority (15%) reporting no specific ML expertise.

Regarding the socio-demographic characteristics, the participants represent a diverse geographic distribution, predominantly residing in the United States (34%), the United Kingdom (32%), and Canada (10%), with the remainder distributed across various European countries (e.g., Germany, Spain, Italy). The average age of the respondents is 37 years (ranging from 19 to 65). In terms of gender, 79% identify as male, 20% as female, and 1% preferred not to disclose. Ethnically, the sample comprises White (61%), Asian (15%), Black (13%), and Mixed or Other (10%) respondents. Finally, the vast majority of the participants (87%) are employed full-time.

5.1. *RQ_1: What ML testing practices are prevalent across implementations and developer perceptions?*

The analysis of the 2,018 test files provides a quantitative overview of the strategies implemented by practitioners. Figure 4 shows the distribution of the dominant strategy identified: we reported only the techniques for which we found concrete evidence in the analyzed test files. Specifically, while our prompt to classify ML techniques included *Symbolic Execution*, *Perturbed Model Validation*, and *Neuron Coverage-based Testing*, no instances of these techniques were identified in the analyzed repositories. The results reveal that few practices dominate the ML testing. The most prevalent strategy is *Smoke Testing*, identified in 933 files (46.2%). This suggests that the primary concern for practitioners is ensuring that the models and data pipelines run without critical errors. Other adopted strategies are *Rule-Based Checking* (369 files, 18.3%), *Black-box Testing* (210 files, 10.4%), and *Domain-specific Test Cases* (174 files, 8.6%). Moreover, advanced techniques discussed in the academic literature are less widely adopted. Specifically, the lower adoption rate of *Metamorphic Testing* (identified in 67 files, 3.3%) compared to foundational practice, stands out given its prominence in academic research. Existing systematic literature reviews rank it as the most studied solution for the ML oracle problem [4, 8], yet this does not appear to translate into widespread industrial use. Other specialized strategies like *Adversarial Testing* and *Fuzz Testing* are extremely rare (identified in one and four files, respectively). This quantitative analysis reveals a gap between the techniques proposed by research and those actually implemented in practitioners' projects.

Furthermore, we explored the relationship between testing strategies and the underlying ML techniques or application domains. Our analysis shows that fundamental techniques such as *Smoke Testing* and *Rule-Based Checking* are largely domain-agnostic, accounting for over 70% of the identified testing traces (2,952 out of 4,004 traces) and appearing across all algorithmic families. Looking at individual techniques, more advanced approaches show stronger associations with specific ML architectures. *White-box Testing* and *Adversarial Testing*, for instance, appear predominantly or exclusively in Deep Learning projects: over 66% of *White-box Testing* traces and 100%

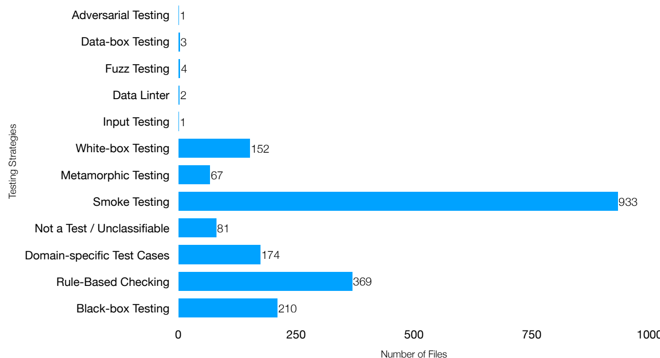


Figure 4: Distribution of the dominant testing strategies identified across the 2,018 test files. We reported only the techniques for which we found concrete evidence in the analyzed test files.

of *Adversarial Testing* traces map directly to neural architectures (e.g., DNNs, CNNs, Transformers). *Metamorphic Testing* also exhibits a relative preference for Deep Learning (which accounts for nearly 45% of its traces), while maintaining a presence in classical models. Conversely, *Data-box Testing* is observed exclusively in classical ML pipelines, with 100% of its traces mapping to traditional algorithms such as Linear Models, Gradient Boosting, and Decision Trees. These results demonstrate that the adoption of advanced testing techniques remains strongly dictated by the underlying ML architecture and development context.

```
# File: idealo_image-super-resolution/.../test_models.py
↪   (Model Unit Testing)
def test_that_the_trainable_layers_change(self):
    # ... setup model and random data x, y ...
    before_step = []
    for layer in self.RRDN.model.layers:
        if len(layer.trainable_weights) > 0:
            before_step.append(layer.get_weights()[0])

    self.RRDN.model.train_on_batch(x, y)

    i = 0
    for layer in self.RRDN.model.layers:
        if len(layer.trainable_weights) > 0:
            self.assertFalse(np.all(before_step[i] ==
            ↪   layer.get_weights()[0]))
            i += 1
```

Listing 1: An example of Model Unit Testing, where the test verifies that a model's internal weights change after a single training step.

To provide a deeper understanding of how these strategies are implemented, we qualitatively analyzed the *35 manually inspected files* (see Section 4.1.4). We observe that project context influences strategy choice: the example illustrated in Figure 5 shows that the high prevalence of *Smoke Testing* often reflects a pragmatic need to ensure the basic operational integrity of complex models. A typical implementation (Figure 5(a)) involves fitting a model on a small synthetic dataset and asserting that the core API (e.g., `evaluate`) is functional. More sophisticated strategies are often tied to specific domains. For instance, in a causal inference library, we observed *Data-box Testing*,

(a) Smoke Testing in *DeepChem*

```
# File: deepchem_deepchem/.../test_graph_conv.py
def test_graph_conv_model():
    tasks, dataset, _, metric = get_dataset(...)
    model = GraphConvModel(len(tasks), mode='classification')
    model.fit(dataset, nb_epoch=20)
    scores = model.evaluate(dataset, [metric])
    # Assert that the model can overfit a small dataset
    assert scores['mean-roc_auc_score'] >= 0.9
```

(b) Data-box Testing in *CausalML*

```
# File: uber_causalml/.../test_meta_learners.py
def test_BaseSLearner(generate_regression_data):
    # Synthetic data with known ground truth for the causal effect (tau)
    y, X, treatment, tau, _, _ = generate_regression_data()
    learner = BaseSLearner(learner=LinearRegression())
    ate_p, _, _ = learner.estimate_ate(X=X, treatment=treatment, y=y)
    # Assert that the model's estimated effect is close to the true effect
    assert ape(tau.mean(), ate_p) < ERROR_THRESHOLD
```

(c) Metamorphic Testing in *GluonNLP*

```
# File: dmlc_gluon-nlp/.../test_models_electra.py
# Original input
contextual_embedding, _ = electra_model(inputs, ...)
# Transformed input (transposed)
contextual_embedding_tn, _ = electra_model_tn(inputs.T, ...)
# Metamorphic relation: assert inverted output equals original output
assert_allclose(contextual_embedding.asnumpy(),
np.swapaxes(contextual_embedding_tn.asnumpy(), 0, 1))
```

Figure 5: Examples of implemented testing strategies from our qualitative analysis.

where the estimated effect of a model is asserted against a ground truth embedded in the synthetic data (Figure 5(b)). Finally, our analysis confirms that advanced techniques, such as *Metamorphic Testing*, although rare, are implemented in mature projects. As shown in Figure 5(c), the developers of an NLP library manually implemented a metamorphic relation to verify model consistency with respect to input layout, showcasing a sophisticated, albeit custom-built, approach to solving the oracle problem.

