On the Diffusion of Test Smells in LLM-Generated Unit Tests

WENDKÛUNI C. OUÉDRAOGO, University of Luxembourg, Luxembourg
YINGHUA LI*, University of Luxembourg, Luxembourg
XUEQI DANG, University of Luxembourg, Luxembourg
XUNZHU TANG, University of Luxembourg, Luxembourg
ANIL KOYUNCU, Bilkent University, Turkey
JACQUES KLEIN, University of Luxembourg, Luxembourg
DAVID LO, Singapore Management University, Singapore
TEGAWENDÉ F. BISSYANDÉ, University of Luxembourg, Luxembourg

LLMs promise to transform unit test generation from a manual burden into an automated solution. Yet, beyond metrics such as compilability or coverage, little is known about the *quality* of LLM-generated tests—particularly their susceptibility to *test smells*, design flaws that undermine readability and maintainability. This paper presents the first multi-benchmark, large-scale analysis of test smell diffusion in LLM-generated unit tests. We contrast LLM outputs with human-written suites (as the reference for real-world practices) and SBST-generated tests from EvoSuite (as the automated baseline), disentangling whether LLMs reproduce human-like flaws or artifacts of synthetic generation. Our study draws on 20,505 class-level suites from four LLMs (GPT-3.5, GPT-4, Mistral 7B, Mixtral 8×7B), 972 method-level cases from TestBench, 14,469 EvoSuite tests, and 779,585 human-written tests from 34,635 open-source Java projects. Using two complementary detection tools (TsDetect and JNose), we analyze prevalence, co-occurrence, and correlations with software attributes and generation parameters. Results show that LLM-generated tests consistently manifest smells such as Assertion Roulette and Magic Number Test, with patterns strongly influenced by prompting strategy, context length, and model scale. Comparisons reveal overlaps with human-written tests—raising concerns of potential data leakage from training corpora—while EvoSuite exhibits distinct, generator-specific flaws. These findings highlight both the promise and the risks of LLM-based test generation, and call for the design of smell-aware generation frameworks, prompt engineering strategies, and enhanced detection tools to ensure maintainable, high-quality test code.

CCS Concepts: • **Software and its engineering** → **Software testing and debugging**; • **Computer systems organization** → *Neural networks*.

Additional Key Words and Phrases: Test Smells, Large Language Models, Unit Test, SBST, Data Leakage, EvoSuite, LLM, Empirical Study

*Corresponding author.

Authors' addresses: Wendkûuni C. Ouédraogo, wendkuuni.ouedraogo@uni.lu, University of Luxembourg, Luxembourg; Yinghua Li, yinghua.li@uni.lu, University of Luxembourg, Luxembourg; Xueqi Dang, xueqi.dang@uni.lu, University of Luxembourg, Luxembourg; Xunzhu Tang, xunzhu.tang@uni.lu, University of Luxembourg, Luxembourg; Anil Koyuncu, anil.koyuncu@cs.bilkent.edu.tr, Bilkent University, Turkey; Jacques Klein, jacques.klein@uni.lu, University of Luxembourg, Luxembourg; David Lo, davidlo@smu.edu.sg, Singapore Management University, Singapore; Tegawendé F. Bissyandé, tegawende.bissyande@uni.lu, University of Luxembourg, Luxembourg.

0004-5411/2025/11-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Wendkûuni C. Ouédraogo, Yinghua Li, Xueqi Dang, Xunzhu Tang, Anil Koyuncu, Jacques Klein, David Lo, and Tegawendé F. Bissyandé. 2025. On the Diffusion of Test Smells in LLM-Generated Unit Tests. *J. ACM* 1, 1 (November 2025), 34 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Unit tests are foundational to modern software development, providing early feedback on correctness and supporting long-term maintainability [7, 59, 61]. However, writing effective unit tests is a labor-intensive and often introduces test smells—design flaws that hinder maintainability and understanding [5, 20, 30, 62].

Traditional automated test generation tools like EvoSuite [18] reduce manual effort but prioritize structural coverage over readability, frequently producing tests with quality issues such as Assertion Roulette or Eager Test [45, 48]. The emergence of Large Language Models (LLMs) such as GPT-4 [11] and Mistral [29] promises a paradigm shift toward generating more human-like, readable test suites [8, 16, 28, 46].

Despite growing interest in LLM-based test generation [44, 71], the quality of generated tests, particularly their susceptibility to test smells, remains largely unexplored. Prior studies by Siddiq et al. [60] and Ouédraogo et al. [43] only briefly addressed test smells without systematic analysis. Moreover, LLMs trained on public codebases risk exhibiting data leakage, potentially replicating flawed patterns from training data [22, 57].

This paper presents the first comprehensive study of test smell diffusion in LLM-generated unit tests. Our contribution is threefold: (1) we evaluate LLM-generated tests across diverse models, prompts, and benchmarks; (2) we analyze both class- and method-level test generation; and (3) we conduct complementary comparisons with SBST-generated tests (to assess automated paradigms) and human-written tests (to evaluate alignment with real-world practices and potential leakage) [6, 45].

We leverage two complementary benchmarks. The first comprises 20,505 class-level test suites generated by four LLMs across three datasets using five structured prompting strategies [44]. The second includes 972 method-level test cases from three LLMs under varying contextual conditions [71]. We compare these against 14,469 EvoSuite-generated tests (automated baseline) and 779,585 human-written tests from 34,635 Java projects (real-world reference).

For methodological robustness, we employ two independent test smell detection tools—TsDetect [51] and JNose [65]—enabling cross-validation and reducing tool-specific bias. We analyze how software characteristics (e.g., KLOC, RFC, CBO, cyclomatic complexity) and LLM parameters influence smell diffusion using multiple statistical measures (Pearson/Spearman correlations, Mutual Information, Kruskal–Wallis tests).

Contributions:

- **Multilevel Analysis of Test Smell Diffusion:** We quantify and compare the prevalence, distribution, and co-occurrence of test smells in LLM-generated test suites, analyzing both class-level and method-level benchmarks.
- **Prompt Engineering and Context Effects:** We examine how structured prompting techniques and contextual variations affect smell patterns in LLM-generated tests, providing insights into the role of prompt design in test quality.
- **Comparison with SBST (RQ1–RQ3):** We contrast LLM-generated tests with EvoSuite-generated tests to investigate differences between modern LLM-based and traditional SBST-based test generation paradigms.

- **Comparison with Human-Written Tests (RQ4):** We compare LLM-generated tests with 779k+ human-written tests to evaluate originality, alignment with real-world testing practices, and potential risks of data leakage.
- **Cross-Tool Validation:** We leverage both TsDetect and JNose to triangulate smell detection, quantify divergences, and ensure robustness across independent implementations.
- **Reproducibility:** All artifacts, including generated tests, smell detection results, and analysis scripts, are publicly available[1] to support full replication.

The remainder of the paper is structured as follows: Section 2 introduces key concepts and related work. Section 3 details our experimental design and datasets. Section 4 presents findings from each analysis. Section 5 addresses implications and threats. Section 6 reviews prior work, and Section 7 concludes.

2 BACKGROUND AND RELATED WORK

This section provides the foundational background (definitions of test smells, detection tooling, and the software characteristics we use as contextual variables) and a concise review of related work on human-written, SBST-generated, and LLM-based unit tests, as well as data-leakage risks in LLM evaluation. We conclude by positioning our study and clarifying how our design addresses gaps in smell coverage, detector reliability, and cross-granularity analysis

2.1 Test Smells and Unit Testing

Test smells are design flaws in test code that negatively impact its maintainability, readability, and reliability [5, 6, 20, 30, 62]. While these smells can be introduced by developers, they also frequently appear in automatically generated tests [6, 62]. The presence of test smells increases the long-term maintenance cost of test suites and complicates debugging when test failures occur. [45, 52].

To further illustrate these test smells, we present two representative examples in Listings 1 and 2, showcasing how they manifest in practice. The first example, from Listing 1, demonstrates how the overuse of assertions leads to test fragility, making tests harder to maintain and understand.

```
@MediumTest
public void testCloneNonBareRepoFromLocalTestServer() throws Exception {
    Clone cloneOp = new Clone(false, integrationGitServerURIFor("small-repo.early.git"),
        helper().newFolder());
    Repository repo = executeAndWaitFor(cloneOp);
    assertThat(repo, hasGitObject("ba1f63e4430bff267d112b1e8afc1d6294db0ccc"));
    File readmeFile = new File(repo.getWorkTree(), "README");
    assertThat(readmeFile, exists());
    assertThat(readmeFile, ofLength(12));
}
```

Listing 1. Example of Assertion Roulette Test Smell

The second, in Listing 2, highlights how hardcoded values—magic numbers—complicate test clarity and hinder long-term evolution. These test smells, by obscuring the intent of tests and complicating their maintenance, present significant obstacles to both test comprehension and long-term evolution.

```
@Test
public void testGetLocalTimeAsCalendar() {
    Calendar localTime = calc.getLocalTimeAsCalendar(BigDecimal.valueOf(15.5D), Calendar.
        getInstance());
    assertEquals(15, localTime.get(Calendar.HOUR_OF_DAY));
    assertEquals(30, localTime.get(Calendar.MINUTE));
}
```

[1] https://anonymous.4open.science/r/LLMTSDiff-B341/

Listing 2. Example of Magic Number Test Smell

These smells represent core obstacles to test comprehension and evolution. For further details and additional examples of test smells, refer to the TsDetect documentation [2] [51]. Detecting such smells has been the subject of extensive research, leading to several automated detectors. In our study, we rely on two widely used tools: TsDetect [51] and JNose Test [65]. However, relying on a single detector risks bias due to tool-specific heuristics or limitations in smell coverage. Recent work by Panichella et al. [49] highlights concerns about the detectability, validity, and reliability of test smell detectors, emphasizing the need for cross-tool validation to enhance confidence in empirical results. Similarly, the systematic mapping study by Aljedaani et al. [2] shows that detection tools often overlap only partially in the types of smells they support. This motivates our decision to use both detectors, which provide complementary coverage and granularity.

In this study, we focus on the most frequently observed test smells across LLM-generated, SBST-generated, and human-written tests. Table 1 presents a summary of the most frequent smells identified in our analysis, along with their descriptions and the impact they have on test quality.

Table 1. Most Frequent Test Smells Observed in Our Study

Test Smell	Description and Impact
Assertion Roulette (AR)	Multiple assertions without messages hinder fault diagnosis.
Magic Number Test (MT)	Numeric literals in assertions reduce clarity and increase maintenance.
Empty Test (EmT)	Tests without executable statements may falsely pass and hide coverage gaps.
Exception Handling (EH)	Manual exception logic disrupts automation; use framework support instead.
Sensitive Equality (SE)	Reliance on `toString()` causes brittle tests prone to break.
Eager Test (EaT)	Tests that call multiple production methods are harder to isolate and comprehend.
Lazy Test (LT)	Multiple tests for one method indicate redundancy and lack of focus.
Unknown Test (UT)	Tests without assertions lack observable behavior and reduce diagnostic power.
Duplicate Assert (DA)	Repeated assertions suggest poor modularization and dilute test intent.
General Fixture (GF)	Unused setup objects increase maintenance cost and slow execution.
Conditional Logic Test (CLT)	Control-flow in tests reduces determinism and hampers localization [51].

2.2 Software Characteristics

Understanding software structure is essential when analyzing the prevalence of test smells. Prior studies have demonstrated that certain structural and object-oriented metrics serve as strong indicators of fault-proneness and maintenance complexity in software systems [12, 31, 54]. These characteristics have also been extensively used in defect prediction tasks [21, 31, 54] and shown to correlate with software quality and test maintainability [4, 9, 10]. Building on these findings, our study focuses on the following characteristics, which we hypothesize to influence the likelihood and type of test smells:

- **Lines of Code (LOC):** Larger codebases tend to exhibit more complexity and maintenance challenges [54]. Such systems often require more extensive tests, increasing the chance of smells like *General Fixture (GF)* or *Lazy Test (LT)* due to oversized test setups.
- **Cyclomatic Complexity (CC):** High control-flow complexity [41] can encourage developers or automated tools to write tests with multiple assertions or branches, raising the likelihood of smells such as *Assertion Roulette (AR)* or *Conditional Logic Test (CLT)*.

[2] https://testsmells.org/pages/testsmellexamples.html

- **Coupling Between Objects (CBO):** Strong coupling between classes [12] makes test isolation harder and may foster smells such as *Eager Test (EaT)* or *General Fixture (GF)* because tests must instantiate and configure many collaborators.
- **Response For a Class (RFC):** A high RFC [12] implies many possible execution paths, which may push test generators toward breadth over focus. This can yield *Duplicate Assert (DA)* or *Lazy Test (LT)* smells as tests attempt to cover numerous interactions in one place.
- **Depth of Inheritance Tree (DIT):** Deep hierarchies [12] can introduce subtle dependencies that are difficult to isolate, leading to smells such as *Unknown Test (UT)* or brittle *Exception Handling (EH)* logic in generated tests.
- **Test Scope (Method-level vs. Class-level):** The granularity of test generation directly affects smell patterns. Prior work has shown that class-level SBST (e.g., EvoSuite) tends to produce tests with frequent smells such as *Assertion Roulette* and *Eager Test* [45], while recent studies on method-level LLM-based test generation report fewer structural issues [60]. Building on these observations, our study systematically analyzes how test smells vary across scopes.

By connecting these software characteristics to potential test smell manifestations, we provide the rationale for analyzing them in conjunction with our empirical results.

2.3 Human-Written, Automated, and LLM-Based Unit Tests

Human-written tests have been extensively analyzed for the presence of test smells, particularly with regard to their impact on test comprehension, maintainability, and effectiveness. Bavota et al.[6] showed that smells like Assertion Roulette and Magic Number Test are common in JUnit tests, impacting comprehension and maintainability. While these tests reflect domain knowledge and are generally more readable, they are time-consuming and often miss edge cases[5, 15, 36, 37]. Datasets like Defects4J [32] and SF110 [19] highlight common issues like test smells and incomplete coverage [3, 62].

In contrast to human-written test cases, tools like EvoSuite [18] automate test generation by focusing on maximizing code coverage. However, as Palomba et al. [45] showed, EvoSuite-generated tests are also prone to test smells such as *Assertion Roulette* and *Eager Test*, raising concerns about their maintainability. More recently, studies have applied LLMs to unit test generation, reporting promising diversity and readability, yet test smells have generally been treated as a secondary aspect rather than a primary focus [43, 60].

Despite advances in prompt engineering, such as techniques like Chain-of-Thought (CoT) [67] and Tree-of-Thought (ToT) [69], LLMs still encounter challenges in achieving the quality of manually written tests, particularly in terms of maintaining clarity and minimizing test smells [44, 60]. Recent studies further highlight that LLM-based test generation can suffer from instability across generations, overfitting to benchmarks, and sensitivity to prompt variations [22, 23, 68]. Although advanced prompting strategies can mitigate some issues, achieving comparable coverage, readability, and maintainability remains challenging. This gap is critical, as it directly impacts the long-term reliability of LLM-generated tests. Our study addresses this gap by systematically analyzing how prompt engineering shapes test smell prevalence and maintainability.

Comparison. Traditional SBST tools such as EvoSuite excel at maximizing structural coverage [18], but the resulting tests often suffer from rigidity and poor readability, with smells like *Assertion Roulette* and *Eager Test* being especially common [45]. In contrast, LLM-based approaches have shown the ability to produce tests that appear stylistically closer to human-written code, offering greater diversity and generalizability in terms of test scenarios [43, 60]. However, they also exhibit distinct weaknesses, including instability across generations, redundant or verbose patterns, and

the emergence of smell types that are less prevalent in SBST or human-written suites. For example, while prior studies have shown that Assertion Roulette and Magic Number Test dominate in human-written tests [6, 52, 62], and Assertion Roulette and Eager Test are most frequent in EvoSuite-generated tests [45], LLM-based generation more often introduces smells such as *General Fixture (GF)*, *Duplicate Assert (DA)*, or *Lazy Test (LT)*. A key distinction is that, unlike SBST, LLMs are not explicitly optimized for coverage. Instead, they generate tests that often appear more "human-like," which alters the distribution of smells observed and motivates a systematic investigation of these differences.