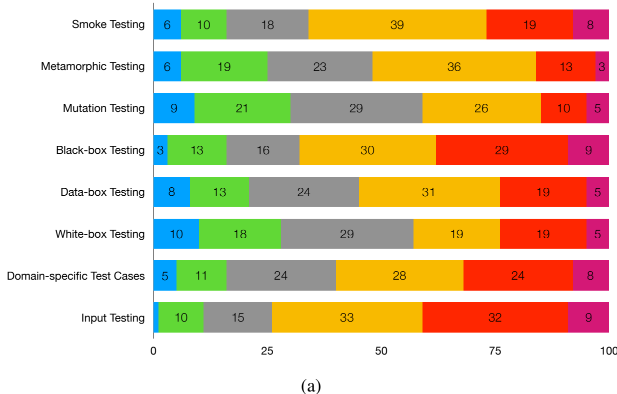
```
# File: namisan_mt-dnn/int_test_encoder.py (Integration
↪ Testing)
def test_encoder(src_dir, checkpoint_path, ...):
    # Execute the entire training/encoding script as a
    ↪ subprocess
    subprocess.call(cmd, shell=True,
    ↪ stdout=subprocess.DEVNULL)

    # Load the output artifact generated by the script
    encoding_0 = torch.load(os.path.join(target_dir,
    ↪ r"cola_encoding.pt"))
    # Load the expected "golden" artifact
    encoding_1 = torch.load(os.path.join(expected_dir,
    ↪ r"cola_encoding.pt"))

    # Assert that the generated artifact matches the golden
    ↪ reference
    assert masked_array.mean() < 1e-4
```

Listing 2: An example of Integration Testing, where an entire training script is executed, and its final output artifact is validated.

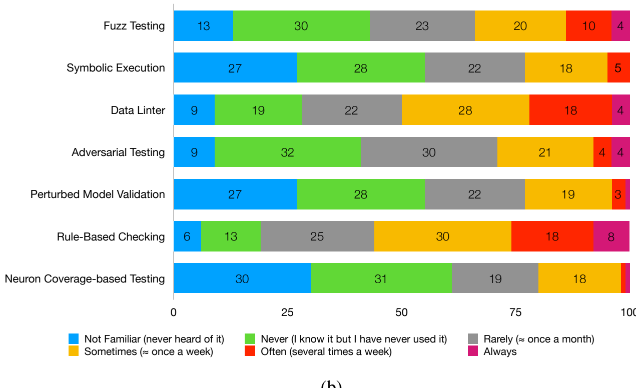
The qualitative analysis reveals how different testing levels are adapted to the ML context. We observed a clear distinction between unit-level tests, which focus on a model's internal mechanisms, and integration-level tests, which validate models within a larger pipeline. As shown in the snippet from the `idealo_image-super-resolution` project (Listing 1), developers test a model component in isolation. The test verifies a core property of the training process, i.e., the model's weights are updated after a single call to `train_on_batch`. This is a classic unit test, but instead of testing a simple function, it tests a fundamental behavior of a learnable component.

In contrast, we also found examples of *Integration Testing*, in which the model is treated as a component within a larger workflow. In the `namisan_mt-dnn` project, the test invokes an entire training script via a subprocess (Listing 2). The test then validates the final artifact produced by this script, an embedding file saved to disk, by comparing it against a "golden" reference file. This validates the integration of the training pipeline rather than isolated model behavior.

Moving to the survey, the results reveal both high familiarity with and frequent use of foundational techniques. Figure 6 shows a high familiarity with fundamental concepts like *Input Testing* (99%), *Black-box Testing* (97%), and *Smoke Testing* (94%). These techniques also report the highest rates of adoption, with 89%, 84%, and 84% of participants, respectively, suggesting their use at least once a month. The survey also highlights a significant gap between awareness and adoption for more advanced techniques. While *Metamorphic Testing* is well-known (94% familiarity), its adoption is lower (75%), with many practitioners reporting low-frequency use (e.g., rarely or sometimes) rather than systematic or regular adoption. This



(a)



(b)

Figure 6: Familiarity and usage frequency of different ML testing techniques as reported by 100 practitioners.

gap becomes particularly evident with specialized testing methods. *Adversarial Testing*, despite being familiar to 91% of participants, shows an adoption rate of only 59%, highlighting a substantial implementation barrier. Other advanced techniques, such as *Symbolic Execution*, *Perturbed Model Validation*, and *Neuron Coverage-based Testing*, remain unfamiliar to 27%, 27%, and 30% of participants, respectively, suggesting these approaches have yet to gain widespread recognition among practitioners.

The open-ended responses provided further insights into practices not fully captured by our initial taxonomy. A key theme was the emergence of *domain-specific testing* techniques. One participant from a scientific modeling background reported the use of *"physics-law consistency checks"* (participant #84). At the same time, another from the financial sector mentioned *"backtesting"* (participant #21) as their primary validation method. For generative models, participant #46 described a behavioral check called *"reasonable variability tests"*, in which the semantic distance between generated answers is measured to ensure prompt stability.

Another prominent practice explicitly mentioned by multiple participants was *Fairness and Bias Testing*. Participants describe its use for *"quantitatively and qualitatively assessing outputs for biases across protected attributes"* and *"quantify fairness metrics like demographic parity"* (participant #56). This suggests that for practitioners, responsible ML is not just a theoretical concern but a concrete testing requirement. Participants also highlighted the use of established software engineering practices adapted to the ML context, such as *Regression Testing* (participant #96) to check for performance degradation after updates and drift monitoring in production.

Key Findings for RQ_1

Our analysis reveals a hierarchy of ML testing practices spanning multiple levels of the development stack. The adaptations of *Smoke Testing* and *Black-box Testing*, dominate both implemented code and practitioner-reported practices, forming an implicit *multi-level strategy*. The mining study identifies fine-grained *Model Unit Testing* and broader pipeline *Integration Testing*, while the survey confirms that testing effort is distributed across data, model, and system levels. In contrast, advanced techniques exhibit a clear gap between theory and practice: despite widespread discussion, *Metamorphic Testing* shows limited adoption, with practitioners often implementing custom solutions. Finally, the survey reveals an expanding focus beyond functional correctness, emphasizing *domain-specific approaches* and growing attention to *Fairness and Bias Testing*.

5.2. *RQ_2: Which ML testing tools are used in code and perceived by practitioners?*

Our analysis of library imports provides a practice-oriented perspective. The results reveal a tooling landscape characterized by two key findings: reliance on a small set of core libraries and limited adoption of specialized ML testing tools. The ecosystem is dominated by general-purpose and core ML libraries. *NumPy* represents the most common dependency (18 058 imports, 28.9%), followed by deep learning frameworks like `torch` (15.9%) and `tensorflow` (9.8%). Standard testing frameworks like *PyTest* (10.3%) and *unittest* are the primary drivers for test execution, alongside MLOps utilities such as `mlflow` (9.0%).

In contrast, the specialized tools identified in our preliminary landscape analysis [23] show lower evidence of direct adoption by a broad base of developers. Prominent tools such as *Deepchecks* and *Giskard* were not detected, and the *Adversarial Robustness Toolbox (ART)* was also not among the frequently imported libraries. Additionally, some tools identified in the preliminary analysis, such as the *MATLAB Deep Learning Toolbox*, could not be tracked in our Python-focused AST analysis as they are standalone environments rather than importable libraries. *LangSmith* appears as an exception, with 244 imports detected. However, a deeper investigation revealed that these imports originate from a single, large project, i.e., `MLflow`. This pattern reflects not widespread adoption across diverse practitioners, but a concentrated integration within a single MLOps framework. Furthermore, our analysis revealed

the use of *TextAttack*, another specialized library for adversarial attacks, which was imported 43 times. Unlike the other tools derived from our preliminary analysis, *TextAttack* emerged solely from the data-driven AST analysis, highlighting the value of the mining approach in uncovering specific adoptions. These findings reinforce our main conclusion: a critical adoption gap exists for specialized ML testing tools, which are not yet part of the standard toolkit for most developers.

The qualitative analysis of the test files provides a compelling explanation for the observed adoption gap. We found that even when practitioners adopt sophisticated testing strategies, they tend to implement them using general-purpose libraries rather than adopting specialized tools. A concrete example is the implementation of *Metamorphic Testing* in the `GluonNLP` project (Figure 5(c)). Specifically, practitioners implemented a non-trivial metamorphic relation, verifying model invariance to data layout transformations, using only foundational libraries like *NumPy* for data manipulation (`np.swapaxes`) and assertions. Despite the existence of dedicated frameworks, such as *Giskard*, we did not find evidence of its practical use.

Moving to the survey results, we can further confirm the gap in the adoption of specialized ML testing tools and provide insights into practitioners' awareness and preferences. The data shows a significant lack of familiarity with many of the specialized tools included in our investigation. For instance, only 33% of participants are familiar with *Giskard*, and 52% of those who are aware reported never using it. Similar results can be observed for *Adversarial Robustness Toolbox (ART)*: 29% are aware of it, and 52% have never used it.

The open-ended responses reinforce these findings and provide insight into their preferences. A dominant theme was the reliance on custom and in-house solutions. Multiple answers stated they use their *"own custom framework"* (participant #22) or have *"wrote our own model-specific testing tools"* (participant #44), with one explicitly mentioning this is done *"to avoid uncontrollable dependencies"* (participant #46).

Furthermore, when external tools are adopted, practitioners seem to prefer comprehensive *MLOps platforms* over standalone testing libraries. Participants reported using tools such as `MLflow` and `TensorFlow Extended (TFX)` (participants #58 and #99) for model validation and monitoring. One participant highlights the use of `Evidently AI`[6] to *"track dataset shifts"*. This suggests that practitioners prefer integrated platforms to cover the entire ML lifecycle. Finally, the open-ended responses also allowed us to identify several other specialized tools used by a small number of participants, such as *Kolena*[7] (participant #67), *InterpretML*[8] (participant #67), *RobustBench*[9] (participant #56), *Parasoft*[10] (participant #72), and *Testim*[11] (participant #72), confirming the fragmented nature of the current tooling ecosystem.

[6]`https://www.evidentlyai.com/`
[7]`https://www.kolena.com/`
[8]`https://interpret.ml/`
[9]`https://robustbench.github.io/`
[10]`https://www.parasoft.com/`
[11]`https://www.testim.io/`

Key Findings for RQ_2

Results reveal a critical, multifaceted gap in adopting specialized ML testing tools. Our mining study shows that specialized tools are almost entirely absent in practice, with developers relying instead on foundational libraries such as `NumPy` and `PyTest`. Survey responses provide a possible explanation for this gap, highlighting a strong preference for custom solutions (40% of participants). Moreover, open-ended responses indicate a growing preference for integrated *MLOps platforms* such as `MLflow` over standalone testing libraries, as practitioners favor tools that cover the entire ML lifecycle rather than focusing on a single testing type.

5.3. *RQ_3: What limitations and barriers emerge across implementations and perceptions of ML testing?*

Although the mining study does not explicitly document testing challenges, our analysis uncovers three recurring patterns that reveal practical limitations. The statistical dominance of *Smoke Testing* (46.2% of files) indicates a widespread focus on verifying basic operational integrity (e.g., ensuring a model runs without crashing) over more complex behavioral validation. The qualitative analysis uncovers a common pattern of enforcing deterministic outcomes. For instance, in `namisan_mt-dnn/int_test_encoder.py` (Listing 2), developers compare model outputs to pre-computed "golden" files, a technique that circumvents the difficulty of testing non-deterministic systems by creating highly constrained, predictable scenarios. Finally, we observed a consistent pattern of manual implementation for complex tests. The near-total absence of specialized ML testing libraries, combined with evidence of manual implementation of advanced strategies like *Metamorphic Testing* using only general-purpose libraries (as seen in Figure 5(c)), points to a significant limitation in the practical applicability of the current tools.

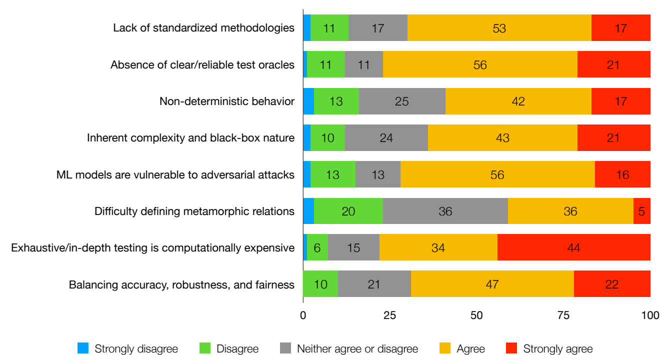


Figure 7: Practitioner agreement with statements about key challenges in ML testing. The chart shows the distribution of responses on a 5-point Likert scale for N=100 participants.

Regarding the survey study, Figure 7 indicates a strong agreement on several fundamental issues. 78% of participants agree

or strongly agree that exhaustive testing is computationally expensive. In the open-ended responses, participants also highlighted *"the human cost: developer-hours spent architecting, developing, integrating, and evaluating testing"* (participant #44). The second most cited challenge is the absence of clear test oracles (77%). The open-ended responses further emphasized the related issue of data, suggesting the *"scarcity of high-quality reference data"* (participant #84) and the problem of *"production data drifts away from training data over time"* (participant #99).

From a technical perspective, the survey highlights as challenges the vulnerability to adversarial attacks (72%) and the difficulty of balancing trade-offs among accuracy, robustness, and fairness (69%). Interestingly, although the non-deterministic behavior of models is acknowledged as a problem, it received the lowest level of agreement among the core technical challenges (59%). The open-ended responses also revealed a significant organizational and communication challenge, which is not purely technical. One participant noted the difficulty of *"Explaining to POs or other devs that our testing is not as deterministic as theirs"* (participant #46). At the same time, another described the *"Pressure from stakeholders and expecting a clear pass/fail result"* (participant #99).