2.4 Data Leakage in LLMs

Data leakage in LLMs refers to cases where models reproduce memorized patterns from their training data, rather than generating novel solutions. In test generation, this can manifest as recurring smells such as *Assertion Roulette* or *Magic Number Test*, which may be inherited from benchmarks like Defects4J or SF110 [23, 60, 68]. Such memorization reduces the effectiveness and maintainability of generated tests, and may also cause benchmark leakage—where models appear to perform well on datasets overlapping with their training data, thus inflating evaluation results [22, 57]. Distinguishing genuine reasoning ability from pattern recall is therefore challenging, underscoring the need for careful evaluation when analyzing LLM-generated tests.

To address these concerns, frameworks like UniTSyn emphasize the importance of careful evaluation to mitigate the impact of data leakage [23]. Our study builds on these insights by comparing test smell patterns between LLM-generated and human-written tests, investigating whether LLMs inadvertently replicate patterns from their training data, thus introducing biases and affecting test quality.

Positioning. Taken together, prior studies highlight four open issues: (i) partial coverage and limited reliability of existing smell detectors, (ii) the influence of test scope (method- vs class-level) on smell prevalence, (iii) the lack of systematic analyses of test smells in LLM-generated tests compared to SBST and human-written suites, and (iv) the risk of data leakage when evaluating LLM outputs on public benchmarks. Our study addresses these issues by combining two complementary detectors, analyzing both SBST- and LLM-generated tests at multiple granularities, and comparing them against a large corpus of human-written tests.

3 EXPERIMENTAL SETUP

3.1 Approach Overview

Our methodology, illustrated in Figure 1, investigates the diffusion of test smells in LLM-generated test suites and contrasts them with SBST-generated and human-written counterparts. We leverage two benchmarks that are complementary in scope: Benchmark 1 [44] targets class-level test generation and analyzes the influence of five distinct prompt engineering strategies across multiple LLMs and datasets, whereas Benchmark 2 [71] focuses on method-level generation and studies the impact of prompt context by varying the amount of information provided to the model. While their generation settings differ, the two benchmarks offer complementary insights into how generation control—via prompting at the class level or contextual variation at the method level—shapes test smell profiles. In addition, we incorporate over 779,000 human-written test cases from 34,635 projects and 14,469 EvoSuite-generated test suites, allowing us to jointly consider human-written, SBST-generated, and LLM-generated tests in our assessments.

We conduct two primary analyses. In **Analysis 1**, we investigate how prompt strategies and generation settings influence the emergence of test smells in LLM-generated suites, comparing their

prevalence and co-occurrence with EvoSuite's SBST-generated tests. This analysis incorporates software attributes such as cyclomatic complexity and class coupling, as well as LLM-specific parameters like input size and context length. To uncover both linear and non-linear relationships, we apply Pearson correlation, Kruskal-Wallis, Spearman, and Mutual Information.

In **Analysis 2**, we compare LLM-generated suites with human-written ones to examine both the prevalence and distribution of test smells as well as their similarity. This dual perspective allows us to characterize how LLM-generated smell profiles align or diverge from those observed in human-written tests. Since LLMs are trained on human-produced code, human-written tests represent the most plausible source of inherited patterns. Beyond descriptive comparison, this analysis also investigates potential data leakage—whether LLMs inadvertently replicate test smells present in human-written tests, rather than producing them as novel generative artifacts. To this end, we measure smell prevalence and type distributions, assess overlap using Cosine Similarity and Jaccard Index, and evaluate frequency distribution differences with Euclidean and Hellinger Distances. Additionally, we use Pearson Correlation, Kruskal-Wallis, Spearman, and Mutual Information to analyze co-occurrence patterns and detect possible replication of human test behaviors in LLM-generated tests.

To ensure the robustness of our findings, we adopt a triangulated analysis strategy [17, 35, 56, 70]. We apply *tool triangulation* via two independent smell detectors (TsDetect and JNose), *data triangulation* using two complementary benchmarks—Benchmark 1 (class-level, varied prompting) and Benchmark 2 (method-level, contextual variation)—and *analytical triangulation* by combining linear and non-linear statistics. These benchmarks offer distinct yet complementary insights into how generation control affects test smell profiles. We present results per tool, identify key trends, and discuss divergences in light of tool-specific heuristics and detectability concerns, as highlighted by Panichella et al. [49].

3.2 Research Questions

- **RQ1: What is the prevalence and distribution of test smells in LLM-generated unit tests, and how do they compare to tests generated by SBST?**
  This research question assesses the presence of test smells in LLM-generated tests across both class-level (Benchmark 1) and method-level (Benchmark 2) generation. We explore: ❶ *Prevalence*: how frequently test smells appear in LLM-generated test cases; ❷ *Distribution*: the distribution of test smell types at class and method levels, considering both LLM- and SBST-generated tests, and how generation strategies influence them; and ❸ *Comparison with EvoSuite*: how LLM-generated test smells differ from those found in tests produced by SBST tools such as EvoSuite. This analysis aims to provide a comprehensive understanding of test smell diffusion across different test generation paradigms and levels of granularity.
- **RQ2: How do test smells co-occur in LLM-generated test suites, and how do these patterns compare to tests generated by SBST?**
  This research question investigates: ❶ *Influence of prompt engineering techniques*: how structured prompting affects test smell prevalence and interaction; ❷ *Effect of context variation*: whether the amount or structure of contextual information impacts test smell emergence; ❸ *Co-occurrence patterns*: whether certain test smells frequently appear together, and how these patterns differ between class-level and method-level generation; ❹ *Comparison with SBST-generated tests*: whether LLM-generated tests exhibit distinct co-occurrence trends compared to SBST-generated tests. By identifying these trends, we aim to uncover structural dependencies and generative biases that may hinder test suite readability, maintainability, or fault localization.
- **RQ3: How do software attributes and LLM parameters influence the prevalence of test smells?**

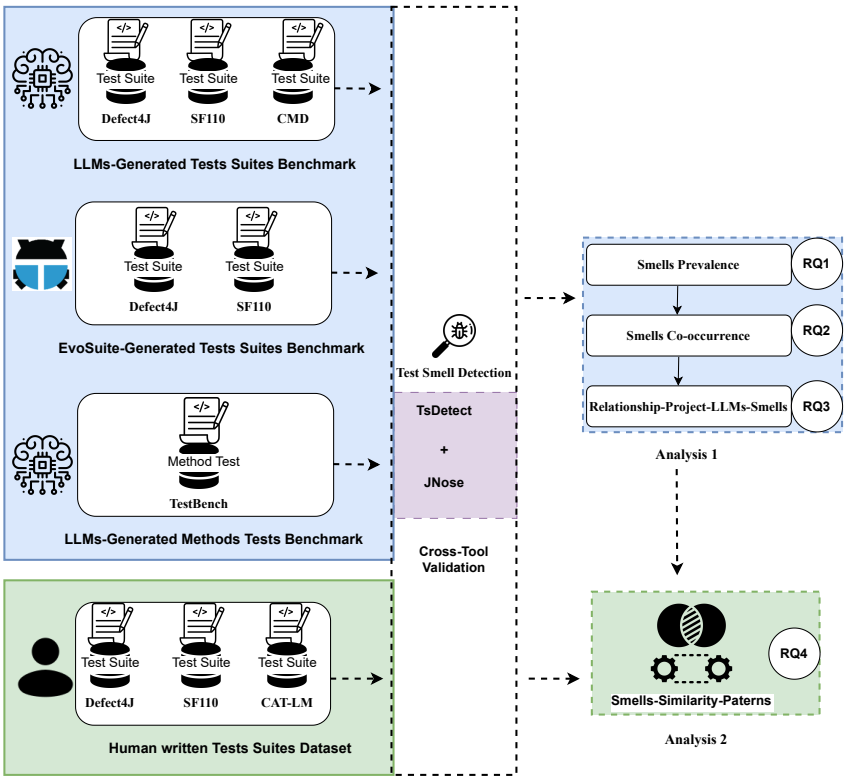


Fig. 1. Overview of the pipeline used to design the analysis.

This research question investigates ❶ *software characteristics*: how project size (LOC, number of methods, classes), complexity metrics (cyclomatic complexity, class coupling), and test scope (method-level vs. class-level) impact test smell prevalence; ❷ *LLM parameters*: the influence of model size, training parameters, context window size, and prompting techniques on test smell patterns; and ❸ *comparative analysis*: how these factors affect test smells differently between LLM-generated and EvoSuite-generated tests. By exploring these relationships, we aim to understand how software attributes and generative parameters contribute to the emergence of test smells, ultimately informing more effective and targeted strategies for both LLM-based and SBST-based test generation.

- **RQ4: Do LLM-generated tests exhibit similar test smells to human-written tests?**
  This research question investigates ❶ *the extent of overlap in test smell types and frequencies* between LLM-generated and human-written tests, and ❷ *whether such similarities may be indicative of data leakage*, where LLMs unintentionally replicate patterns learned from test suites present in their training data. This analysis aims to assess the originality and generalizability of LLM-generated test code, and the potential influence of training artifacts on test quality.

3.3 Datasets and benchmarks

Our study considers the following datasets and benchmarks: **Benchmark 1** [44], a class-level evaluation with LLM- and SBST-generated test suites across multiple datasets and prompting strategies; **Benchmark 2** [71], a method-level benchmark exploring different prompt context

levels; in addition to a large corpus of human-written test suites collected from Defects4J, SF110, and Cat-LM [53]. Together, these resources provide LLM-, SBST-, and human-generated tests for our comparative analyses.

3.3.1 Benchmark 1. **Benchmark 1** [44] targets class-level unit test generation and evaluates the impact of prompt engineering across multiple LLMs and datasets. It comprises 20,505 test suites generated using four models—GPT-3.5, GPT-4, Mistral 7B, and Mixtral 8x7B—on three datasets: Defects4J, SF110, and CMD (Table 2).

Table 2. Datasets Used in Benchmark 1 and 2

Dataset	Description	#Proj	#Classes / Methods
SF110	Java programs from academia and industry for SBST research.	74	182 classes
Defects4J	Real-world bug-fix scenarios exposing faults.	16	477 classes
CMD	Curated modern open-source Java repos (e.g., OceanBase).	2	31 classes
TestBench	108 Java methods from 9 open-source projects across domains.	9	108 methods

The CMD dataset includes modern, production-grade Java systems (e.g., Conductor OSS[3], OceanBase Developer Center (ODC)[4]), complementing Defects4J and SF110 with recent modular codebases. Due to resource constraints, GPT-4 and Mixtral 8x7B were only applied to CMD, while EvoSuite was excluded from CMD due to Java compatibility issues [5].

Each model was prompted using five structured strategies: ZSL, FSL, CoT, ToT, and GToT (Table 3). For comparison, 14,469 EvoSuite-generated test suites were used as a structural testing baseline.

Table 3. Overview of Prompting Strategies and Contexts Used in Benchmark 1 and Benchmark 2

Strategy / Context	Benchmark	Description	Objective
Zero-Shot Learning (ZSL)	B1	No examples or context; the model directly generates tests.	Serves as baseline to evaluate raw test generation ability.
Few-Shot Learning (FSL)	B1	Provides 1-2 examples before test generation.	Helps the model infer correct structure and semantics of test code.
Chain-of-Thought (CoT)	B1	Generates test logic through step-by-step reasoning.	Improves assertion clarity and logical flow.
Tree-of-Thought (ToT)	B1	Explores multiple reasoning branches before generating tests.	Encourages structural diversity and robustness.
Guided Tree-of-Thought (GToT)	B1	Simulates discussion among multiple "testers" refining test generation iteratively.	Aims to improve correctness, coverage, and variability.
Self-Contained Context (SCC)	B2	Only the target function's signature and body are provided.	Evaluates base-level capability without class context.
Simple Context (SC)	B2	Full class structure excluding function bodies.	Preserves structural information with reduced length.
Full Context (FC)	B2	Entire class including target method is given.	Tests whether full context leads to more accurate and meaningful test cases.

All LLMs used a temperature of 0.7, with model-specific token limits. Each class underwent 30 generation attempts. EvoSuite was executed using the DynaMOSA algorithm [47], with 3 minutes per class and 30 iterations. Importantly, EvoSuite's post-processing—designed to reduce flaky tests and refactor away certain test smells—was deliberately disabled to ensure a fair comparison with the raw outputs of LLMs.

[3] https://github.com/conductor-oss/conductor
[4] https://github.com/oceanbase/odc
[5] https://github.com/EvoSuite/evosuite/issues/433

Table 4. Overview of Large Language Models (LLMs) and SBST Tools Used Across Benchmarks

Model / Tool	Benchmark(s)	Size	Type	Strengths
GPT-3.5-Turbo	B1, B2	~175B (estimated)	Proprietary (OpenAI)	Widely used for LLM-based test generation; good balance of cost, performance, and generalization.
GPT-4	B1, B2	>175B	Proprietary (OpenAI)	Stronger reasoning, coherence, and test intent alignment; higher generation cost.
Mistral 7B	B1	7B	Open-source (Mistral)	Lightweight and efficient; less capable for multi-step reasoning in test generation.
Mixtral 8x7B	B1	8×7B (MoE)	Open-source (Mistral)	Mixture-of-Experts improves generation diversity with selective routing.
CodeLlama-13B-Instruct	B2	13B	Open-source (Meta)	Strong coding ability, but context handling weaker than proprietary models.
EvoSuite	B1	N/A	Open-source SBST Tool	Provides high structural coverage; readability and maintainability are not prioritized.

Table 5. Summary of Collected Test Suites

Type	Model/Tool	Bench	#Proj	#Suites
LLM	GPT-3.5	B1	3	16,744
	GPT-4	B1	1	421
	Mistral 7B	B1	3	3,318
	Mixtral 8x7B	B1	1	22
	GPT-3.5	B2	9	324
	GPT-4	B2	9	324
	CodeLlama-13B	B2	9	324
			Total LLM	**21,801**
SBST	EvoSuite	B1	2	14,469
Human	—	All	34,635	779,585

3.3.2 Benchmark 2. **Benchmark 2** [71] targets method-level unit test generation, enabling a fine-grained evaluation across dimensions such as syntactic correctness, compilability, runtime validity, coverage, and mutation-based defect detection. It includes 972 test cases generated by three LLMs—GPT-3.5, GPT-4, and CodeLlama-13B-Instruct—under three prompt contexts: Self-Contained Context (SCC), Simple Context (SC), and Full Context (FC), with 108 test methods produced per model-context pair (Table 2).

Prompting strategies and their context definitions are detailed in Table 3. The collected dataset enables the analysis of test smell diffusion across varying levels of prompt granularity and conditioning.

The 108 target functions span nine diverse open-source Java projects, covering domains such as concurrency, microservices, data visualization, and low-code platforms. All LLMs were configured with a temperature of 0.7; OpenAI models used top-p sampling, and for consistency, only top-p was used for CodeLlama. Token limits were adjusted to each model's maximum context window (Table 4).

3.3.3 Human-written tests dataset. In addition to LLM-generated tests, we created a combined dataset of human-written tests from SF110 [19], Defects4J [32], and the Cat-LM benchmark [53]. Our dataset includes 34,635 well-maintained Java projects, with the vast majority (34,563) coming from the Cat-LM benchmark, complemented by 56 projects from SF110 and 16 from Defects4J (Table 5). The resulting corpus comprises 779,585 test cases, with Cat-LM contributing most significantly (776,847 tests), while SF110 and Defects4J provide 1,546 and 1,192 tests respectively. This comprehensive collection of human-written tests from diverse sources enables a robust comparison with LLM-generated tests, allowing us to systematically identify similarities and

differences in test quality characteristics and smell patterns between human and AI-generated testing approaches.

3.4 Software Attributes and LLM Parameters

Understanding the factors influencing test smell diffusion is crucial for optimizing test generation techniques. We analyze these factors across three sources of tests: LLM-generated, SBST-generated (EvoSuite), and human-written. The focus depends on the experiment: in **Analysis 1**, we contrast LLMs with EvoSuite to study how generation strategies and software attributes (e.g., project size, complexity, test scope, and class coupling) affect smell prevalence and distribution; in **Analysis 2**, we contrast LLMs with human-written tests to examine prevalence, distribution, and similarity, with a particular focus on potential data leakage. Human tests serve as the relevant baseline for inherited patterns, while SBST tests provide a baseline for alternative automated, coverage-driven generation. Table 6 summarizes both software system attributes (e.g., project size, complexity, test scope, and class coupling) and LLM-specific parameters (e.g., model size, temperature, top_p, and context length) considered in our evaluation.