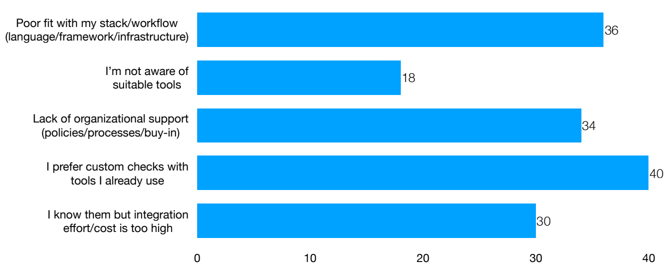


Figure 8: Primary barriers to the adoption of specialized ML testing tools, as reported by practitioners.

To better understand the root causes of the previously identified tool adoption gap, we asked about the main barriers they faced during ML testing development. As shown in Figure 8, the most cited barrier is a preference for custom checks with existing tools (40%), which directly corroborates the custom implementation pattern we identified in the mining study. This is closely followed by practical integration challenges, such as a poor fit with the current stack/workflow (36%) and a perceived high integration effort or cost (30%). These integration barriers become clearer when situated within the broader MLOps landscape. Our findings align with systemic challenges documented in the MLOps literature [30, 39, 43], suggesting that the low adoption of specialized testing tools is not an isolated phenomenon, but rather a manifestation of a wider engineering challenge: the fragmentation of the MLOps ecosystem and the technical debt associated with integrating standalone ML components into existing production pipelines. As such, the testing-specific obstacles we observed appear consistent with known difficulties that lead practitioners to favor custom, easily controllable implementations over third-party solutions.

Key Findings for RQ$_3$

Our analysis shows a comprehensive picture of challenges in ML testing. Data sources converge on the centrality of foundational issues like the *Oracle Problem* and *Non-Determinism*. While often discussed theoretically in the literature, the survey confirms that they are major pain points for practitioners, and our mining study reveals their practical impact through the prevalence of workarounds such as "golden file" testing. The survey also allowed us to prioritize these challenges, showing that practical concerns such as computational and human costs (78% agreement) are felt most acutely. Finally, the inadequacy of the tooling ecosystem emerges as a central, multi-faceted challenge. The preference for custom logic observed in the code aligns with the survey, which found that practitioners cite a poor fit with their workflow (36%), high integration effort (30%), and lack of awareness (18%) as key barriers to adopting specialized tools.

6. Discussion and Implications of the Study

In this section, we first contextualize the findings within the current body of knowledge on ML testing. Afterwards, we discuss the actionable implications of the results.

6.1. State of Practice vs. State of the Art: How Far Apart Are They?

The current literature offers two main systematic analyses documenting the state of the art [4, 8]. By comparing our findings with these earlier, academically oriented overviews, we aim to provide a more practice-grounded perspective that complements them. Table 6 overviews the key convergences and gaps between state of the art and state of the practice.

We observe that researcher develops sophisticated techniques to address theoretical challenges, while practitioners prefer simpler methods to ensure operational integrity. Academic studies largely emphasized formal and data-driven testing approaches, including *Metamorphic Testing*, *Adversarial and Fuzz Testing*, and *Neuron Coverage* analysis. These are designed to address core theoretical challenges in ML system validation, such as the *Oracle Problem*, *Robustness* to input perturbations, and the need for *systematic coverage* of internal model behavior. However, the near-total absence of these techniques in practice points to a concrete gap. Our findings suggest that industry is largely overlooking the most mature academic solution to one of its most pressing challenges, namely the Oracle Problem. This likely reflects a mismatch between the assumptions underlying academic approaches—which tend to prioritize methodological rigor and formal completeness—and the practical constraints of industrial settings. Indeed, while such solutions offer measurable completeness, they often require controlled experimental conditions, specialized tools, or substantial computational resources, thereby limiting their applicability in real-world environments.

Table 6: Comparison between the state of the art and the state of practice.

Testing Aspect	State of the Art	State of Practice
Prevalent Techniques	Focus on advanced, model-centric techniques: *Metamorphic Testing*, *Adversarial Example Generation*, *Neuron Coverage*, and *Fuzzing*.	Dominated by foundational, pragmatic practices: *Smoke Testing*, *Rule-Based Checking*, and *Black-box Testing*. Advanced techniques are rarely implemented.
Tooling Ecosystem	Emphasis on specialized, often academic, tools designed for specific tasks: *DeepXplore*, *ART*, *DeepGauge*, *Giskard*.	Overwhelming reliance on general-purpose libraries (`NumPy`, `PyTest`) for custom solutions. Specialized tools are almost absent.
Primary Challenges	Theoretical and technical challenges are central: the *Oracle Problem*, ensuring *Robustness*, and defining *Coverage Metrics*.	Practical and resource-related challenges are most pressing: *Computational and Human Costs*, the *Oracle Problem*, and significant barriers to tool adoption (e.g., *high integration effort*, *poor workflow fit*).

In contrast, our results indicate that practitioners tend to adopt lighter-weight strategies centered on operational verification rather than formal assurance. For instance, the high frequency of *Smoke* and *Black-box* testing activities suggests that developers favor simple, reproducible procedures to validate whether ML components behave as expected under typical conditions. In other terms, instead of verifying fine-grained behavioral properties or structural coverage, developers typically focus on ensuring that models load correctly, produce plausible outputs, and maintain stable performance across datasets and releases. These practices reflect a pragmatic orientation: testing is perceived less as a scientific assessment of model correctness and more as a safeguard for reliability, maintainability, and workflow continuity. This contrast highlights a misalignment between the academic proposals for techniques and the immediacy of real-world development and industrial needs. The former are optimized for theoretical soundness and systematic exploration of model behavior, whereas the latter prioritize ease of integration, interpretability, and rapid feedback. Bridging this divide will likely require methods that combine the interpretive power and rigor of research prototypes with the usability, automation, and minimal overhead demanded in practice. Bridging this gap requires methods that combine the rigor of research prototypes with practical demands for usability, automation, and minimal overhead.

Consistent with this perspective, we cannot trace the specialized tools proposed in research, as developers tend to favor the flexibility and control offered by general-purpose libraries for implementing custom, ad hoc testing logic. This does not stem from a lack of awareness of available solutions but from a deliberate choice driven by pragmatic concerns. Practitioners perceive specialized tools as costly to adopt, poorly integrated with existing workflows, and sometimes misaligned with the structure of real-world development activities.

However, we also observe important commonalities between research and practice. Both communities identify the *Oracle Problem* as a central, still-unresolved challenge that limits the automation and reliability of ML testing. Despite methodological differences, there is a broad consensus that the definition of what constitutes "correct" behavior for ML systems, particularly those with stochastic outputs, remains an open question. Furthermore, the practitioner-driven emergence of *Fairness and Bias Testing*, as identified in our survey, mirrors a rapidly expanding focus on responsible and socially aware ML validation. This overlap indicates a progressive shift in priorities: as ML testing evolves, the boundary between technical quality assurance and ethical accountability blurs. Such convergence suggests a shared agenda in which societal impact and trustworthiness are both research imperatives and practical concerns.

In conclusion.

The comparison reveals a field characterized by both maturity and fragmentation. Academic research has produced a rich body of methods that address ML-specific testing challenges with theoretical rigor; however, these advances have not yet fully translated into widespread industrial adoption. Practitioners, at the same time, prefer pragmatic, lightweight approaches that ensure reliability and continuity but often lack formal grounding. This highlights the need for approaches that are not only scientifically robust but also easily integrable into real development workflows. At the same time, the emerging convergence on challenges such as the oracle problem and fairness testing suggests that research and practice are aligning around shared priorities, offering a promising foundation for the next stage of ML testing.