Table 6. Characteristics of LLMs and Software Artifacts in Benchmark 1 and 2.

Benchmark	Model	#Params (B)	Ctx Len	Temp.	Top-p	#Classes	#Methods	KLOC	CBO	RFC	DIT	CyCo
Benchmark 1 (Class-Level)	GPT-3.5 Turbo	175	4096	0.7	1.0	690	8922	85.28	7.77	14.01	1.55	2.49
	GPT-4	1750	8192	0.7	1.0	31	164	2.37	12.68	26.24	1.32	2.25
	Mistral 7B	7	4096	0.7	0.95	690	8922	85.28	7.77	14.01	1.55	2.49
	Mixtral 8x7B	46.7	8192	0.7	0.95	31	164	2.37	12.68	26.24	1.32	2.25
	EvoSuite-Defects4J	–	–	–	–	477	5905	50.96	6.23	13.24	1.55	2.55
	EvoSuite-SF110	–	–	–	–	182	2853	31.95	4.3	14.79	1.56	2.67
Benchmark 2 (Method-Level)	GPT-3.5 Turbo	175	4096	0.7	1.0	–	108	1.38	1.98	4.75	1	4.99
	GPT-4	1750	8192	0.7	1.0	–	108	1.38	1.98	4.75	1	4.99
	CodeLlama-13B	13	4096	0.7	0.95	–	108	1.38	1.98	4.75	1	4.99

* #Params (B): #Parameters (Billion), Ctx Len: Context Length, Temp: Temperature.

3.4.1 Software System Attributes. We describe the software attributes used to evaluate their influence on test smell prevalence, employing a set of well-established metrics: LOC, number of methods, number of classes, Cyclomatic Complexity (CyCo), Coupling Between Objects (CBO), Response for a Class (RFC), and Depth of Inheritance Tree (DIT). Following prior work [6, 45], we compute total values for size metrics (e.g., LOC, classes, methods) and project-level means for structural metrics (e.g., CBO, RFC, DIT, CyCo) to ensure comparability. These metrics are extracted using `ck`[6] for object-oriented features and `Lizard`[7] for complexity and method-level information.

3.4.2 LLM-Specific Parameters. LLMs differ in several key parameters—such as model size, context length, temperature, and top-p sampling—that may influence the structure and quality of generated test cases. Although we reused existing benchmarks and did not configure these parameters ourselves, our study takes into account their differences across models to explore potential associations with test smell diffusion. Open-source models support both top-k and top-p sampling, while OpenAI models expose only top-p for nucleus sampling [25]. For consistency, all models were evaluated under top-p sampling. Table 6 summarizes these parameters, including model size (number of parameters), context length (token window), temperature (diversity control), and top-p (sampling scope).

[6] https://github.com/mauricioaniche/ck
[7] https://github.com/terryyin/lizard

3.5 Test Smell Detection Strategy

To ensure comprehensive and reliable test smell detection, we employ both TsDetect [51] and JNose Test [65]. We rely on the most recent version of TsDetect (v2.2)[8], which supports 21 test smells—the same number as JNose. Although their coverage is aligned, the tools differ significantly in implementation and reporting granularity. TsDetect relies on static structural heuristics and provides smell-level classification across entire projects, whereas JNose enables more fine-grained localization at the method or class level, with multiple analysis modes (per class, per smell, per file).

Using both tools enables cross-validation of smell detection through independent implementations and offers complementary perspectives—leveraging TsDetect's robust detection pipeline alongside JNose's precise mapping of smell instances within test classes and lines of code.

3.6 Metrics

3.6.1 Co-occurrence. In the context of this work, we apply co-occurrence [1, 50, 55] to quantify how often two test smells appear together within the same project or class, relative to the total occurrences of one of the test smells. In this work, we use the following formula to compute the co-occurrence of two test smells t_i and t_j:

$$\text{co-occurrence}_{t_i,t_j} = \frac{|t_i \wedge t_j|}{|t_i|} \tag{1}$$

Where $|t_i \wedge t_j|$ represents the number of times both test smell t_i and t_j appear together in the same project or class (joint occurrences), and $|t_i|$ is the total number of occurrences of test smell t_i.

By calculating the co-occurrence between various test smells, we can identify whether certain test smells tend to appear together more frequently than by random chance. This helps uncover potential relationships between test smells and provides insights into how they might affect software quality collectively.

3.6.2 PMCC. The Pearson Product-Moment Correlation Coefficient (PMCC) [13, 34] is a metric that measures the strength and direction of the linear relationship between two continuous variables. It is commonly denoted by r and can take values between -1 and 1. Cohen et al. [13] offered a set of guidelines for interpreting the correlation coefficient r. The formula for the Pearson correlation coefficient r between two variables X and Y is:

$$r = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2 \sum_{i=1}^{n}(Y_i - \bar{Y})^2}} \tag{2}$$

Where X_i and Y_i are the individual sample points, $\bar{X}$ and $\bar{Y}$ represent the means of X and Y, respectively, and n denotes the number of data points.

We use PMCC to assess correlations between the presence of test smells and system characteristics(Table 6).

3.6.3 Similarity metrics. To compare the patterns of test smells between LLM-generated and human-written test suites, we use several similarity metrics, each designed to capture different aspects of the test smell distribution.

- **Cosine Similarity** We use Cosine Similarity [26, 40] to compare the frequency of test smells between the LLM-generated and human-written test suites.

[8] https://github.com/TestSmells/TestSmellDetector/releases/tag/v2.2

$$\text{Cosine Similarity} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{3}$$

Where A_i and B_i are vectors representing the frequency of test smells in the respective test suites.

- **Jaccard Index** We apply the Jaccard Index [27, 42] to assess the overlap in the frequency of test smells between the two types of test suites, focusing on how often specific smells appear.

$$J(A, B) = \frac{\sum_{i=1}^{n} \min(A_i, B_i)}{\sum_{i=1}^{n} \max(A_i, B_i)} \tag{4}$$

Where A and B represent the frequencies of test smells in the LLM-generated and human-written test suites, respectively.

- **Euclidean Distance** We use this metric [58, 66] to compare the frequency distribution of test smells between LLM-generated and human-written test suites. This allows us to quantify how the distribution of various test smells differs between the two sets.

$$d(A, B) = \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2} \tag{5}$$

Where A and B are vectors representing the frequency of test smells in the respective test suites.

- **Hellinger Distance** We use the Hellinger Distance [24, 33] to compare the normalized frequency distributions of test smells across the LLM-generated and human-written test suites. This metric is particularly useful for understanding the divergence between test smell distributions across both types of test suites.

$$H(A, B) = \frac{1}{\sqrt{2}} \sqrt{\sum_{i=1}^{n} (\sqrt{A_i} - \sqrt{B_i})^2} \tag{6}$$

Where A and B are the probability distributions of test smell frequencies.

Each of these metrics provides a different lens through which we compare the LLM-generated and human-written test suites, enabling a multi-faceted analysis of how similar or divergent their test smell patterns are.

3.6.4 Non-Linear Statistical Tests. While previous studies by Bavota et al. [6] and Palomba et al.[45] used Pearson correlation (PMCC) to analyze relationships between test smells and system characteristics in both human-written and generated tests, PMCC only detects linear correlations with constant rates of change. Test smell occurrences, however, may follow non-linear patterns with threshold effects where quality shifts at specific breakpoints.

To capture these complex relationships, we complement PMCC with three non-linear approaches: i) Threshold-Based Analysis (Kruskal-Wallis test) [38] to identify stepwise changes in test smell occurrences; ii) Spearman's Rank Correlation [63] to detect monotonic but not strictly linear relationships; and iii) Mutual Information (MI) [14] to measure non-linear dependencies between variables when no clear monotonic trend exists.

- **Threshold-Based Analysis** Threshold-based analysis examines how test smell occurrences shift at specific breakpoints in key variables like model size, context length, and project complexity. Using the Kruskal-Wallis test [38], we identify stepwise or piecewise non-linear patterns rather than smooth transitions, such as comparing test smell frequency before and after a 4096-token context length threshold to detect significant quality changes.

$$H = \frac{12}{n(n+1)} \sum_{i=1}^{g} \frac{R_i^2}{n_i} - 3(n+1) \tag{7}$$

Where g is the number of groups, n is the total number of observations, n_i is the number of observations in group i, and R_i is the sum of ranks in group i.

- **Spearman's Rank Correlation** Spearman's rank correlation [63] detects monotonic relationships (consistently increasing or decreasing trends) between variables, capturing non-linear patterns that Pearson correlation would miss, such as when test smells increase sharply beyond certain LLM model sizes without following a linear progression.

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)} \tag{8}$$

Where d_i is the difference between the ranks of corresponding values, and n is the number of data points.

- **Mutual Information (MI)** Mutual Information (MI) [14] quantifies the general dependence between variables, detecting non-linear, non-monotonic relationships that correlation metrics miss, such as when Magic Number Test smells show complex dependencies on LLM model size.

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \log \left(\frac{p(x,y)}{p(x)p(y)} \right) \tag{9}$$

Where $p(x,y)$ is the joint probability distribution of X and Y, while $p(x)$ and $p(y)$ are the marginal probability distributions of X and Y.

3.7 Implementation and Configuration

We conducted our test smell analysis using Python 3.10 and Java 17. To detect smells, we relied on two complementary tools: *TsDetect v2.2*, which supports automated batch execution, and *JNose 2.2.0*, which operates through a web interface. Since JNose lacks CLI support, we developed an automation layer based on *Playwright v1.51.0*[9] and *BeautifulSoup v4.13.3*[10] to simulate UI interaction and parse the resulting CSV reports programmatically.

We relied on `ck`[11] 0.7.0 for extracting object-oriented metrics (e.g., CBO, RFC, DIT) and on `Lizard` 1.17.23 for complexity and method-level statistics (e.g., Cyclomatic Complexity, number of methods). These metrics were then used in correlation and distributional analyses to examine the relationship between software structure, generation settings, and test smell diffusion.

All experiments were executed on a machine equipped with an Intel Core i9-14900K CPU (32 threads, 6.0GHz), 64 GB RAM, and an NVIDIA RTX 5000 Ada GPU (32 GB VRAM). The full pipeline was fully automated to support scalable, reproducible analysis across all benchmarks.

4 EXPERIMENTAL RESULTS

4.1 [RQ1]: Prevalence and Comparison of Test Smells in LLM-vs. SBST-Generated Tests

[Experiment Design]: We conduct this comparison at two levels of test generation granularity: class-level and method-level, based on Benchmark 1 and Benchmark 2 (described in Section 3.3). To assess consistency between detectors, we compute the mean and standard deviation of absolute delta values per smell type, and highlight the top-5 largest deltas to reveal key divergences.

[Experiment Results]:

[9] https://github.com/microsoft/playwright
[10] https://www.crummy.com/software/BeautifulSoup/
[11] https://github.com/mauricioaniche/ck

Table 7. Test Smells Distribution by Model and Datasets from Benchmark 1 Detected by TsDetect and JNose.

(a) Detected by TsDetect

Model	Dataset	AR	CLT	CI	DfT	EmT	EH	GF	MG	RP	RA	SE	VT	ST	EaT	LT	DA	UT	IT	RO	MT	DpT
GPT 3.5-Turbo	Defects4J	46.94	3.35	0.10	0.00	31.64	19.26	4.57	0.89	0.00	1.08	18.78	0.00	0.65	41.16	48.67	6.97	35.16	0.00	01.01	99.78	0.00
	CMD	17.08	0.14	0.00	0.00	62.40	12.81	15.15	0.00	0.00	0.28	0.83	0.00	0.00	16.12	15.70	0.14	77.27	0.00	0.00	100.00	0.00
	SF110	48.53	2.14	0.06	0.00	17.96	12.91	07.09	2.44	0.05	1.98	20.39	0.00	0.30	49.82	51.73	8.41	28.70	0.02	2.70	99.79	0.00
	Average	37.51	1.87	0.05	0.00	37.33	14.99	8.93	1.11	0.01	1.11	13.33	0.00	0.31	35.70	38.70	5.17	47.04	0.01	1.23	99.85	0.00
GPT 4	CMD	22.80	0.48	0.00	0.00	27.55	19.95	18.53	0.00	0.24	0.48	1.19	0.00	0.00	27.79	32.54	0.48	50.83	0.24	0.00	98.34	0.00
Mistral 7B	Defects4J	54.54	8.49	0.20	0.00	5.81	15.68	0.85	0.20	0.13	0.91	13.00	0.00	0.85	44.68	56.04	17.57	7.84	0.07	0.13	92.95	0.00
	CMD	14.29	0.00	0.00	0.00	28.57	7.14	7.14	0.00	0.00	0.00	0.00	0.00	0.00	14.29	7.14	0.00	42.86	0.00	0.00	71.43	0.00
	SF110	41.51	4.51	0.06	0.00	7.78	10.43	2.20	0.23	0.56	1.69	13.99	0.00	0.17	44.16	44.90	7.73	12.92	0.06	0.23	91.88	0.00
	Average	36.78	4.33	0.08	0.00	14.05	11.08	3.39	0.14	0.23	0.86	8.99	0.00	0.34	34.37	36.02	8.43	21.20	0.04	0.12	85.42	0.00
Mixtral 8x7B	CMD	31.82	4.55	0.00	0.00	27.27	22.73	22.73	0.00	0.00	0.00	4.55	0.00	0.00	31.82	31.82	0.00	50.00	0.00	0.00	95.45	0.00
EvoSuite	Defects4J	38.24	0.00	0.00	0.00	6.90	93.10	0.00	4.44	0.00	0.00	10.71	0.00	0.00	62.95	82.18	6.73	0.38	0.00	4.34	100.00	0.00
	SF110	50.78	0.00	0.00	0.00	6.90	93.10	0.00	4.44	0.00	0.00	17.53	0.00	0.00	54.67	76.94	0.07	6.90	0.00	3.96	100.00	0.00
	Average	44.51	0.00	0.00	0.00	6.90	93.10	0.00	4.44	0.00	0.00	14.12	0.00	0.00	58.81	79.56	3.40	3.64	0.00	4.15	100.00	0.00

(b) Detected by JNose

Model	Dataset	AR	CLT	CI	DfT	EmT	EH	GF	MG	RP	RA	SE	VT	ST	EaT	LT	DA	UT	IT	RO	MT	DpT
GPT 3.5	Defects4J	71.15	1.92	0.00	0.00	30.77	15.38	5.77	1.92	0.00	1.92	21.15	1.92	1.92	40.38	57.69	3.85	7.69	0.00	1.92	28.85	0.00
	CMD	50.00	0.00	0.00	0.00	40.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	20.00	20.00	0.00	40.00	0.00	0.00	0.00	0.00
	SF110	84.62	3.85	0.00	0.00	15.38	7.69	3.85	3.85	0.00	0.00	19.23	3.85	0.00	50.00	69.23	11.54	15.38	0.00	3.85	34.62	0.00
	Average	68.59	1.92	0.00	0.00	28.72	7.69	3.21	1.92	0.00	0.64	13.46	1.92	0.64	36.79	48.97	5.13	21.02	0.00	1.92	21.16	0.00
GPT 4	CMD	100.00	0.00	0.00	0.00	0.00	0.00	50.00	0.00	0.00	0.00	0.00	0.00	0.00	50.00	50.00	0.00	0.00	0.00	0.00	50.00	0.00
Mistral 7B	Defects4J	83.33	0.00	0.00	0.00	16.67	8.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00	41.67	33.33	16.67	8.33	0.00	0.00	16.67	0.00
	SF110	100.00	6.25	0.00	0.00	0.00	6.25	0.00	0.00	0.00	0.00	12.50	0.00	0.00	43.75	31.25	18.75	0.00	0.00	0.00	18.75	0.00
	Average	91.67	3.12	0.00	0.00	8.34	7.29	0.00	0.00	0.00	0.00	6.25	0.00	0.00	42.71	32.29	17.71	4.17	0.00	0.00	17.71	0.00
Mixtral 8x7B	CMD	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
EvoSuite	Defects4J	55.56	0.00	0.00	0.00	16.67	66.67	0.00	0.00	0.00	0.00	16.67	5.56	0.00	38.89	66.67	0.00	0.00	0.00	0.00	27.78	0.00
	SF110	90.00	0.00	0.00	0.00	10.00	80.00	0.00	0.00	0.00	0.00	10.00	0.00	0.00	80.00	90.00	0.00	0.00	0.00	0.00	50.00	0.00
	Average	72.78	0.00	0.00	0.00	13.34	73.34	0.00	0.00	0.00	0.00	13.34	2.78	0.00	59.45	78.34	0.00	0.00	0.00	0.00	38.89	0.00

* AR: Assertion Roulette, CLT: Conditional Logic Test, CI: Constructor Initialization, DfT: Default Test, EmT: Empty Test, EH: Exception Handling, GF: General Fixture, MG: Mystery Guest, RP: Redundant Print, RA: Redundant Assertion, SE: Sensitive Equality, VT: Verbose Test, ST: Sleepy Test, EaT: Eager Test, LT: Lazy Test, DA: Duplicate Assert, UT: Unknown Test, IT: Ignored Test, RO: Resource Optimism, MT: Magic Number Test, DpT: Dependent Test.

Table 8. Test Smells Distribution from Benchmark 2 Detected by TsDetect and JNose.

(a) Detected by TsDetect

Model	AR	CLT	CI	DfT	EmT	ECT	EH	GF	MG	RP	RA	SE	VT	ST	EaT	LT	DA	UT	IT	RO	MT	DpT
GPT 3.5-Turbo	98.98	16.20	1.05	0.00	0.00	1.08	0.00	4.68	0.00	0.00	0.34	0.00	0.37	0.00	0.00	14.75	0.00	0.00	6.78	0.00	0.00	93.60
GPT 4	94.43	28.24	10.79	0.00	0.00	0.00	0.00	12.76	2.97	0.00	0.00	0.00	1.98	0.00	0.65	49.51	11.14	22.95	2.64	0.00	0.00	99.03
CodeLlama 13B	92.75	40.37	5.71	0.00	0.00	0.00	0.00	3.49	0.00	0.00	0.32	0.32	1.27	0.00	0.32	24.02	0.00	6.03	3.12	0.00	0.00	85.66
Average	95.39	28.27	5.85	0.00	0.00	0.36	0.00	6.98	0.99	0.00	0.22	0.11	1.21	0.00	0.32	29.43	3.71	9.66	4.18	0.00	0.00	92.76

(b) Detected by JNose

Model	AR	CLT	CI	DfT	EmT	ECT	EH	GF	MG	RP	RA	SE	VT	ST	EaT	LT	DA	UT	IT	RO	MT	DpT
GPT 3.5-Turbo	94.85	3.09	0.00	0.00	3.09	8.25	0.00	0.00	0.00	1.03	0.00	1.03	0.00	0.00	17.53	0.00	0.00	9.28	0.00	0.00	45.36	0.00
GPT 4	95.96	20.20	0.00	0.00	0.00	12.12	0.00	7.07	0.00	0.00	0.00	3.03	38.38	1.01	25.25	19.19	11.11	4.04	0.00	0.00	59.60	0.00
CodeLlama 13B	98.98	24.49	0.00	0.00	0.00	6.12	0.00	0.00	0.00	1.02	1.02	2.04	9.18	0.00	15.31	1.02	17.35	5.10	0.00	0.00	66.33	0.00
Average	96.60	15.93	0.00	0.00	1.03	8.83	0.00	2.36	0.00	0.68	0.34	2.03	15.85	0.34	19.36	6.74	9.49	6.14	0.00	0.00	57.10	0.00

* AR: Assertion Roulette, CLT: Conditional Logic Test, CI: Constructor Initialization, DfT: Default Test, EmT: Empty Test, ECT: Exception Catching Throwing, EH: Exception Handling, GF: General Fixture, MG: Mystery Guest, RP: Redundant Print, RA: Redundant Assertion, SE: Sensitive Equality, VT: Verbose Test, ST: Sleepy Test, EaT: Eager Test, LT: Lazy Test, DA: Duplicate Assert, UT: Unknown Test, IT: Ignored Test, RO: Resource Optimism, MT: Magic Number Test, DpT: Dependent Test.

Prevalence of Test Smells. Assertion Roulette (AR) emerges as the most dominant smell across all settings (Tables 7–8). In Benchmark 1 (class level), AR affects 38–49% of LLM-generated suites according to TsDetect, and 50–100% according to JNose, with EvoSuite also showing high prevalence (44–73% depending on the tool). In Benchmark 2 (method level), AR is detected at consistently

high rates across tools (92.8–99%), confirming it as the most pervasive smell across models and granularities.

Finding 1: *Assertion Roulette is pervasive across all LLMs and granularities, exceeding 90% prevalence at method level and remaining high at class level, with EvoSuite exhibiting similar vulnerability.*

Beyond AR, LLM-generated tests also suffer from recurring weaknesses in logic depth and exception coverage. At class level (Table 7), Lazy Test (LT) is widespread, ranging from 7–56% across LLMs, while EvoSuite shows even higher prevalence (77–82%). Unknown Test (UT) reaches up to 77% in LLMs but remains almost absent in EvoSuite (0–7%), owing to its structurally enforced assertions. Exception Handling (EH) is similarly underrepresented in LLM outputs (7–23%), in stark contrast to EvoSuite (93%). Method-level generation (Table 8) alleviates some of these issues: LT decreases to 15–50% (TsDetect) and below 20% (JNose), while UT falls to 0–23% (TsDetect) and 4–9% (JNose). However, EH remains virtually absent across all LLMs at method level, suggesting that finer-grained generation improves modularity and assertion clarity but fails to address exception scenarios.

Finding 2: *LLM-generated tests frequently exhibit Lazy Test, Unknown Test, and missing Exception Handling. While method-level generation reduces LT and UT prevalence, exception coverage remains absent—contrasting sharply with SBST (EvoSuite).*

Distribution Across Smells and Granularity. Smell distribution differs by test granularity. While Assertion Roulette (AR) dominates at both levels, Lazy Test (LT) and Exception Handling (EH) are less frequent at method level: LT drops from 7–56% at class level (LLMs; Table 7) to 15–50% (TsDetect) and 0–19% (JNose) at method level (Table 8), and EH falls to 0% across LLMs in Benchmark 2. Cross-tool disagreement (Tables 9 and 11) further shapes the observed distributions: AR is highly consistent at method level ($\Delta = 1.21$), but several fine-grained smells show large deltas between TsDetect and JNose. In Benchmark 1 (LLMs), the largest deltas occur for Magic Number Test (MT, $\Delta = 72.55$), Unknown Test (UT, $\Delta = 35.97$), and Empty Test (EmT, $\Delta = 17.28$) (Table 9). In Benchmark 2, the largest deltas are for Dependent Test (DpT, $\Delta = 57.10$), Duplicate Assert (DA, $\Delta = 22.69$), and Lazy Test (LT, $\Delta = 19.04$), followed by Sleepy Test (ST, $\Delta = 14.64$) and Conditional Logic Test (CLT, $\Delta = 12.34$).

Finding 3: *Smell distribution varies with granularity and detector: AR remains dominant across levels, but LT and EH are less frequent at method level, while fine-grained smells (e.g., MT, DpT, DA) exhibit large cross-tool variability (up to $\Delta = 72.55$), as shown in Tables 9 and 11.*

Comparison with EvoSuite. EvoSuite shares AR and LT with LLMs but differs on UT and EH. At class level (Benchmark 1), UT reaches 8–77% in LLMs yet remains 0–7% in EvoSuite; EH is 7–23% in LLMs versus 93% in EvoSuite according to TsDetect (66–80% in JNose) (Table 7). AR is high for both (LLMs: 38–49% TsDetect and 50–100% JNose; EvoSuite: 38–51% TsDetect and 56–90% JNose). Cross-tool variability (Tables 10) is also lower for EvoSuite: the mean delta is 6.51 with std. dev. 14.06 in Benchmark 1, compared to 8.64 and 17.68 for LLMs; at method level (Benchmark 2), LLM tests still show notable deltas for several smells (e.g., DpT, DA, LT) even though AR is stable across tools (Tables 9-10).

Finding 4: *Compared to EvoSuite, LLM-generated tests not only exhibit higher Unknown Test and weaker Exception Handling (while both share AR and LT), but also induce greater cross-tool variability between TsDetect and JNose (mean Δ=8.64 vs. 6.51; std. dev.=17.68 vs. 14.06). The largest Δ values for LLMs occur on MT and UT in Benchmark 1 and DpT, DA, and LT in Benchmark 2 (Tables 7–11).*

Table 9. Delta Values between TsDetect and JNose for Test Smell Distributions (Benchmark 1 and 2)

Benchmark	AR	CLT	CI	DfT	EmT	EH	GF	MG	RP	RA	SE	VT	ST	EaT	LT	DA	UT	IT	RO	MT	DpT
Benchmark 1 (LLM Avg)	32.84	1.55	0.03	0.00	17.28	13.44	0.10	0.17	0.12	0.45	2.09	0.48	0.00	0.04	1.96	2.19	35.97	0.07	0.14	72.55	0.00
Benchmark 1 (EvoSuite)	28.27	0.00	0.00	0.00	6.44	19.76	0.00	4.44	0.00	0.00	0.78	2.78	0.00	0.64	1.22	3.40	3.64	0.00	4.15	61.11	0.00
Benchmark 2	1.21	12.34	5.85	0.00	1.03	8.47	0.00	4.62	0.99	0.68	0.12	1.92	14.64	0.34	19.04	22.69	5.78	3.52	4.18	0.00	57.10

Table 10. Mean and Standard Deviation of Delta Values between TsDetect and JNose for Test Smell Distributions

Benchmark	Mean Delta	Std Deviation
Benchmark 1 - LLM	8.641	17.678
Benchmark 1 - EvoSuite	6.506	14.055
Benchmark 2	7.834	12.750

Table 11. Top-5 Largest Deltas between TsDetect and JNose

B1-LLM			B1-EvoSuite			B2-LLM		
Smell	Δ	Rk	Smell	Δ	Rk	Smell	Δ	Rk
MT	72.55	1	MT	61.11	1	DpT	57.10	1
UT	35.97	2	AR	28.27	2	DA	22.69	2
AR	32.84	3	EH	19.76	3	LT	19.04	3
EmT	17.28	4	EmT	6.44	4	ST	14.64	4
EH	13.44	5	MG	4.44	5	CLT	12.34	5

Summary of **RQ1:** LLM-generated tests systematically suffer from smells such as Assertion Roulette, Lazy Test, Unknown Test, and missing Exception Handling. While method-level generation alleviates some issues (lower LT and UT), exception handling remains absent, unlike EvoSuite which consistently covers exceptions. Smell distribution is strongly influenced by test granularity and the detection tool: AR dominates across levels, but fine-grained smells (e.g., MT, DpT, DA) show extreme cross-tool variability. Compared to EvoSuite, LLM outputs amplify UT, underuse EH, and induce greater instability between detectors, underscoring both the originality and fragility of LLM-generated test structures.

4.2 [RQ2]: Test Smell Co-occurrence and Prompt Influence in LLM vs. SBST Suites

[Experiment Design]: We analyze test smell co-occurrence using two benchmarks. Benchmark 1 varies prompting strategies (ZSL, FSL, CoT, ToT, GToT) on class-level test suites, while Benchmark 2 varies contextual information (Self-Contained, Simple, Full) on method-level test cases. After measuring individual smell prevalence, we compute normalized co-occurrence matrices for each benchmark, capturing the frequency of smell pairs across generation settings. Analyses are performed independently with TsDetect and JNose to assess detector consistency. We quantify inter-tool disagreement by calculating the mean and standard deviation of delta values between the two matrices, highlighting the five smell pairs with the largest disagreements. For comparison, we apply the same analysis to EvoSuite-generated test suites from Benchmark 1, enabling assessment of smell interaction patterns across LLM and SBST paradigms.

[Experiment Results]:

Influence of Prompt Engineering Techniques. In Benchmark 1, prompting style measurably affects smell prevalence. For *Assertion Roulette (AR)*, TsDetect reports rates from **20.94%** (GToT) to **54.77%** (ZSL), with JNose generally higher; structured reasoning prompts (GToT, CoT) *moderately* reduce AR (Table 12). However, other smells remain sensitive to prompt design: *Lazy Test (LT)* spans

large intervals across prompts (e.g., **26.44%–73.50%**), indicating that some structured prompts can also correlate with more verbose or redundant logic.

Finding 5: *Structured reasoning prompts (e.g., GToT, CoT) can* moderately *reduce AR at class level, but other smells (e.g., LT) remain highly prompt-sensitive, showing wide variance across strategies (Table 12).*

Effect of Context Variation. In Benchmark 2, varying the amount/structure of context (self-contained, simple, full) has limited effect on the main smells: AR remains **consistently above 90%** across tools (TsDetect **94.78–96.23%**, JNose **91.74–93.14%**), *Dependent Test (DpT)* exceeds **95%** in TsDetect regardless of context, and *Unknown Test (UT)* persists across levels. LT is generally lower than in Benchmark 1, yet *Exception Handling (EH)* remains absent (Tables 13, 8).

Finding 6: *Context granularity has limited impact on smells at method level: AR stays* > 90%, *DpT remains high, UT persists, and EH is not recovered, indicating that more input context alone does not mitigate key smell patterns (Table 13).*

Co-occurrence Patterns (Tools & Granularity). Co-occurrence structures differ by detector and granularity. With TsDetect at class level (Benchmark 1), we observe dense structural clusters—*GF, RO, RA, SE, AR*—often exceeding **90%**, sometimes reaching **100%**; JNose highlights more semantic pairings such as *LT+DA* and *UT+DA* (up to **100%**) (Figures 2a–2b). At method level (Benchmark 2), clusters become denser (e.g., *CLT, CI, ECT* with TsDetect; *VT* as hub with many smells in JNose) (Figures 3a–3b). Cross-tool disagreement on co-occurrence increases with granularity: mean Δ rises from **0.287** (σ = **0.349**) at class level to **0.323** (σ = **0.370**) at method level, with top-5 pairs reaching Δ = **1.00** (Tables 14, 15).

Finding 7: *Co-occurrence depends on both detector and granularity: TsDetect emphasizes clusters centered on assertions and fixtures (e.g., GF, RO, RA, SE, AR), whereas JNose surfaces semantically grounded pairings (e.g., LT+DA, UT+DA). Disagreement grows at method level (mean* Δ *from 0.287 to 0.323), with several pairs reaching maximum divergence (*$\Delta = 1.00$*).*

EvoSuite exhibits distinctive co-occurrence trends compared to LLMs. With TsDetect, *Empty Test (EmT)* often anchors clusters that include *AR, EaT, LT, DA* (frequently up to **100%**); JNose shows strong exception-related groupings (e.g., *EH+EaT*, *EH+MT*, approaching **99–100%**). These clusters largely reflect EvoSuite's limited assertion synthesis and omission of exception handling. In contrast, LLM-generated clusters are shaped by verbose or repetitive logic (e.g., *VT* as a hub; *LT+DA*, *UT+DA*), reflecting tendencies toward redundancy and loosely scoped assertions. Together, these differences highlight that EvoSuite clusters emerge from omission of logic and exception handling, whereas LLM clusters arise from overproduction and repetition of assertions (Figures 2c–2d).

Finding 8: *EvoSuite co-occurrence is anchored by Empty Test and exception-related patterns, reflecting the absence of logic and exception handling, whereas LLMs accumulate clusters driven by redundancy, verbosity, and weakly specified logic (e.g., VT, LT+DA, UT+DA), evidencing distinct generative biases.*

Relation to Prior Work. These findings align with prior observations by Panichella et al. [49], who noted moderate disagreement between detectors on EvoSuite- and JTexpert-generated tests. Our study extends this perspective by incorporating LLM-generated outputs and introducing granularity as a key factor. Unlike prior rule-based or manually crafted tests, LLM-generated tests—especially at the method level—exacerbate detector divergence, raising concerns about the generalizability of existing detection tools to modern generation paradigms.

Table 12. Test Smells Distribution by Prompt Techniques from Benchmark 1 Detected by TsDetect and JNose.