6.2. Implications of the Study

Starting from $\mathbf{RQ}_1$, we observed that practitioners, even without formal terminology, apply a comprehensive multi-level strategy, covering data, model, and system levels, as summarized in Table 7. The evidence from the two phases indicates that practitioners address quality concerns across the entire ML stack. To some extent, our findings suggest that testing in practice has organically evolved into a distributed process, where validation tasks are embedded throughout the ML pipeline rather than concentrated in a dedicated testing phase.

The analysis also highlights a hierarchy. The foundation of ML testing in practice is built on pragmatic, often straightforward strategies such as *Smoke Testing* and *Rule-Based Checking*, which aim to ensure basic operational integrity. In contrast, advanced techniques such as *Metamorphic* and *Adversarial Testing*, although prominent in the academic literature,

Table 7: Summary of results for $\mathbf{RQ}_1$, showing the multi-level strategy and hierarchy of ML testing across the two phases of our study. Colors and labels indicate the extent to which each practice appeared: **green** represents high adoption/usage, **yellow** medium adoption/usage, and **red** low or no evidence/adoption.

	Mining Study	Survey Study
Robustness Testing	Low	Low
Coverage-based Testing	Low	Low
Model Unit Testing	Medium	Low
Integration Testing	Medium	High
Model System Testing	Low	High
Smoke Testing	High	High
Metamorphic Testing	Medium	High
Mutation Testing	Low	Low
Black-Box Testing	High	High
Data-Box Testing	Medium	High
White-Box Testing	High	Medium
Domain-specific test cases	High	High
Input Testing	Medium	High
Fuzz Testing	Medium	Low
Symbolic Execution	Low	Low
Data Linter	Medium	High
Adversarial Examples	Medium	Low
Perturbed Model Validation	Low	Medium
Rule-Based Checking	High	High
Neuron Coverage-Based Testing	Low	Low

exhibit very low adoption rates. This gap suggests a preference for lightweight, easy-to-maintain controls, while advanced approaches are perceived as costly to learn, difficult to scale, and offering unclear short-term benefits. As such, our findings suggest that current practitioners' practices prioritize continuity and efficiency over methodological completeness, emphasizing solutions that are adaptable, transparent, and immediately actionable in real development contexts.

➔ Implication from $\mathbf{RQ}_1$.

Future research and tools should support and formalize a multi-level testing approach. This means (i) making testing responsibilities across data, model, and system levels explicit and standardized, and (ii) offering lightweight frameworks and libraries that align with these practices, rather than introducing entirely new paradigms. Building on this perspective, and considering the predominance of pragmatic testing strategies alongside the low adoption of advanced techniques, future work should improve the *usability*, *scalability*, and *integration* of advanced approaches, while also addressing barriers such as *learning cost*, *maintainability*, and *perceived short-term value*. The seamless integration of advanced testing into operators' established workflows could ultimately increase the likelihood of real adoption.

Moving to $\mathbf{RQ}_2$, as shown in Table 8, we observed a critical adoption gap for specialized tools. Instead of adopting them, developers prefer custom implementations, even for complex testing logic, using foundational libraries such as NumPy and PyTest. This preference is not accidental: developers trust the flexibility and control of tools they already master, preferring

Table 8: Summary of results for $\mathbf{RQ}_2$, showing the tools adoption to perform ML testing across the two phases of our study. Colors and labels indicate the extent to which each practice appeared: **green** represents high adoption/usage, **yellow** medium adoption/usage, and **red** low or no evidence/adoption.

	Mining Study	Survey Study
Giskard	Low	Medium
LangSmith	Medium	Low
Adversarial Robustness Toolbox (ART)	Low	Medium
Deepchecks	Low	Low
MATLAB Deep Learning Toolbox	Low	Low
numpy	High	Low
pytest	High	Low
tensorflow	High	High
pyro	High	Low
MLflow	High	High
common	Medium	Low
os	Medium	Low
sklearn	Medium	Medium
mxnet	Medium	Low
TextAttack	Medium	Low
Others (Evidently AI, Kolena, InterpretML, RobustBench)	Low	Medium

transparency over feature-rich but unfamiliar frameworks. As a consequence, ML testing remains highly decentralized and code-centric, with testing logic embedded directly in scripts, notebooks, and pipelines rather than externalized to dedicated infrastructure. While this practice enables rapid experimentation and lowers entry barriers, it also limits reproducibility and standardization, making quality assurance dependent on individual expertise rather than shared methodological conventions.

➔ Implication from $\mathbf{RQ}_2$.

Given the critical gap in the adoption of specialized tools and the prevalence of custom implementations, future work should study and formalize recurring custom testing patterns. Tool developers should design solutions that augment and extend these practices, for example, via lightweight plugins, wrappers, or libraries rather than replacing them with standalone frameworks that disrupt established workflows.

Finally, our survey identified a set of pragmatic barriers to adoption, as reported in Table 9. The preference for a custom approach is not driven by ignorance, but by deliberate choice: 40% of practitioners prefer custom checks with tools they already master. This is compounded by a perceived high integration effort (30%) and a poor fit with existing workflows (36%), suggesting that the current generation of specialized tools fails to meet developers' practical needs. These findings reinforce previous observations, confirming that practitioners' resistance to specialized tools is primarily driven by usability and workflow compatibility rather than by skepticism toward testing itself. In other words, the issue is not a lack of awareness, but a mismatch between research-driven design assumptions and the operational constraints of software teams.

➔ Implication from $\mathbf{RQ}_3$.

Since the preference for custom testing stems from pragmatic barriers, trust in familiar tools, high integration costs, and poor workflow fit, future tools must lower adop-

Table 9: Summary of results for **RQ**$_3$, showing the challenges in performing ML testing across the two phases of our study. Colors and labels indicate the extent to which each practice appeared: **green** represents high adoption/usage, **yellow** medium adoption/usage, and **red** low or no evidence/adoption.

	Mining Study	Survey Study
Oracle Problem	Low	High
Non-Determinism	Medium	High
Inherent complexity	Medium	High
Lack of standardized methods	Low	High
Vulnerability to adversial attacks	Low	High
Difficulty in defining metamorphic rel.	Low	Medium
High computational costs	Low	High
Balancing accuracy, robustness, and fairness	Low	High
Poor fit with my stack	Low	High
Preference for using custom solutions	High	High

tion costs and align with developers' cost-benefit perceptions. Tool developers should prioritize seamless integration, workflow compatibility, and incremental adoption to ensure that new tools deliver immediate, tangible value without disrupting established practices.

7. Threats to validity

This section discusses the threats to the validity and the strategies to mitigate them [47].