(a) Detected by TsDetect

Prompt	AR	CLT	CI	DfT	EmT	EH	GF	MG	RP	RA	SE	VT	ST	EaT	LT	DA	UT	IT	RO	MT	DpT
CoT	31.34	3.69	0.02	0.00	23.83	20.60	4.55	0.43	0.22	0.69	9.91	0.00	0.36	28.35	50.96	7.91	39.43	0.00	0.44	98.04	0.00
FSL	44.91	4.63	0.09	0.00	10.37	13.97	18.31	0.61	0.31	1.58	14.99	0.00	0.36	45.83	40.31	9.19	35.78	0.05	0.69	95.83	0.00
ToT	39.77	3.76	0.10	0.00	21.19	27.64	7.94	0.46	0.08	0.94	12.89	0.00	0.41	39.30	55.58	7.23	40.21	0.01	0.55	98.51	0.00
GToT	20.94	0.79	0.02	0.00	37.43	15.68	11.60	0.38	0.00	0.71	3.59	0.00	0.06	22.43	26.44	1.46	50.94	0.14	0.42	91.82	0.00
ZSL	54.77	4.60	0.09	0.00	10.48	17.82	22.77	0.74	0.29	0.85	14.38	0.00	0.51	43.15	48.75	8.78	20.32	0.06	0.73	99.72	0.00
Average	38.09	3.45	0.06	0.00	20.84	19.13	12.99	0.52	0.17	0.96	11.09	0.00	0.33	35.81	44.12	6.83	37.72	0.05	0.56	96.67	0.00

(b) Detected by JNose

Prompt	AR	CLT	CI	DfT	EmT	EH	GF	MG	RP	RA	SE	VT	ST	EaT	LT	DA	UT	IT	RO	MT	DpT
CoT	66.25	22.50	0.00	0.00	30.00	0.00	2.50	0.00	0.00	0.00	26.25	0.00	1.25	42.50	62.50	43.75	3.75	0.00	0.00	35.00	0.00
FSL	92.13	0.00	0.00	0.00	7.87	0.00	8.33	0.00	0.00	0.00	32.87	0.00	0.00	59.72	59.72	15.74	11.11	0.00	0.00	35.65	0.00
ToT	100.00	0.00	0.00	0.00	0.00	0.00	6.50	2.50	0.00	0.00	11.50	0.00	0.00	53.67	73.50	3.33	24.00	0.00	2.50	38.83	0.00
GToT	65.83	0.00	0.00	0.00	35.83	0.00	0.00	0.00	0.83	0.00	6.67	0.00	0.00	32.50	24.17	1.67	5.00	0.00	0.00	11.67	0.00
ZSL	75.83	2.50	0.00	0.00	13.33	0.00	2.50	5.00	0.00	0.00	7.50	0.00	0.00	20.00	30.00	12.50	24.17	0.00	5.00	27.50	0.00
Average	80.01	5.00	0.00	0.00	17.41	0.00	3.97	1.50	0.17	0.00	16.96	0.00	0.25	41.68	49.98	15.40	13.61	0.00	1.50	29.73	0.00

Table 13. Test Smells Distribution by Prompt among all projects from Benchmark 2 Detected by TsDetect and JNose

(a) Detected by TsDetect

Prompt	AR	CLT	CI	DfT	EmT	ECT	EH	GF	MG	RP	RA	SE	VT	ST	EaT	LT	DA	UT	IT	RO	MT	DpT
SCC	94.78	29.55	7.14	0.00	0.00	0.34	0.00	6.25	0.33	0.00	0.34	0.32	1.63	0.00	0.00	28.80	3.63	11.08	4.68	0.00	0.00	95.04
SC	95.16	31.44	5.56	0.00	0.00	0.37	0.00	8.46	2.00	0.00	0.32	0.00	1.67	0.00	0.65	32.52	3.67	10.21	4.23	0.00	0.00	95.34
FC	96.23	23.82	4.85	0.00	0.00	0.37	0.00	6.23	0.64	0.00	0.00	0.00	0.32	0.00	0.32	26.96	3.85	7.69	3.64	0.00	0.00	87.90
Average	95.39	28.27	5.85	0.00	0.00	0.36	0.00	6.98	0.99	0.00	0.22	0.11	1.21	0.00	0.32	29.43	3.71	9.66	4.18	0.00	0.00	92.76

(b) Detected by JNose

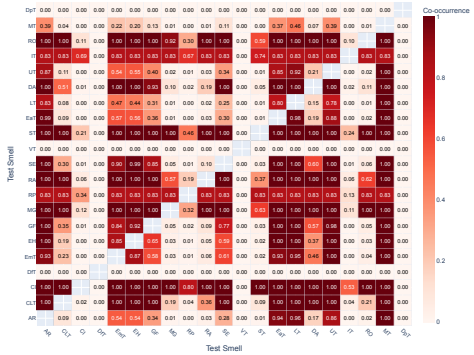
Prompt	AR	CLT	CI	DfT	EmT	ECT	EH	GF	MG	RP	RA	SE	VT	ST	EaT	LT	DA	UT	IT	RO	MT	DpT
SCC	93.14	7.77	0.00	0.00	0.43	2.77	0.00	0.39	0.00	0.43	0.37	1.94	7.83	0.00	8.84	3.96	4.19	3.33	0.00	0.00	43.62	0.00
SC	91.74	5.68	0.00	0.00	0.47	5.18	2.01	2.01	0.00	0.41	0.00	2.10	9.28	0.40	16.66	4.42	5.68	3.57	0.00	0.00	42.86	0.00
FC	93.06	12.63	0.00	0.00	0.45	5.49	0.00	0.44	0.00	0.00	0.00	0.00	9.21	0.44	15.35	4.58	3.83	2.79	0.00	0.00	46.32	0.00
Average	92.65	8.69	0.00	0.00	0.45	4.48	0.67	0.95	0.00	0.28	0.12	1.35	8.77	0.28	13.62	4.32	4.57	3.23	0.00	0.00	44.27	0.00

* SCC: Self-Contained Context, SC: Simple Context, FC: Full Context, AR: Assertion Roulette, CLT: Conditional Logic Test, CI: Constructor Initialization, DfT: Default Test, EmT: Empty Test, ECT: Exception Catching Throwing, EH: Exception Handling, GF: General Fixture, MG: Mistery Guest, RP: Redundant Print, RA: Redundant Assertion, SE: Sensitive Equality, VT: Verbose Test, ST: Sleepy Test, EaT: Eager Test, LT: Lazy Test, DA: Duplicate Assert, UT: Unknown Test, IT: Ignored Test, RO: Resource Optimism, MT: Magic Number Test, DpT: Dependent Test.

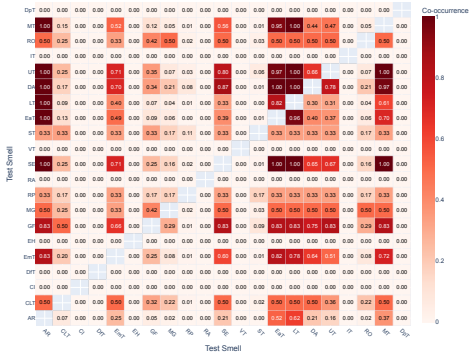
Table 14. Mean and Standard Deviation of Delta Values between TsDetect and JNose for Test Smell Co-occurrence

Benchmark	Mean Delta	Std Deviation
Benchmark 1	0.287	0.349
Benchmark 2	0.323	0.370

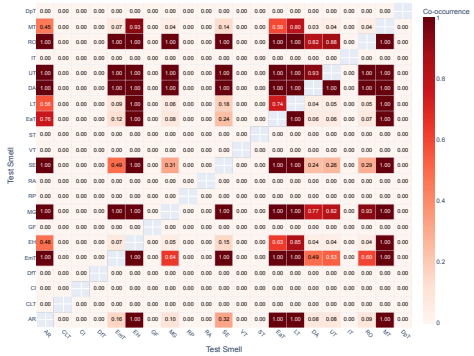
Summary of **RQ2:** Test smell co-occurrence in LLM-generated suites is influenced more by prompting strategies than by context granularity: reasoning-based prompts (e.g., CoT, GToT) moderately reduce AR at class level, but key smells (LT, UT, DpT) persist even with richer context. Detector choice further shapes observed clusters—TsDetect emphasizing assertion- and fixture-centered structures, while JNose surfaces semantic pairings such as LT+DA or UT+DA—with disagreement peaking at method level. Compared to EvoSuite, which clusters around omissions (Empty Test, missing exceptions), LLMs form redundancy- and verbosity-driven clusters, highlighting distinct generative biases.



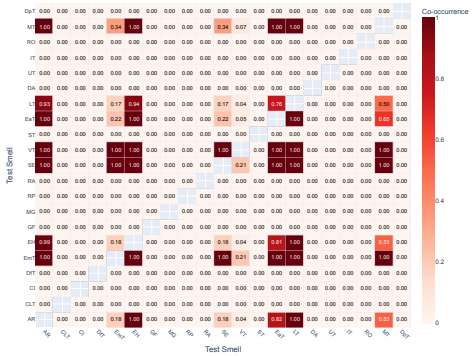
(a) TsDetect Co-occurrence Matrix (Benchmark 1 - LLM)



(b) JNose Co-occurrence Matrix (Benchmark 1 - LLM)



(c) TsDetect Co-occurrence Matrix (Benchmark 1 - EvoSuite)



(d) JNose Co-occurrence Matrix (Benchmark 1 - EvoSuite)

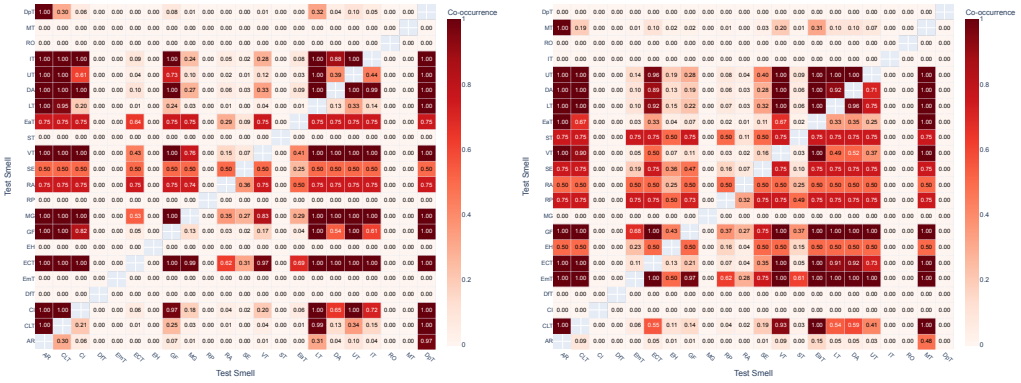
Fig. 2. Test Smell Co-occurrence Matrices detected by TsDetect and JNose for Benchmark 1 (LLMs and EvoSuite).

Table 15. Top-5 Largest Deltas between TsDetect and JNose for Co-occurrence Matrices

Benchmark 1 - LLM					Benchmark 1 - EvoSuite					Benchmark 2				
Smell Pair	**TsDetect**	**JNose**	**Delta**	**Rank**	**Smell Pair**	**TsDetect**	**JNose**	**Delta**	**Rank**	**Smell Pair**	**TsDetect**	**JNose**	**Delta**	**Rank**
RA-EH	1.00	0.00	1.00	1	SE-VT	0.00	1.00	1.00	1	EmT-EaT	0.00	1.00	1.00	1
CI-EmT	1.00	0.00	1.00	2	RO-MG	1.00	0.00	0.93	2	VT-IT	1.00	0.00	1.00	2
CI-RO	1.00	0.00	1.00	3	UT-RO	1.00	0.00	0.88	3	DA-MT	0.00	1.00	1.00	3
CLT-EH	1.00	0.00	1.00	4	UT-MG	1.00	0.00	0.82	4	MG-UT	1.00	0.00	1.00	4
RA-CLT	1.00	0.00	1.00	5	DA-MG	1.00	0.00	0.77	5	MG-DA	1.00	0.00	1.00	5

4.3 [RQ3]: Impact of Software Attributes and LLM Parameters on Test Smell Prevalence

[Experiment Design]: We analyze test smell prevalence in LLM-generated tests (Benchmarks 1 and 2) and EvoSuite-generated tests (Benchmark 1). As explanatory variables, we consider both software attributes—LOC, number of classes/methods, cyclomatic complexity, CBO, RFC, and DIT—and LLM parameters such as model size, context window, temperature, and top-p. To capture relationships between these features and smell prevalence, we adopt a hybrid correlation strategy: Pearson's correlation (PMCC) is applied across all benchmarks to measure linear associations, while Spearman's rank correlation, Mutual Information, and Kruskal–Wallis tests are selectively used to detect monotonic and non-linear effects, restricted to Benchmark 1 where variance is



(a) TsDetect Co-occurrence Matrix (b) JNose Co-occurrence Matrix

Fig. 3. Test Smell Co-occurrence Matrices for Benchmark 2 detected by TsDetect and JNose.

sufficient. Non-linear analyses are omitted when variable distributions are invariant or statistically unsuitable (e.g., method-level tests in Benchmark 2). Invariant smells and features are filtered to ensure statistical validity. This design enables robust yet scalable correlation analysis across diverse generation settings.

[Experiment Results]:

Non-Linear Influence Analysis (Benchmark 1). We first assess whether software attributes or LLM parameters exert non-linear effects on smell prevalence. Using Kruskal–Wallis and Mutual Information (MI), we analyzed LLM-generated suites in Benchmark 1, which exhibits sufficient variance across features. Benchmark 2 was excluded from non-linear analysis due to highly invariant distributions (e.g., AR >90%, DpT >95%). Results (Tables 16, 17) show no significant non-linear effects: all p-values ≥ 0.1, and MI scores were zero or negligible ($\sim 10^{-16}$, attributable to floating-point noise).

Finding 9: *Non-linear analyses (Kruskal–Wallis and MI) show no measurable influence of software attributes (size, complexity, coupling) or LLM-specific parameters (model size, context length, decoding settings) on test smell prevalence in Benchmark 1.*

Impact of Software Attributes (Benchmark 1). Correlation analysis (Spearman and Pearson) shows that project size and complexity/coupling exert distinct influences (Figures 4, 5a, 5b). Larger systems (more classes, methods, LOC) amplify redundant and superficial smells. TsDetect reports strong positive correlations $\rho \approx +0.84$ for AR, RA, SE, EaT, LT, while JNose highlights MG, RO, and VT ($\rho \approx +0.58$). Higher complexity and coupling (CBO, RFC, CyCo) shift smell prevalence toward architectural flaws: TsDetect reports $\rho \geq +0.89$ for EH, GF, CLT, while JNose corroborates with EmT and CLT. Verbosity (VT) decreases slightly, suggesting a transition from shallow redundancy to deeper structural issues.

Finding 10: *Software attributes shape smell prevalence: large-scale systems amplify redundant smells (AR, RA, MG), while higher complexity and coupling induce architectural flaws (EH, GF, CLT). This suggests a progression from superficial to structural defects as systems grow in size and interdependence.*

Influence of LLM Parameters. LLM generation settings modulate smell behavior across both benchmarks (Figures 4a–4b, 6a–6b). With Benchmark 1, longer contexts reduce assertion-level

smells: TsDetect reports $r \in [-0.91, -0.51]$ for AR, RA, SE, DA (Figures 4a); JNose shows $\rho \approx -0.94$ for CLT, EmT, EH, SE, DA, UT (Figures 4b). At method level with Benchmark 2 (Figures 6a–6b), longer contexts instead inflate verbosity smells: GF, MG, DA, UT, LT, VT all reach $r \geq 0.97$ (both detectors), while AR decreases modestly ($r \approx -0.26$). Benchmark 1 shows a trade-off: higher top-p reduces CLT ($\rho \approx -0.89$ in TsDetect) but increases EmT/MT ($\rho \approx +0.89$). Benchmark 2 highlights detector divergence: TsDetect finds top-p increases AR ($r = +0.71$) while JNose shows the opposite ($r = -0.97$); both agree DpT grows with top-p ($r \approx +0.91$). Model size effects are mild in Benchmark 1, but JNose highlights strong positives for GF ($\rho \approx +0.95$) and moderate rises for LT/MT ($\rho \approx +0.80$). In Benchmark 2, larger models amplify verbosity smells (GF, UT, VT) with $r \geq 0.97$.