Threats to Construct Validity. In the mining study, threats arise from heuristics identifying ML projects and detecting testing practices, which may miss relevant activity (e.g., non-standard test formats/tools). Similarly, a specific threat concerns the reliability of using LLMs to classify testing strategies, as they may hallucinate or misinterpret code intent. To reduce these threats, we applied several mitigation strategies. In the first place, we evaluated the classifier on a manually annotated stratified sample designed to reflect the actual distribution of files encountered during classification. As such, this validation setup provides a representative estimate of the classifier's performance on the overall dataset, allowing us to assess its reliability before applying it to the full set of candidate files. Second, we used a *SELF-REFINE* [50] procedure that allows the model to iteratively review and revise its initial classification through a feedback loop. In practice, the model first produces an initial prediction, then provides feedback on its correctness, and finally refines the classification if necessary. This iterative process encourages the model to re-evaluate its reasoning and correct potential mistakes, thereby reducing the likelihood of inconsistent or poorly justified classifications, as demonstrated in prior studies on self-refinement techniques [51]. Third, we conducted a qualitative human validation on a sample of files to verify the plausibility of the LLM's rationale. Such a validation did not rely solely on a single aggregate accuracy value: we explicitly reported precision, recall, and F1-score for the positive class, thus providing a more transparent view of the behavior of the classifier. This latter analysis enabled us to monitor and control the trade-off between false positives and false negatives, ensuring the classifier maintained acceptable performance for our large-scale exploratory analysis. Indeed, the combined analysis of these metrics enabled us to identify potential inconsistencies in the classification behavior and verify that they remained within acceptable bounds, giving us confidence that the classifier was sufficiently reliable for identifying aggregate testing patterns across the dataset. While we acknowledge that automated analysis cannot fully replace human judgment, the consistency observed between the LLM's reasoning and manual inspection gives us confidence in the aggregated results. For transparency and reproducibility, we release the list of analyzed repositories, regular expressions, and prompts analyzed [23].

Perhaps more importantly, the purpose of the LLM-based classification in our study is to support a large-scale exploratory analysis of aggregate patterns, rather than to make claims about the correctness of any individual file classification. For this reason, while some residual noise may remain, we expect its impact on the high-level trends and conclusions to be limited. In particular, our analysis is based on aggregated statistics across a large dataset of 2,018 files, where occasional misclassifications are unlikely to systematically bias the observed patterns. Moreover, our findings show clear differences in prevalence across testing strategies (e.g., *Smoke Testing* appearing in nearly half of the files, while advanced techniques such as *Metamorphic Testing* appear in only a small fraction), suggesting that small classification errors would not materially affect the main conclusions of the study. For these reasons, we consider the classifier sufficiently reliable for identifying large-scale testing patterns, although we acknowledge that individual file classifications may still contain some residual noise.

In the survey, construct validity may be threatened by the design and interpretation of questions by respondents. Ambiguities in wording or the use of technical jargon could lead to misinterpretation, while self-reported responses may not always reflect actual practices. To mitigate these issues, we followed established guidelines for designing surveys in software engineering [55], ensuring that questions were clearly phrased, jargon was minimized, and response formats were consistent. We also piloted the questionnaire with a small group of external reviewers and revised the survey based on their feedback. Nonetheless, some advanced testing concepts may still have been understood differently across participants, particularly given their varied backgrounds and experience levels.

Threats to Internal Validity. In our study, we do not aim to establish causal relationships; thus, internal validity threats can be considered limited. However, in the mining study, unobserved confounding factors (e.g., project popularity, activity level) may influence which projects were selected and how testing behavior was detected. To mitigate this risk, we sampled from a broad set of repositories and applied consistent filtering criteria across the dataset. Future research could strengthen ecological validity by expanding the dataset, including more diverse sources, and explicitly modeling potential confounders.

As for the survey, the primary threat to internal validity stems from potential self-selection bias [68], particularly due to our reliance on voluntary participation through the Prolific platform. In particular, practitioners with a specific interest in testing may have been more inclined to participate, possibly skew-

ing the sample toward more engaged or knowledgeable respondents. Recruiting software practitioners for survey-based studies remains a persistent challenge, as evidenced by the low response rates commonly reported in prior research [55, 69, 70]. In this respect, relying on an established recruitment platform such as Prolific is a pragmatic, increasingly adopted strategy for accessing a broader, more diverse population. To mitigate self-selection and voluntary response bias [62, 63], we provided monetary compensation. Unlike unpaid surveys that are organically distributed through professional networks or social media, which tend to attract participants already particularly interested in the topic, the use of a compensated recruitment platform helps reach a broader and more heterogeneous pool of practitioners. This approach reduces the likelihood that the sample is composed primarily of individuals with a pre-existing strong interest in testing practices, thereby helping to diversify the participant population.

We also restricted participation to individuals with a proven track record (at least 50 completed tasks and a 100% approval rate), which serves as a proxy for worker quality and engagement. Furthermore, we embedded multiple attention checks and technical validation questions to filter out unqualified respondents. While we could not conduct a task-based assessment of coding skills due to the survey's scale, these criteria increase our confidence in the data quality.

Threats to Conclusion Validity. A key risk in mixed-method studies is the potential to over-interpret weak or inconsistent signals across different data sources [22]. To address this potential concern, we employed triangulation across the two phases, ensuring that findings were not considered in isolation but cross-verified. For instance, the lack of tool adoption observed in the mining study was directly explained by the survey responses regarding integration costs. Nonetheless, part of our interpretation involves qualitative coding, which may introduce bias. This was mitigated through multiple rounds of cross-checking and discussion among the authors.

Threats to External Validity. The mining study is based on open-source repositories hosted on GitHub. While the *MARK* dataset provides a comprehensive view of the open-source ML ecosystem, it may not fully reflect practices in proprietary, closed-source industrial environments, particularly in safety-critical domains (e.g., automotive) where testing processes are often more rigorous and regulated. Similarly, although the survey respondents are diverse in role and geography, they may not statistically represent the global population of ML practitioners. We partially mitigated these threats by ensuring broad coverage across domains and tools, but we recognize that further work, particularly through industrial case studies, would be valuable to deepen our perspective and validate findings in highly regulated settings.

8. Conclusion

This paper proposed an empirical investigation into how ML testing is implemented and perceived in practice. Motivated by the gap between academic proposals and the realities of software development, our study performs a mining study of 398 open-source repositories and a survey of 100 practitioners.

Our findings reveal a hierarchy of ML testing practices. While the state of the art offers a landscape of ML testing strategies and tools, practitioners rely on foundational, pragmatic strategies such as *Smoke Testing* and *Rule-Based Checking*, often implemented with general-purpose libraries. Conversely, we identified a critical adoption gap in the use of advanced techniques and specialized tools. Methods like *Metamorphic Testing* are rarely implemented, and dedicated testing tools are almost absent from the projects we analyzed. Our survey results help to explain this gap, highlighting significant practical barriers, including a preference for custom solutions, high integration costs, and a poor fit with existing developer workflows. These findings have direct implications: researchers should focus on the usability and integration of existing techniques, tool developers should prioritize seamless integration with ML platforms, and educators should balance teaching advanced theory with the pragmatic, foundational practices used in industry.

Our future research agenda includes replications of this work targeting different ecosystems (e.g., mobile or embedded ML), longitudinal analyses of testing evolution, and deeper qualitative studies of team workflows to help validate and refine our conclusions. Moreover, we aim to develop *human-oriented testing strategies* that are better aligned with the needs and constraints of real-world practitioners.

Credits

Alfonso Cannavale: Formal analysis, Investigation, Data Curation, Validation, Writing - Original Draft, Visualization. **Valeria Pontillo**: Formal analysis, Investigation, Data Curation, Validation, Writing - Original Draft, Visualization. **Fabio Palomba**: Supervision, Validation, Writing - Review & Editing. **Andrea De Lucia**: Supervision, Validation, Writing - Review & Editing.

Data Availability

The data collected in this research, along with the scripts and the results of the experiments, are publicly available in our online appendix [23].

Acknowledgement

We acknowledge the use of ChatGPT-4 and Grammarly to ensure linguistic accuracy and enhance the readability of this article. We acknowledge the support of the projects *FAIR* (Code: PE0000013), funded under the NRRP MUR program by the European Union – NextGenerationEU *Qual-AI* (Code: D53D23008570006), funded under the MUR PRIN 2022 program, Project PRIN 2022 PNRR *FRINGE*: context-aware FaiRness engineerING in complex software systEms (grant n. P2022553SL, CUP: D53D23017340001), and the European HORIZON-KDT-JU-2023-2-RIA research project *MATISSE*:

Model-based engineering of Digital Twins for early verification and validation of Industrial Systems (grant 101140216-2, KDT232RIA_00017).

References

[1] J. Zhou, F. Chen, Human and Machine Learning, Springer, 2018.

[2] P. Wang, E. Fan, P. Wang, Comparative analysis of image classification algorithms based on traditional machine learning and deep learning, Pattern recognition letters 141 (2021) 61–67.

[3] J. Ni, Y. Chen, Y. Chen, J. Zhu, D. Ali, W. Cao, A survey on theories and applications for self-driving cars based on deep learning methods, Applied Sciences 10 (2020) 2749.

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

[5] M. Felderer, R. Ramler, Quality assurance for ai-based systems: Overview and challenges (introduction to interactive session), in: International Conference on Software Quality, Springer, 2021, pp. 33–42.

[6] T. M. King, J. Arbon, D. Santiago, D. Adamo, W. Chin, R. Shanmugam, Ai for testing today and tomorrow: industry perspectives, in: 2019 IEEE international conference on artificial intelligence testing (AITest), IEEE, 2019, pp. 81–88.

[7] M. Pezzè, M. Young, Software testing and analysis: process, principles, and techniques, John Wiley & Sons, 2008.

[8] V. Riccio, G. Jahangirova, A. Stocco, N. Humbatova, M. Weiss, P. Tonella, Testing machine learning based systems: a systematic mapping, Empirical Software Engineering 25 (2020) 5193–5254.

[9] A. Martin-Lopez, Ai-driven web api testing, in: Proceedings of the ACM/IEEE 42nd international conference on software engineering: companion proceedings, 2020, pp. 202–205.

[10] C. Wan, S. Liu, S. Xie, Y. Liu, H. Hoffmann, M. Maire, S. Lu, Automated testing of software that uses machine learning apis, in: Proceedings of the 44th International Conference on Software Engineering, 2022, pp. 212–224.

[11] K. Tam, M. Walter, A. Barrett, D. Walker, Adversarial ai testcases for maritime autonomous systems, AI, Computer Science and Robotics Technology (2023).

[12] K. Filus, J. Domańska, Similarity-driven adversarial testing of neural networks, Knowledge-Based Systems 305 (2024) 112621.

[13] A. Aggarwal, S. Shaikh, S. Hans, S. Haldar, R. Ananthanarayanan, D. Saha, Testing framework for black-box ai models, in: 2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), IEEE, 2021, pp. 81–84.

[14] S. Hyun, M. Guo, M. A. Babar, Metal: metamorphic testing framework for analyzing large-language model qualities, in: 2024 IEEE Conference on Software Testing, Verification and Validation (ICST), IEEE, 2024, pp. 117–128.

[15] S. Segura, D. Towey, Z. Q. Zhou, T. Y. Chen, Metamorphic testing: Testing the untestable, IEEE Software 37 (2018) 46–53.

[16] C.-A. Sun, H. Dai, H. Liu, T. Y. Chen, Feedback-directed metamorphic testing, ACM Transactions on Software Engineering and Methodology 32 (2023) 1–34.

[17] T. Zhang, B. Kantarci, U. Siddique, Ai-augmented metamorphic testing for comprehensive validation of autonomous vehicles, in: 2025 IEEE/ACM 1st International Workshop on Software Engineering for Autonomous Driving Systems (SE4ADS), IEEE, 2025, pp. 1–4.

[18] F. Harel-Canada, L. Wang, M. A. Gulzar, Q. Gu, M. Kim, Is neuron coverage a meaningful measure for testing deep neural networks?, in: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2020, pp. 851–862.

[19] Z. Yang, J. Shi, M. H. Asyrofi, D. Lo, Revisiting neuron coverage metrics and quality of deep neural networks, in: 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE, 2022, pp. 408–419.

[20] J. Kim, R. Feldt, S. Yoo, Guiding deep learning system testing using surprise adequacy, in: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), IEEE, 2019, pp. 1039–1049.

[21] G. Rossolini, A. Biondi, G. Buttazzo, Increasing the confidence of deep neural networks by coverage analysis, IEEE Transactions on Software Engineering 49 (2022) 802–815.

[22] C. A. McKim, The value of mixed methods research: A mixed methods study, Journal of mixed methods research 11 (2017) 202–222.

[23] A. Cannavale, V. Pontillo, F. Palomba, A. De Lucia, Replication package: Understanding machine learning testing in practice, 2026. URL: https://figshare.com/articles/journalcontribution/_b_Replication_Package_Understanding_Machine_Learning_Testing_in_Practice_b_/31076047/1. doi:10.6084/m9.figshare.31076047.

[24] A. Aleti, Software testing of generative ai systems: Challenges and opportunities, in: 2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE), IEEE, 2023, pp. 4–14.

[25] H. B. Braiek, F. Khomh, On testing machine learning programs, Journal of Systems and Software 164 (2020) 110542.

[26] D. Marijan, A. Gotlieb, M. K. Ahuja, Challenges of testing machine learning based systems, in: 2019 IEEE international conference on artificial intelligence testing (AITest), IEEE, 2019, pp. 101–102.

[27] S. Grumbach, Z. Lacroix, On non-determinism in machines and languages, Annals of Mathematics and Artificial Intelligence 19 (1997) 169–213.

[28] A. Mallick, K. Hsieh, B. Arzani, G. Joshi, Matchmaker: Data drift mitigation in machine learning for large-scale systems, Proceedings of Machine Learning and Systems 4 (2022) 77–94.

[29] N. Baumann, E. Kusmenko, J. Ritz, B. Rumpe, M. B. Weber, Dynamic data management for continuous retraining, in: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, 2022, pp. 359–366.

[30] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, D. Dennison, Hidden technical debt in machine learning systems, Advances in neural information processing systems 28 (2015).

[31] S. Albelali, M. Ahmed, Testing machine learning and deep learning systems: Achievements and challenges, Arabian Journal for Science and Engineering (2025) 1–52.

[32] A. Tocchetti, L. Corti, A. Balayn, M. Yurrita, P. Lippmann, M. Brambilla, J. Yang, Ai robustness: a human-centered perspective on technological challenges and opportunities, ACM Computing Surveys 57 (2025) 1–38.

[33] A. M. Alaa, M. Van der Schaar, Demystifying black-box models with symbolic metamodels, Advances in neural information processing systems 32 (2019).