Finding 11: *LLM parameters directly govern smell variability: constrained decoding (short context, low top-p) produces brittle and repetitive tests (CLT, EmT, UT), while permissive settings (long context, high top-p) suppress assertion smells but may inflate verbosity and ambiguous logic. These effects intensify at method level, where model size and context length linearly amplify redundancy-oriented smells (GF, MG, DA, UT, LT, VT; r up to 1.00).*

Comparative Analysis: LLMs vs. EvoSuite (Benchmark 1). In contrast to LLMs, EvoSuite exhibits highly deterministic correlation profiles (Figures 5c, 5d): $r = \pm 1.0$ for most attributes. Assertion smells (AR, EH, MT) correlate negatively with size but positively with complexity, while SE, VT, and EmT follow the opposite trend. Several smells (CLT, GF, DpT, RO) are absent altogether. These patterns reflect EvoSuite's rigid, template-driven generation: predictable but lacking adaptability compared to LLMs.

Finding 12: *EvoSuite exhibits deterministic correlations ($r = \pm 1.0$) between smell prevalence and software attributes, reflecting rigid template-driven rules. LLMs instead show variance across $-0.9 \leq r \leq +1.0$, revealing greater variability but less stability.*

Tool Sensitivity and Stochastic Smells. Some smells—such as RP, MT, IT, and UT—exhibit weak or inconsistent correlations ($|r| < 0.4$), fluctuating across detectors and benchmarks. In Benchmark 2 and EvoSuite, many are absent altogether, suggesting structural homogenization or detector blind spots. JNose tends to capture fine-grained semantic noise (e.g., VT, ECT), while TsDetect favors rigid syntactic patterns. These inconsistencies suggest that certain smells emerge stochastically or evade systematic modeling.

Finding 13: *Some smells (RP, MT, IT, UT) emerge stochastically or remain undetected, highlighting detector-specific blind spots and the limitations of current smell modeling.*

Table 16. Summary of Kruskal-Wallis Test Results on Benchmark 1 (LLM-Generated Test Suites)

Tool	Total Comparisons	#p < 0.01	0.01 ≤ p < 0.05	0.05 ≤ p ≤ 0.1	0.1 < p < 0.5	0.5 ≤ p ≤ 1.0
TsDetect	180	0	0	0	120	60
JNose	160	0	0	0	108	52

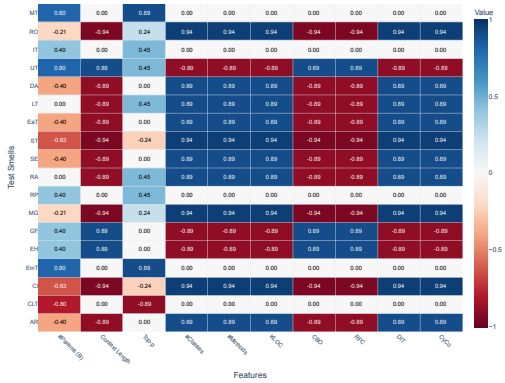
Table 17. Mutual Information (MI) Results on Benchmark 1 LLM-Generated Tests.

(a) Detected by TsDetect

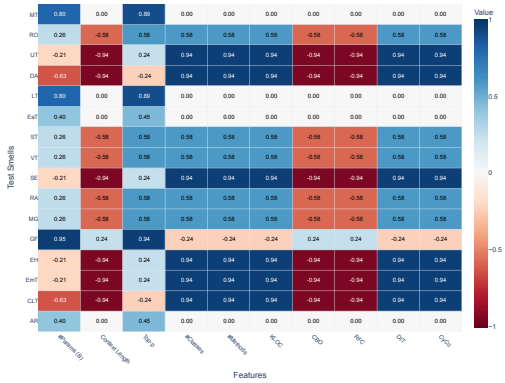
	AR	CLT	MG	EaT	UT	IT	RO	MT
#Params (B)	-	2.22e-16	-	-	-	-	-	-
Context Length	2.22e-16	-	1.11e-16	2.22e-16	1.11e-16	1.11e-16	1.11e-16	2.22e-16
Top-p	2.22e-16	-	1.11e-16	2.22e-16	1.11e-16	1.11e-16	1.11e-16	2.22e-16
#Classes	2.22e-16	-	1.11e-16	2.22e-16	1.11e-16	1.11e-16	1.11e-16	2.22e-16
#Methods	2.22e-16	-	1.11e-16	2.22e-16	1.11e-16	1.11e-16	1.11e-16	2.22e-16
KLOC	2.22e-16	-	1.11e-16	2.22e-16	1.11e-16	1.11e-16	1.11e-16	2.22e-16
CBO	2.22e-16	-	1.11e-16	2.22e-16	1.11e-16	1.11e-16	1.11e-16	2.22e-16
RFC	2.22e-16	-	1.11e-16	2.22e-16	1.11e-16	1.11e-16	1.11e-16	2.22e-16
DIT	2.22e-16	-	1.11e-16	2.22e-16	1.11e-16	1.11e-16	1.11e-16	2.22e-16
CyCo	2.22e-16	-	1.11e-16	2.22e-16	1.11e-16	1.11e-16	1.11e-16	2.22e-16

(b) Detected by JNose

	CLT	GF	MG	RA	VT	ST	RO
#Params (B)	2.22e-16	-	-	-	-	-	-
Context Length	-	1.11e-16	1.11e-16	1.11e-16	1.11e-16	1.11e-16	1.11e-16
Top-p	-	1.11e-16	1.11e-16	1.11e-16	1.11e-16	1.11e-16	1.11e-16
#Classes	-	1.11e-16	1.11e-16	1.11e-16	1.11e-16	1.11e-16	1.11e-16
#Methods	-	1.11e-16	1.11e-16	1.11e-16	1.11e-16	1.11e-16	1.11e-16
KLOC	-	1.11e-16	1.11e-16	1.11e-16	1.11e-16	1.11e-16	1.11e-16
CBO	-	1.11e-16	1.11e-16	1.11e-16	1.11e-16	1.11e-16	1.11e-16
RFC	-	1.11e-16	1.11e-16	1.11e-16	1.11e-16	1.11e-16	1.11e-16
DIT	-	1.11e-16	1.11e-16	1.11e-16	1.11e-16	1.11e-16	1.11e-16
CyCo	-	1.11e-16	1.11e-16	1.11e-16	1.11e-16	1.11e-16	1.11e-16



(a) Spearman Correlation results from TsDetect (Benchmark 1.)



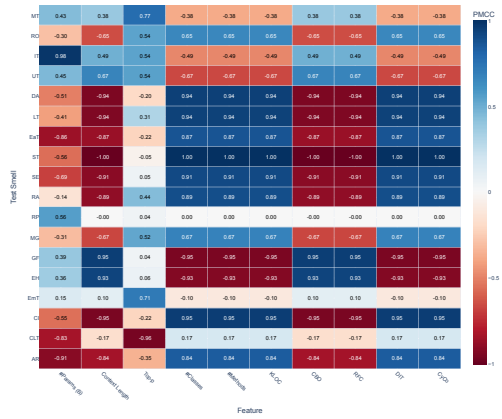
(b) Spearman Correlation results from JNose (Benchmark 1.)

Fig. 4. Spearman Correlation results between Project and LLM Characteristics and Test Smell Presence.

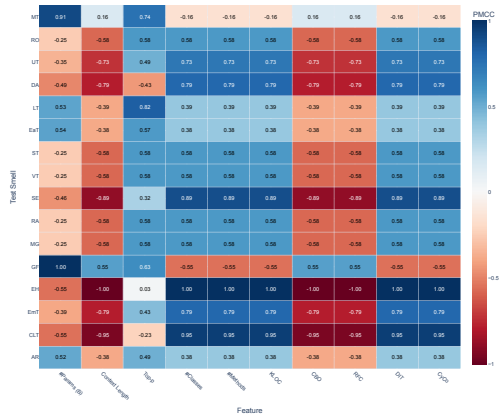
Summary of **RQ3:** Non-linear analyses show no significant effects of software or LLM parameters on smell prevalence (Finding 9). Instead, linear correlations reveal that project size amplifies redundant smells, while higher complexity and coupling induce architectural flaws (Finding 10). LLM parameters strongly modulate variability: constrained decoding (short context, low top-p) yields brittle and repetitive tests, whereas permissive settings reduce assertion smells but inflate verbosity and ambiguity, with effects intensifying at method level (Finding 11). Compared to LLMs, EvoSuite shows rigid template-driven correlations ($r = \pm 1.0$), producing predictable but shallow patterns (Finding 12). Finally, some smells arise stochastically or remain undetected, reflecting detector blind spots and current modeling limits (Finding 13).

4.4 [RQ4]: Similarity of test smell patterns in LLM-generated tests to human-written tests.

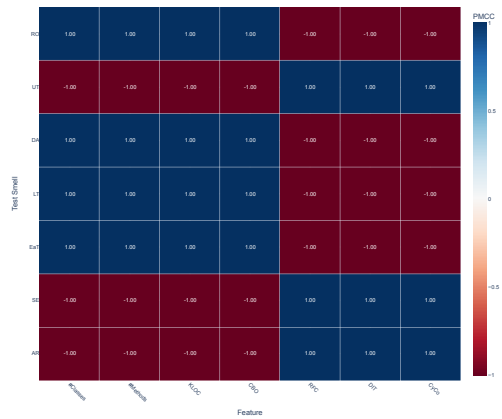
[Experiment Design]: We assess the structural similarity between LLM-generated (Benchmark 1) and human-written test suites (Defects4J, SF110, CAT-LM) to evaluate potential data leakage and overlap in smell patterns. Our comparison involves two phases. First, we measure distributional similarity using Cosine and Jaccard indices for type overlap, and Euclidean and Hellinger distances for frequency differences across smell distributions. Second, we analyze co-occurrence and correlation patterns through Pearson and Spearman correlations between mean smell vectors, complemented by Mutual Information and Kruskal–Wallis tests to capture non-linear and group-based differences.



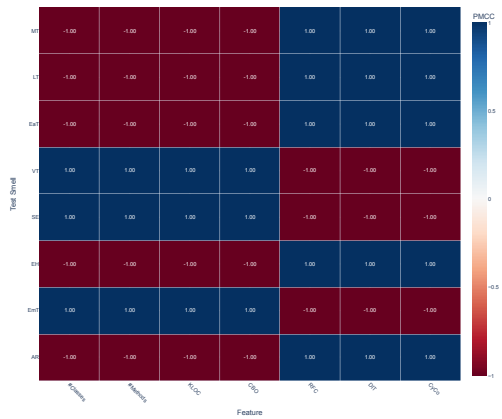
(a) PMCC result from TsDetect (Benchmark 1-LLM).



(b) PMCC result from JNose (Benchmark 1-LLM).



(c) PMCC result from TsDetect (Benchmark 1-EvoSuite).



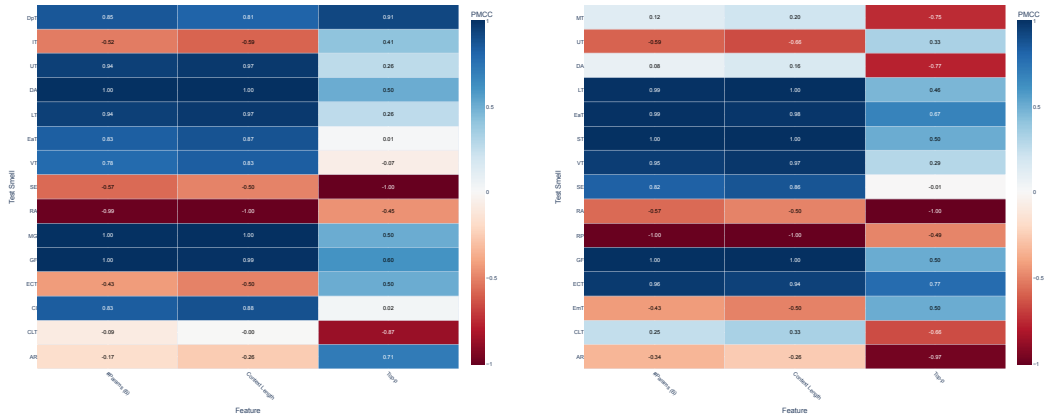
(d) PMCC result from JNose (Benchmark 1-EvoSuite).

Fig. 5. PMCC results between Project and LLM Characteristics and Test Smell Presence in Benchmark 1.

To account for dataset imbalance (i.e., multiple LLM models vs. fewer human datasets), we compute metrics by averaging smell distributions within each group (LLM vs. human) before comparison. Metrics are omitted when variance is insufficient for statistical validity. This design enables a principled, multi-dimensional comparison of smell profiles between LLM and human test suites.

[Experiment Results]:

Overlap and Divergence in Smell Profiles. Comparing human- and LLM-generated tests reveals both convergences and divergences across smell types and detectors (Tables 18a, 18b, 7). With TsDetect (Table 18a), human suites are dominated by Assertion Roulette (AR: 98.83%) and Dependent Test (DpT: 81.22%), whereas LLM tests reproduce AR but introduce verbosity-oriented smells (GF, LT, DA). EvoSuite (Table 7), in contrast, suppresses diversity: several smells are absent entirely (CLT, GF, DpT), reflecting template-driven rigidity. Cross-tool discrepancies are substantial. TsDetect consistently reports higher prevalence for structural and behavioral smells (e.g., DpT 81.22%



(a) PMCC result from TsDetect (Benchmark 2-LLM). (b) PMCC result from JNose (Benchmark 2-LLM).

Fig. 6. PMCC results between LLM Characteristics and Test Smell Presence in Benchmark 2.

in humans, 93% in LLMs), while JNose emphasizes stylistic issues such as Magic Number Test (MT: 29.57% in humans(Table 18b), up to 66.33% in GPT-4 tests (Table 7)). Some categories show convergence across tools (e.g., CI: ∼25%), while others remain tool-dependent (GF, LT, MT). Certain smells (EH, EmT, DfT) are undetected altogether, suggesting rarity or structural invisibility.

Finding 14: *Human and LLM tests share dominant structural issues (AR, DpT), but LLMs introduce verbosity-related patterns (GF, LT, DA) absent in humans. EvoSuite diverges strongly, producing rigid but shallow tests.*

Detector Complementarity. To better understand these contrasts, we analyzed detector-specific deltas (Table 19). TsDetect consistently reports higher prevalence for coupling-based and behavioral smells, while JNose emphasizes style- and token-level patterns. For instance, TsDetect exceeds JNose by +81.22 points for DpT and +43.47 for GF, while JNose excels on MT (+26.73 points). On average, deltas across all smells reach 18.69 ± 23.99 percentage points, underscoring the divergent detection philosophies. Neither tool alone captures the full spectrum, but their combined use provides more holistic coverage.

Finding 15: *TsDetect favors structural and behavioral coupling smells (large deltas for DpT, GF), while JNose captures stylistic patterns (e.g., MT). Cross-tool deltas (mean difference ∼19 percentage points, stdev ∼24 percentage points) confirm that tool complementarity is necessary for reliable smell profiling.*

Originality vs. Leakage. Similarity metrics provide contrasting pictures depending on the detector (Figures 7–8b). Under TsDetect, LLM-generated test suites diverge sharply from human tests: Cosine similarity is low (0.34), distances are high (Euclidean 169.41, Hellinger 0.68), and correlations are near-zero or negative (Pearson −0.02, Spearman −0.12; MI = 0). This suggests that LLM tests exhibit behaviorally novel and structurally distinct smell patterns. Under JNose, however, similarities are more pronounced: Cosine similarity rises to 0.77, correlations are positive (Pearson 0.68, Spearman 0.45), and MI is non-zero (0.21). These results suggest that LLM outputs partially replicate surface-level or stylistic structures, potentially reflecting shallow imitation of training artifacts. Given that

datasets like Defects4J and SF110 are known to appear in LLM pretraining corpora, such overlap may reflect memorization or leakage.