[34] T. Baluta, Z. L. Chua, K. S. Meel, P. Saxena, Scalable quantitative verification for deep neural networks, in: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), IEEE, 2021, pp. 312–323.

[35] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao, Y. Wang, Deepmutation: Mutation testing of deep learning systems, in: 2018 IEEE 29th international symposium on software reliability engineering (ISSRE), IEEE, 2018, pp. 100–111.

[36] Q. Hu, L. Ma, X. Xie, B. Yu, Y. Liu, J. Zhao, Deepmutation++: A mutation testing framework for deep learning systems, in: 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2019, pp. 1158–1161.

[37] M. Wang, N. Yang, D. H. Gunasinghe, N. Weng, On the robustness of ml-based network intrusion detection systems: An adversarial and distribution shift perspective, Computers 12 (2023) 209.

[38] M. Sponner, B. Waschneck, A. Kumar, Adapting neural networks at runtime: Current trends in at-runtime optimizations for deep learning, ACM Computing Surveys 56 (2024) 1–40.

[39] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, T. Zimmermann, Software engineering for machine learning: A case study, in: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), IEEE, 2019, pp. 291–300.

[40] S. Li, J. Guo, J.-G. Lou, M. Fan, T. Liu, D. Zhang, Testing machine learning systems in industry: an empirical study, in: Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice, 2022, pp. 263–272.

[41] Q. Song, M. Borg, E. Engström, H. Ardö, S. Rico, Exploring ml testing in practice: Lessons learned from an interactive rapid review with axis communications, in: Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI, 2022, pp. 10–21.

[42] M. Openja, F. Khomh, A. Foundjem, Z. M. Jiang, M. Abidi, A. E. Hassan, An empirical study of testing machine learning in the wild, ACM Transactions on Software Engineering and Methodology 34 (2024) 1–63.

[43] S. Mäkinen, H. Skogström, E. Laaksonen, T. Mikkonen, Who needs mlops: What data scientists seek to accomplish and how can mlops help?, in: 2021 IEEE/ACM 1st workshop on AI engineering-software engineering for AI (WAIN), IEEE, 2021, pp. 109–112.

[44] A. Serban, K. Van der Blom, H. Hoos, J. Visser, Adoption and effects of software engineering best practices in machine learning, in: Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2020, pp. 1–12.

[45] Z. Codabux, F. Fard, R. Verdecchia, F. Palomba, D. Di Nucci, G. Recupito, Teaching mining software repositories, in: Handbook on Teaching Empirical Software Engineering, Springer, 2024, pp. 325–362.

[46] J. A. Krosnick, P. J. Lavrakas, N. Kim, Survey research., Cambridge University Press, 2014.

[47] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in software engineering, volume 236, Springer, 2012.

[48] V. De Martino, G. Recupito, G. Giordano, F. Ferrucci, D. Di Nucci, F. Palomba, Into the ml-universe: An improved classification and characterization of machine-learning projects, Journal of Systems and Software 230 (2025) 112471. URL: https://www.sciencedirect.com/science/article/pii/S0164121225001396. doi:https://doi.org/10.1016/j.jss.2025.112471.

[49] R. Widyasari, Z. Yang, F. Thung, S. Qin Sim, F. Wee, C. Lok, J. Phan, H. Qi, C. Tan, Q. Tay, D. Lo, Niche: A curated dataset of engineered machine learning projects in python, in: 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR), IEEE, 2023, pp. 62–66.

[50] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, S. Gupta, B. P. Majumder, K. Hermann, S. Welleck, A. Yazdanbakhsh, P. Clark, Self-refine: Iterative refinement with self-feedback, Advances in neural information processing systems 36 (2023) 46534–46594.

[51] A. Della Porta, S. Lambiase, F. Palomba, Do prompt patterns affect code quality? a first empirical assessment of chatgpt-generated code, in: Proceedings of the 29th International Conference on Evaluation and Assessment in Software Engineering, 2025, pp. 181–192.

[52] A. Kong, S. Zhao, H. Chen, Q. Li, Y. Qin, R. Sun, X. Zhou, E. Wang, X. Dong, Better zero-shot reasoning with role-play prompting (2024) 4099–4113.

[53] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners, Advances in neural information processing systems 33 (2020) 1877–1901.

[54] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, W. Ye, Y. Zhang, Y. Chang, P. S. Yu, Q. Yang, X. Xie, A survey on evaluation of large language models, ACM transactions on intelligent systems and technology 15 (2024) 1–45.

[55] B. A. Kitchenham, S. L. Pfleeger, Principles of survey research part 2: designing a survey, ACM SIGSOFT Software Engineering Notes 27 (2002) 18–20.

[56] J. Morris, E. Lifland, J. Y. Yoo, J. Grigsby, D. Jin, Y. Qi, Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in nlp (2020) 119–126.

[57] K. H. Morin, Value of a pilot study, Journal of Nursing Education 52 (2013) 547–548.

[58] D. Andrews, B. Nonnecke, J. Preece, Conducting research on the internet: Online survey design, development and implementation guidelines (2007).

[59] F. Ebert, A. Serebrenik, C. Treude, N. Novielli, F. Castor, On recruiting experienced github contributors for interviews and surveys on prolific, in: International Workshop on Recruiting Participants for Empirical Software Engineering, 2022.

[60] B. Reid, M. Wagner, M. d'Amorim, C. Treude, Software engineering user study recruitment on prolific: An experience report, arXiv preprint arXiv:2201.05348 (2022).

[61] A. Alami, M. Zahedi, N. Ernst, Are you a real software engineer? best practices in online recruitment for software engineering studies, in: Proceedings of the 1st IEEE/ACM International Workshop on Methodological Issues with Empirical Studies in Software Engineering, 2024, pp. 52–57.

[62] J. J. Heckman, Selection bias and self-selection, in: Econometrics, Springer, 1990, pp. 201–224.

[63] J. W. Sakshaug, A. Schmucker, F. Kreuter, M. P. Couper, E. Singer, Evaluating active (opt-in) and passive (opt-out) consent bias in the transfer of federal contact data to a third-party survey agency, Journal of Survey Statistics and Methodology 4 (2016) 382–416.

[64] K. J. Hunt, N. Shlomo, J. Addington-Hall, Participant recruitment in sensitive surveys: a comparative trial of 'opt in'versus 'opt out'approaches, BMC Medical Research Methodology 13 (2013) 1–8.

[65] T. Hall, V. Flynn, Ethical issues in software engineering research: a survey of current practice, Empirical Software Engineering 6 (2001) 305–317.

[66] T. Nemoto, D. Beglar, Likert-scale questionnaires, in: JALT 2013 conference proceedings, 2014, pp. 1–8.

[67] S. Cavanagh, Content analysis: concepts, methods and applications., Nurse researcher 4 (1997) 5–16.

[68] J. J. Heckman, Selection bias and self-selection, in: The new Palgrave dictionary of economics, Springer, 2018, pp. 12130–12147.

[69] D. Lo, N. Nagappan, T. Zimmermann, How practitioners perceive the relevance of software engineering research, in: 10th Joint Meeting on Foundations of Software Engineering, 2015, pp. 415–425.

[70] T. Punter, M. Ciolkowski, B. Freimut, I. John, Conducting on-line surveys in software engineering, in: 2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings., IEEE, 2003, pp. 80–88.