Finding 16 *While TsDetect highlights originality in LLM smell patterns, JNose reveals partial alignment with human suites, suggesting that LLMs may replicate shallow stylistic structures from training data. This raises concerns about potential data leakage and underlines the need for multi-tool triangulation.*

Table 18. Test Smells Distribution in Human-written Test Suites Detected by TsDetect and JNose

(a) Detected by TsDetect

Dataset	AR	CLT	CI	DfT	EmT	ECT	EH	GF	MG	RP	RA	SE	VT	ST	EaT	LT	DA	UT	IT	RO	MT	DpT
Defects4J	99.75	60.52	31.43	26.49	0.00	3.77	0.00	54.15	8.21	1.09	1.01	5.62	14.42	0.00	0.34	58.09	52.47	36.04	28.67	6.96	1.26	95.14
SF110	99.55	44.89	26.78	30.72	0.00	4.85	0.00	58.34	7.96	2.72	3.36	3.43	6.40	0.00	1.75	47.80	42.30	14.29	32.99	2.91	4.14	81.31
CAT-LM	97.19	37.55	13.04	9.38	0.02	1.43	0.00	34.14	7.02	2.83	3.43	1.15	5.37	0.00	1.32	30.98	29.34	12.91	19.39	9.24	3.11	67.21
Averages	98.83	47.65	23.75	22.20	0.01	3.35	0.00	48.88	7.73	2.21	2.60	3.40	8.73	0.00	1.14	45.62	41.37	21.08	27.02	6.37	2.84	81.22

(b) Detected by JNose

Dataset	AR	CLT	CI	DfT	EmT	ECT	EH	GF	MG	RP	RA	SE	VT	ST	EaT	LT	DA	UT	IT	RO	MT	DpT
Defects4J	74.11	18.04	30.00	0.00	2.86	23.04	0.00	4.64	2.14	1.61	1.25	4.46	10.89	0.71	3.39	4.82	14.11	11.96	1.07	2.32	41.43	0.00
SF110	59.38	15.62	40.62	0.00	3.12	21.88	0.00	6.25	3.12	0.00	6.25	6.25	15.62	3.12	0.00	6.25	9.38	21.88	0.00	3.12	28.12	0.00
CAT-LM	57.85	9.77	11.79	0.07	12.58	7.19	0.00	5.33	2.26	3.61	1.10	3.96	9.36	0.76	1.35	2.74	6.89	24.42	9.20	2.52	19.17	0.00
Average	63.78	14.48	27.47	0.07	6.19	17.37	0.00	5.41	2.51	1.74	2.87	4.89	11.96	1.53	1.58	4.60	10.13	19.42	3.42	2.65	29.57	0.00

* AR: Assertion Roulette, CLT: Conditional Logic Test, CI: Constructor Initialization, DfT: Default Test, EmT: Empty Test, ECT: Exception Catching Throwing, EH: Exception Handling, GF: General Fixture, MG: Mystery Guest, RP: Redundant Print, RA: Redundant Assertion, SE: Sensitive Equality, VT: Verbose Test, ST: Sleepy Test, EaT: Eager Test, LT: Lazy Test, DA: Duplicate Assert, UT: Unknown Test, IT: Ignored Test, RO: Resource Optimism, MT: Magic Number Test, DpT: Dependent Test.

Table 19. Delta Values between TsDetect and JNose for Test Smell Distributions (Human-Written Test Suites)

Dataset	AR	CLT	CI	DfT	EmT	ECT	EH	GF	MG	RP	RA	SE	VT	ST	EaT	LT	DA	UT	IT	RO	MT	DpT
Human-Written	35.05	33.17	3.72	22.13	6.18	14.02	0.00	43.47	5.22	0.47	0.27	1.49	3.23	1.53	0.44	41.02	31.24	1.66	23.60	3.72	26.73	81.22
Mean ± StdDev	**18.69 ± 23.99**																					

Summary of **RQ4:** LLM-generated tests share dominant structural issues with human-written ones (e.g., Assertion Roulette, Dependent Test), but also introduce verbosity-driven patterns (GF, LT, DA) that are absent from human suites (Finding 14). Detector deltas confirm that TsDetect emphasizes structural and behavioral coupling, while JNose focuses on stylistic patterns such as Magic Number Test, reinforcing the need for cross-tool triangulation (Finding 15). Finally, while TsDetect highlights originality in LLM smell profiles, JNose reveals partial stylistic alignment with human tests, raising concerns about shallow imitation or potential training-data leakage (Finding 16).

5 DISCUSSION

5.1 Implications

5.1.1 Implications for Researchers. Our study offers new empirical foundations for understanding how test smells emerge and evolve in LLM-generated unit tests. The multi-perspective comparison between LLM- and human-written tests—combined with tool triangulation (TsDetect vs. JNose) and metric triangulation (correlation, mutual information, statistical testing)—highlights both structural divergences and superficial similarities in test smell profiles.

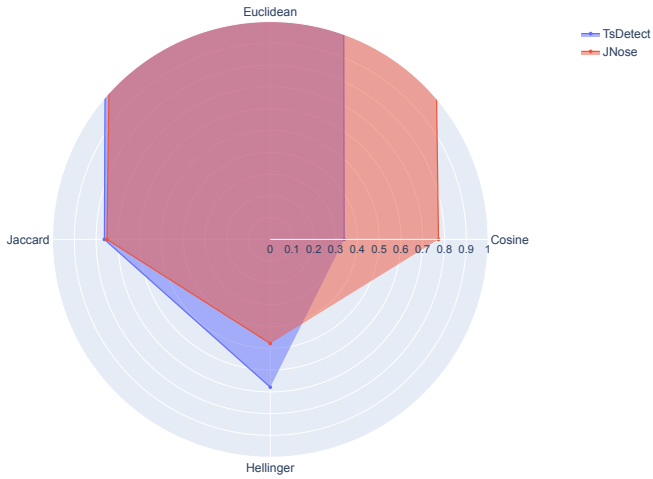
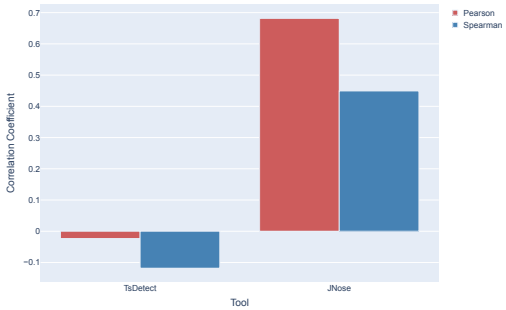
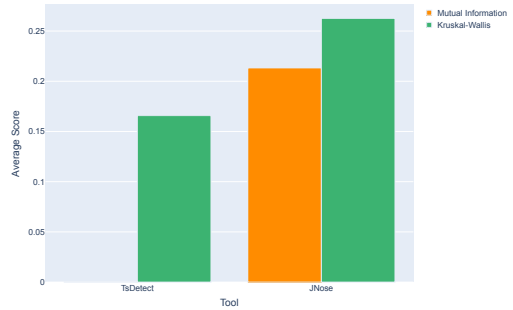


Fig. 7. Similarity Metrics between Human and LLM Distributions.



(a) PMCC and Pearson Metrics between Human and LLM Test Smell Distributions.



(b) Mutual Information and Kruskal-Wallis Metrics between Human and LLM Test Smell Distributions.

Fig. 8. Comparison of Different Correlation Metrics between Human and LLM Test Smell Distributions.

Importantly, our results challenge simplistic assumptions that LLMs merely replicate human behavior. As shown in our findings (e.g., Finding 14–16), TsDetect highlights originality in LLM smell patterns that diverge from human-written suites, whereas JNose reveals partial overlaps that may reflect shallow stylistic mimicry or even data leakage. These diverging signals underscore the need for caution when interpreting similarity as evidence of understanding: what appears as alignment may, in reality, stem from pattern-level memorization.

This has direct implications for research on LLM reliability and evaluation. Evidence of surface-level similarity (e.g., high Pearson correlation under JNose, but null mutual information under TsDetect) may indicate training-set leakage or memorization of public repositories like Defects4J and SF110. Our findings support calls from recent work (e.g., Hartmann et al. [22], Siddiq et al. [60]) for more robust evaluation protocols, including controlled benchmarks, training data audits, and memorization checks.

Furthermore, the consistent presence of smells such as Assertion Roulette and Lazy Test—even under advanced prompting strategies—suggests that prompt engineering alone is insufficient to suppress smell diffusion. Future research should explore hybrid approaches combining LLM generation with automated repair or post-processing, and systematically evaluate smell mitigation at both generation and refinement stages.

Key Takeaway for Researchers: LLMs do not simply reproduce human testing habits—they exhibit a mix of novel, stylistic, and potentially memorized smell patterns. These behaviors demand stronger interpretability, auditability, and debiasing mechanisms in empirical software engineering research.

5.1.2 Implications for Developers. Our findings provide critical guidance for developers who adopt LLMs to assist in writing or maintaining unit tests. While LLMs significantly reduce manual effort, their output is not free from quality concerns. Pervasive smells such as Assertion Roulette and Lazy Test remain dominant, even across advanced models like GPT-4 and across both class- and method-level generations. This suggests that LLM-generated tests may compromise maintainability or robustness unless carefully reviewed.

Notably, LLMs tend to underrepresent behaviors like Exception Handling, which are crucial for robustness testing. At the same time, their tendency to produce verbose or repetitive assertions hints at a lack of contextual grounding and limited behavioral insight. These limitations are exacerbated by evidence of memorization risks: similarity metrics show that some LLM-generated tests structurally resemble known human-written test suites—possibly due to overlap with training corpora—raising questions about originality and long-term generalization.

To mitigate these issues, developers should integrate smell detection tools like TsDetect and JNose directly into the CI/CD pipeline. Manual inspection remains essential for critical code paths, particularly when adopting aggressive prompting strategies or larger models. Developers should also favor hybrid workflows, combining LLM-based generation with targeted manual refinement or automated repair systems that enforce test design best practices.

Key Takeaway for Developers: In the context of automated unit test generation using prompting techniques and varying input contexts, LLM-generated tests should be treated as draft-quality artifacts that require validation. Our findings underscore the need to integrate smell detection, human review, and guided refinement into test workflows to ensure maintainable and robust test code.

5.2 Threats to Validity

External Validity. Our study targets Java-based systems, which limits generalizability to other ecosystems (e.g., Python, JavaScript). The breadth and diversity of our datasets and LLM configurations strengthen the robustness of our results within the Java ecosystem, but they do not mitigate the language-specific threat. Consequently, our findings should be interpreted as valid for Java-based testing practices, while further experiments are required to assess whether these trends hold in other programming languages and testing paradigms. To support robustness within Java, Benchmark 1 encompasses two well-established datasets—Defects4J (faulty real-world code), SF110 (diverse academic and industrial projects), and a curated dataset CMD (modern codebases)—which together cover a wide range of software styles and quality profiles. Benchmark 2 complements this with 108 real-world functions from nine projects across varied domains, including concurrency (JCTools), computer vision (JavaCV), data processing (Zxing), utility libraries (Commons-lang, Commons-math), and cloud microservices (Apollo, Jeecg Boot). Our evaluation also spans seven LLM configurations, combining proprietary (GPT-3.5, GPT-4) and open-source models (Mistral 7B, Mixtral 8x7B, CodeLlama 13B-Instruct), applied under diverse prompting paradigms (e.g., Zero-Shot,

Few-Shot, Chain-of-Thought, Full Context). The tests produced reflect both method- and class-level contexts, enabling a multi-granularity perspective on test smell emergence. While future LLMs or test generation paradigms may evolve with different characteristics, expanding this analysis to other programming languages and non-LLM-based generators remains an essential avenue for replication and broader generalization.

Internal Validity. A key threat stems from the test smell detectors themselves. While TsDetect and JNose represent complementary detection philosophies (semantic-graph and syntactic-pattern, respectively), both are imperfect. TsDetect may miss stylistic smells like Magic Number Test, while JNose tends to over-detect template-based smells, as confirmed in prior evaluations reporting over 70% false positive rates for certain smells in human tests [49]. To limit detector-induced bias, we cross-compared results between the two tools and explicitly highlighted divergences throughout the analysis. This triangulation reduces, but does not eliminate, the risk that findings reflect detector limitations rather than true semantic defects.

Another internal threat lies in the statistical assumptions made. Pearson correlation assumes linearity, which may not hold across all project and smell characteristics. To compensate, we incorporated non-parametric tests (Spearman, Kruskal–Wallis) and Mutual Information to capture non-linear or distributional relationships. Nevertheless, the absence of an independent ground truth for smell presence limits our ability to fully validate the correctness of detected patterns. A large-scale manual annotation of LLM-generated tests would further strengthen conclusions, but remains infeasible at this stage.

Construct Validity. Construct threats arise from the conceptual framing of smells and their interpretation. While our study focuses on the presence and distribution of smells, this does not automatically imply harmfulness. As shown by Panichella et al.[49], many detected smells in human tests may not degrade maintainability. Conversely, Bavota et al.[6] argue that even isolated smells can impair evolution and comprehension. Our findings fall between these positions: some smells like General Fixture or Duplicated Test may indicate critical maintenance risks in both human and LLM-generated tests, while others (e.g., Magic Number Test) may be contextually benign. This underscores the need to refine smell taxonomies and ground them in their practical impact—an open challenge in the test quality literature.

Conclusion Validity. Our statistical analyses used standard thresholds (e.g., $\alpha = 0.05$), but some p-values in Kruskal-Wallis tests were near significance boundaries (e.g., $p = 0.1658$). We thus interpret distributional similarities and differences with caution, favoring effect size and divergence metrics (e.g., Mutual Information) over binary significance claims. Additionally, correlations do not imply causation. Although strong associations were found between certain LLM types and smell profiles, especially in TsDetect, these patterns may reflect prompt artifacts, model memorization, or project-specific traits rather than inherent generalization failures.

6 RELATED WORK

Test smells, like code smells, are indicators of poor design or implementation practices that may hinder the maintainability, readability, or evolvability of software. Over the past two decades, research has extensively investigated the presence, detection, and implications of test smells in both human-written and automatically generated test suites.

Test Smells and Their Impact

The concept of test smells was first introduced by Van Deursen et al.[64], who described test design flaws such as Assertion Roulette, Eager Test, and Mystery Guest. These smells signal structural weaknesses that reduce test maintainability. Subsequent empirical studies, including the foundational work by Bavota et al.[6], demonstrated that test smells are widespread and negatively

impact program comprehension and maintenance, even when occurring in isolation. Test smells were found to increase change-proneness and reduce code understandability across a wide range of software systems.

However, recent work by Panichella et al. [49] critically revisited the assumptions and detection practices surrounding test smells. They observed that existing detection tools often produce a high rate of false positives and negatives, and that several canonical smells are not necessarily indicative of actual design flaws in developer-written or generated tests. Their findings call for re-evaluating the definitions and thresholds used by test smell detectors, and suggest the necessity for human-in-the-loop or context-aware evaluations.

Test Smells in Human-Written Tests

Several studies have focused on analyzing test smells in manually written test suites. Bavota et al.[6] examined open-source Java projects and confirmed the persistent presence of smells like Assertion Roulette and Magic Number Test. These issues are not just cosmetic—they correlate with higher maintenance effort and contribute to long-term technical debt. Peruma et al.[52] extended this line of research to mobile applications, showing that mobile-specific smells (e.g., Conditional Test Logic, Constructor Initialization) often emerge early in a project's lifecycle and tend to persist.

Test Smells in Automatically Generated Tests

Automatically generated tests have also been studied for their tendency to accumulate smells. EvoSuite, a widely used test generation tool, has been shown to produce high-coverage test suites that nonetheless suffer from smells such as Eager Test and Assertion Roulette [45, 48]. Panichella et al. [49] further demonstrated that the incidence of smells varies substantially depending on the generation tool and configuration, and that not all detected smells are semantically harmful. Their hand-annotated benchmark revealed that certain smells, while prevalent, do not correlate well with real maintenance concerns.

Test Smells in LLM-Generated Tests

Recent studies have begun examining test smells in LLM-generated code. Siddiq et al.[60] used TsDetect to analyze tests from models like GPT-3.5 and StarCoder[39], finding frequent issues such as Assertion Roulette, Magic Number Test, and Empty Test—especially in Codex outputs—despite syntactic correctness. Ouédraogo et al. [43] extended this line of work by evaluating the impact of prompting strategies (e.g., Zero-shot, Chain-of-Thought) on smell prevalence, showing that LLMs outperform EvoSuite in readability but still introduce smells, particularly under basic prompting. Their evaluation, however, had several key limitations: it employed only a single smell detector (TsDetect), focused exclusively on class-level test generation, and lacked cross-benchmark validation or cross-tool triangulation—limiting the generalizability of their findings despite the inclusion of multiple models and prompt strategies.

In contrast, our study expands the scope and rigor of test smell analysis in LLM-generated code. We adopt a dual-benchmark approach spanning class-level (Benchmark 1) and method-level (Benchmark 2) generation, enabling multi-granularity analysis. Our model set includes recent open-source LLMs—Mistral 7B, Mixtral 8x7B, and CodeLlama 13B-Instruct—offering a broader view beyond proprietary models. We evaluate diverse prompt strategies: Benchmark 1 tests structured reasoning (e.g., CoT, ToT, GToT), while Benchmark 2 varies prompt context (Self-Contained, Simple, Full). Crucially, we use two complementary detection tools (TsDetect and JNose) to cross-validate findings and mitigate tool bias. Beyond frequency counts, we apply co-occurrence analysis, correlation metrics, and distributional similarity (e.g., cosine, mutual information, Kruskal-Wallis), offering a multidimensional view of smell propagation in LLM-generated tests.

To our knowledge, this is the first study to offer a systematic, tool-diverse, and statistically grounded analysis of test smell diffusion in LLM-generated tests, with comparisons to over 770k human-written tests and structured examination of prompt, model, and software characteristics. Our findings show that LLM-generated tests exhibit a dual behavior: while TsDetect reveals smell distributions that diverge from human-written tests (suggesting generation-specific artifacts), JNose captures partial overlaps that may reflect shallow stylistic imitation. This interplay raises important implications for tool reliability, evaluation fairness, and the possibility of training-data leakage.

Summary

Overall, while prior work has established the prevalence and potential harm of test smells in both human-written and automatically generated tests, the emergence of LLMs necessitates a paradigm shift. Our findings reveal that LLM-generated tests do not merely replicate known smell patterns—they often introduce novel and structurally divergent smell profiles, particularly under minimal or shallow prompting strategies. This raises new questions about the adequacy of existing smell taxonomies and detection tools, which were originally designed for human-written code. As AI-assisted development becomes mainstream, reassessing the semantics, detection boundaries, and contextual relevance of test smells is no longer optional—it is essential for ensuring the reliability and maintainability of future software systems.

7 CONCLUSION

This paper presents the first comprehensive analysis of test smell diffusion in LLM-generated unit tests, examining over 20,000 test suites across four models and multiple prompting strategies. Through systematic comparison with SBST tools and human-written tests, we reveal crucial quality characteristics of LLM-generated test code.

Our findings show that LLM-generated tests consistently exhibit specific smell signatures—particularly Assertion Roulette, Magic Number Test, and Empty Test—with prevalence strongly influenced by model scale, context length, and prompting strategy. While larger models reduce some structural smells, they paradoxically amplify others, revealing complex trade-offs in generation parameters. Software complexity metrics further exacerbate smell emergence, suggesting LLMs struggle with complex codebases where high-quality tests are most critical.

Comparative analysis reveals distinct paradigmatic differences: EvoSuite produces predictable template-based patterns, while LLMs exhibit troubling similarities to human-written test patterns. Statistical analysis shows substantial overlaps between LLM-generated and human-written smell profiles, raising serious concerns about data leakage and inadvertent reproduction of flawed practices from training corpora.

These findings highlight the need for smell-aware generation techniques, robust detection methods for LLM-specific quality patterns, and comprehensive evaluation frameworks that balance functional correctness with maintainability. As LLMs become integral to software testing workflows, ensuring generated test code quality is essential for sustainable software development.

ACKNOWLEDGEMENTS

This research was funded in whole, or in part, by the Luxembourg National Research Fund (FNR), grant reference AFR PhD bilateral, project reference 17185670. This work was also supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 949014). For the purpose of open access, and in fulfilment of the obligations arising from the grant agreement, the author has applied a Creative Commons Attribution 4.0 International (CC BY 4.0) license to any Author Accepted Manuscript version arising from this submission.

REFERENCES

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*. 207–216.

[2] Wajdi Aljedaani, Anthony Peruma, Ahmed Aljohani, Mazen Alotaibi, Mohamed Wiem Mkaouer, Ali Ouni, Christian D Newman, Abdullatif Ghallab, and Stephanie Ludi. 2021. Test smell detection tools: A systematic mapping study. In *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering*. 170–180.

[3] Alberto Bacchelli, Paolo Ciancarini, and Davide Rossi. 2008. On the effectiveness of manual and automatic unit test generation. In *2008 The Third International Conference on Software Engineering Advances*. IEEE, 252–257.

[4] Ebrahim Bagheri and Dragan Gasevic. 2011. Assessing the maintainability of software product line feature models using structural metrics. *Software Quality Journal* 19 (2011), 579–612.

[5] Gabriele Bavota, Abdallah Qusef, Rocco Oliveto, Andrea De Lucia, and David Binkley. 2012. An empirical analysis of the distribution of unit test smells and their impact on software maintenance. In *2012 28th IEEE international conference on software maintenance (ICSM)*. IEEE, 56–65.

[6] Gabriele Bavota, Abdallah Qusef, Rocco Oliveto, Andrea De Lucia, and Dave Binkley. 2015. Are test smells really harmful? an empirical study. *Empirical Software Engineering* 20 (2015), 1052–1094.

[7] Kent Beck. 2000. *Extreme programming explained: embrace change*. addison-wesley professional.

[8] Matteo Biagiola, Gianluca Ghislotti, and Paolo Tonella. 2025. Improving the readability of automatically generated tests using large language models. In *2025 IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 162–173.

[9] Lionel C. Briand, Christian Bunse, and John W. Daly. 2001. A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs. *IEEE Transactions on Software Engineering* 27, 6 (2001), 513–530.

[10] Lionel C Briand, Jürgen Wüst, John W Daly, and D Victor Porter. 2000. Exploring the relationships between design measures and software quality in object-oriented systems. *Journal of systems and software* 51, 3 (2000), 245–273.

[11] Introducing ChatGPT. 2023. OpenAI. *URL: https://openai.com/index/chatgpt/[accessed 2024-05-21]* (2023).

[12] Shyam R Chidamber and Chris F Kemerer. 1994. A metrics suite for object oriented design. *IEEE Transactions on software engineering* 20, 6 (1994), 476–493.

[13] Jacob Cohen. 1988. *Statistical Power Analysis for the Behavioural Sciences* (second ed.). Routledge.

[14] Thomas M Cover. 1999. *Elements of information theory*. John Wiley & Sons.

[15] Ermira Daka and Gordon Fraser. 2014. A survey on unit testing practices and problems. In *2014 IEEE 25th International Symposium on Software Reliability Engineering*. IEEE, 201–211.

[16] Amirhossein Deljouyi, Roham Koohestani, Maliheh Izadi, and Andy Zaidman. 2024. Leveraging large language models for enhancing the understandability of generated unit tests. *arXiv preprint arXiv:2408.11710* (2024).

[17] Uwe Flick. 2018. Designing qualitative research. (2018).

[18] Gordon Fraser and Andrea Arcuri. 2011. Evosuite: automatic test suite generation for object-oriented software. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. 416–419.

[19] Gordon Fraser and Andrea Arcuri. 2014. A large-scale evaluation of automated unit test generation using evosuite. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 24, 2 (2014), 1–42.

[20] Vahid Garousi and Barış Küçük. 2018. Smells in software test code: A survey of knowledge in industry and academia. *Journal of systems and software* 138 (2018), 52–81.

[21] Tibor Gyimóthy, Rudolf Ferenc, and Istvan Siket. 2005. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software engineering* 31, 10 (2005), 897–910.

[22] Valentin Hartmann, Anshuman Suri, Vincent Bindschaedler, David Evans, Shruti Tople, and Robert West. 2023. Sok: Memorization in general-purpose large language models. *arXiv preprint arXiv:2310.18362* (2023).

[23] Yifeng He, Jiabo Huang, Yuyang Rong, Yiwen Guo, Ethan Wang, and Hao Chen. 2024. UniTSyn: A Large-Scale Dataset Capable of Enhancing the Prowess of Large Language Models for Program Testing. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 1061–1072.

[24] Ernst Hellinger. 1909. Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen. *Journal für die reine und angewandte Mathematik* 1909, 136 (1909), 210–271.

[25] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2019. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751* (2019).

[26] Anna Huang et al. 2008. Similarity measures for text document clustering. In *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, Vol. 4. 9–56.

[27] Paul Jaccard. 1901. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bull Soc Vaudoise Sci Nat* 37 (1901), 547–579.

[28] Kush Jain and Claire Le Goues. 2025. TestForge: Feedback-Driven, Agentic Test Suite Generation. *arXiv preprint arXiv:2503.14713* (2025).

[29] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825* (2023).
[30] Nildo Silva Junior, Larissa Rocha, Luana Almeida Martins, and Ivan Machado. 2020. A survey on test practitioners' awareness of test smells. *arXiv preprint arXiv:2003.05613* (2020).
[31] Marian Jureczko. 2011. Significance of different software metrics in defect prediction. *Software Engineering: An International Journal* 1, 1 (2011), 86–95.
[32] René Just, Darioush Jalali, and Michael D Ernst. 2014. Defects4J: A database of existing faults to enable controlled testing studies for Java programs. In *Proceedings of the 2014 international symposium on software testing and analysis*. 437–440.
[33] Thomas Kailath. 1967. The divergence and Bhattacharyya distance measures in signal selection. *IEEE transactions on communication technology* 15, 1 (1967), 52–60.
[34] Maurice George Kendall. 1943. The advanced theory of statistics. (1943).
[35] Barbara A Kitchenham, Shari Lawrence Pfleeger, Lesley M Pickard, Peter W Jones, David C. Hoaglin, Khaled El Emam, and Jarrett Rosenberg. 2002. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on software engineering* 28, 8 (2002), 721–734.
[36] Pavneet Singh Kochhar, Tegawendé F Bissyandé, David Lo, and Lingxiao Jiang. 2013. Adoption of software testing in open source projects–A preliminary study on 50,000 projects. In *2013 17th european conference on software maintenance and reengineering*. IEEE, 353–356.
[37] Pavneet Singh Kochhar, Tegawendé F Bissyandé, David Lo, and Lingxiao Jiang. 2013. An empirical study of adoption of software testing in open source projects. In *2013 13th International Conference on Quality Software*. IEEE, 103–112.
[38] William H Kruskal and W Allen Wallis. 1952. Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association* 47, 260 (1952), 583–621.
[39] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. 2023. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161* (2023).
[40] Christopher D Manning. 2008. Introduction to information retrieval.
[41] Thomas J McCabe. 1976. A complexity measure. *IEEE Transactions on software Engineering* 4 (1976), 308–320.
[42] What Is Data Mining. 2006. *Introduction to data mining*. Springer.
[43] Wendkûuni C Ouédraogo, Kader Kaboré, Haoye Tian, Yewei Song, Anil Koyuncu, Jacques Klein, David Lo, and Tegawendé F Bissyandé. 2024. Large-scale, Independent and Comprehensive study of the power of LLMs for test case generation. *arXiv preprint arXiv:2407.00225* (2024).
[44] Wendkuuni C Ouedraogo, Kader Kabore, Haoye Tian, Yewei Song, Anil Koyuncu, Jacques Klein, David Lo, and Tegawende F Bissyande. 2024. Llms and prompting for unit test generation: A large-scale evaluation. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 2464–2465.
[45] Fabio Palomba, Dario Di Nucci, Annibale Panichella, Rocco Oliveto, and Andrea De Lucia. 2016. On the diffusion of test smells in automatically generated test code: An empirical study. In *Proceedings of the 9th international workshop on search-based software testing*. 5–14.
[46] Rangeet Pan, Myeongsoo Kim, Rahul Krishna, Raju Pavuluri, and Saurabh Sinha. 2025. Aster: Natural and multi-language unit test generation with llms. *arXiv preprint arXiv:2409.03093* (2025).
[47] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. 2017. Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets. *IEEE Transactions on Software Engineering* 44, 2 (2017), 122–158.
[48] Annibale Panichella, Sebastiano Panichella, Gordon Fraser, Anand Ashok Sawant, and Vincent J Hellendoorn. 2020. Revisiting test smells in automatically generated tests: limitations, pitfalls, and opportunities. In *2020 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 523–533.
[49] Annibale Panichella, Sebastiano Panichella, Gordon Fraser, Anand Ashok Sawant, and Vincent J Hellendoorn. 2022. Test smells 20 years later: detectability, validity, and reliability. *Empirical Software Engineering* 27, 7 (2022), 170.
[50] Athanasios Papoulis. 1965. *Random variables and stochastic processes*. McGraw Hill.
[51] Anthony Peruma, Khalid Almalki, Christian D Newman, Mohamed Wiem Mkaouer, Ali Ouni, and Fabio Palomba. 2020. Tsdetect: An open source test smells detection tool. In *Proceedings of the 28th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*. 1650–1654.
[52] Anthony Peruma, Khalid Saeed Almalki, Christian D Newman, Mohamed Wiem Mkaouer, Ali Ouni, and Fabio Palomba. 2019. On the distribution of test smells in open source android applications: An exploratory study.(2019). *Citado na* (2019), 13.
[53] Nikitha Rao, Kush Jain, Uri Alon, Claire Le Goues, and Vincent J Hellendoorn. 2023. CAT-LM training language models on aligned code and tests. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*.

IEEE, 409–420.
[54] Dominik Arne Rebro, Stanislav Chren, and Bruno Rossi. 2023. Source code metrics for software defects prediction. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*. 1469–1472.
[55] Sheldon M Ross, Sheldon M Ross, Sheldon M Ross, and Sheldon M Ross. 1976. *A first course in probability*. Vol. 2. Macmillan New York.
[56] Per Runeson and Martin Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering* 14 (2009), 131–164.
[57] June Sallou, Thomas Durieux, and Annibale Panichella. 2023. Breaking the silence: the threats of using llms in software engineering. *arXiv preprint arXiv:2312.08055* (2023).
[58] Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk. 2008. Nearest-neighbor methods in learning and vision. *IEEE Trans. Neural Networks* 19, 2 (2008), 377.
[59] James Shore and Shane Warden. 2021. *The art of agile development*. " O'Reilly Media, Inc.".
[60] Mohammed Latif Siddiq, Joanna Cecilia Da Silva Santos, Ridwanul Hasan Tanvir, Noshin Ulfat, Fahmid Al Rifat, and Vinícius Carvalho Lopes. 2024. Using large language models to generate junit tests: An empirical study. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*. 313–322.
[61] Saleem Siddiqui. 2021. *Learning Test-Driven Development*. " O'Reilly Media, Inc.".
[62] Davide Spadini, Fabio Palomba, Andy Zaidman, Magiel Bruntink, and Alberto Bacchelli. 2018. On the relation of test smells to software code quality. In *2018 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 1–12.
[63] Charles Spearman. 1961. The proof and measurement of association between two things. (1961).
[64] Arie Van Deursen, Leon Moonen, Alex Van Den Bergh, and Gerard Kok. 2001. Refactoring test code. In *Proceedings of the 2nd international conference on extreme programming and flexible processes in software engineering (XP2001)*. Citeseer, 92–95.
[65] Tássio Virgínio, Luana Martins, Larissa Rocha, Railana Santana, Adriana Cruz, Heitor Costa, and Ivan Machado. 2020. Jnose: Java test smell detector. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering*. 564–569.
[66] Hai Wang, Zeshui Xu, Hamido Fujita, and Shousheng Liu. 2016. Towards felicitous decision making: An overview on challenges and trends of Big Data. *Information Sciences* 367 (2016), 747–765.
[67] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.
[68] Ruijie Xu, Zengzhi Wang, Run-Ze Fan, and Pengfei Liu. 2024. Benchmarking benchmark leakage in large language models. *arXiv preprint arXiv:2404.18824* (2024).
[69] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems* 36 (2024).
[70] Robert K Yin. 2017. *Case study research and applications: Design and methods*. Sage publications.
[71] Quanjun Zhang, Ye Shang, Chunrong Fang, Siqi Gu, Jianyi Zhou, and Zhenyu Chen. 2024. TestBench: Evaluating Class-Level Test Case Generation Capability of Large Language Models. *arXiv preprint arXiv:2409.17561* (2024).