A Systematic Literature Review on Automated Unit Test Generation Techniques, Algorithms, and Tools

Muhammad Hammad Ali, Muhammad Abbas, Syed Mukhtar-ul-Hassan, Mudassar Adeel Ahmed
Department of Software Engineering, Capital University of Science and Technology, Islamabad, Pakistan
*Correspondence: hammadmuhammad2288@gmail.com

Citation | Ali. M. H, Abbas. M, Hassan. S. M. U, Ahmed. M. A, "A Systematic Literature Review on Automated Unit Test Generation Techniques, Algorithms, and Tools", IJIST, Special Issue pp 694-739, May 2026
Received| April 09, 2026 **Revised|** May 15, 2026 **Accepted|** May 17, 2026 **Published|** May 19, 2026.

The generation of automated unit tests has advanced rapidly with Natural Language Processing (NLP) and Large Language Model (LLM)-based methods. Nonetheless, the field is still disjointed in terms of techniques, algorithms, tools, and verification practices, making it difficult to determine where significant progress has been made, and what major limitations persist. This paper is a systematic literature review of 48 primary studies on automated unit test generation across four dimensions of analysis: generation techniques, underlying algorithms, tools/frameworks, and verification or post-generation improvement mechanisms. The results indicate that tools and frameworks account for the highest number of studies, with 27 (56.3%), followed by test generation techniques and algorithms with 14 (29.2%), verification and evaluation approaches with 10 (20.8%), and assertion, repair, and test suite improvement mechanisms with 8 (16.7%) studies. Evidence has been reported that the latest LLM-based and hybrid methods have shown better contextual generation and workflow integration, though these methods heavily rely on validation, repair, and quality of oracles. As an example, using mutation-guided prompting has been reported to reach a mutation score of 93.57% on synthetic buggy code and detect up to 28% more faulty human-written snippets, and industrial LLM-based test improvement has reported 75% build success, 57% reliable pass rate, 25% coverage increase, and 73% acceptance by engineers. The review in general indicates that the trend of automated unit test generation is no longer about isolated generation of tests, but rather about the generation of tests that are integrated into quality-engineering pipelines. New developments will not rely as much on the generate raw tests but on integrating generation with verification, assertion enhancement, repair, reproducible evaluation, and deploying these capabilities in industrial settings.

Keywords: Automated Unit Test Generation; Software Testing; Systematic Literature Review; Natural Language Processing; Large Language Models; Test Verification; Test Oracle Generation; Mutation Testing; Assertion Generation

Introduction:

The automated generation of unit tests is no longer viewed as a purely technical automation task, but a software-engineering challenge influenced by scalability, semantic correctness and operability. Unit testing is imperative in modern development settings to safeguard the quality of the software, but manual building of tests is costly, monotonous, and challenging to maintain with the high-rate of releases. This prompted new attention to automated generation techniques, especially since Large Language Models (LLMs) have started to augment or even surpass previous search-based and rule-based systems in the English tasks that need knowledge about the context of the code and developers. Search-based software test-data generation provides an important foundation, demonstrating how optimization and search strategies were previously used to generate test data, and highlighting recent developments in which LLM-based systems have significantly expanded this direction [1]. Nevertheless, it should not be confused with a solution that has just been achieved. Previous research on automated tests and search-based test generation demonstrates that the years-old automated testing was driven by the effectiveness, maintainability, and tool support and test-data generation limitations, therefore, recent LLM-based methods should be regarded as a continuation of a more ancient automation trend instead of a novel area of expertise [2][1]. This enhances the factual basis of the review. The evidence base is methodologically disproportionate, and more power of generation did not necessarily lead to more power of test or industry reliability [3][4].

This change is significant since the use of conventional automated unit test generation techniques and those based on LLMs are not the same in terms of addressing the same issue. Search-based, symbolic and feedback directed generation are still useful classical methods since they exhibit structural expressiveness, reproducibility and can be useful in exploring the program paths. Past optimisation methods on automated unit Test generation reveal that the older generation methods of using algorithmic search and objective based generation were significant backgrounds to the recent change of using LLM based methods [5]. Another instance of the pre-LLM algorithms being used to enhance the effectiveness of test coverage and test generation is the optimisation-based object-oriented test case generation, where the generation strategies are structured search approaches [6]. Related traditional approaches also include use-case-driven classification, optimisation-based object-oriented test generation, and other algorithmic strategies that demonstrate how pre-LLM research attempted to improve test coverage and generation efficiency through more structured test-selection logic [7][5][6]. These studies provide useful contrast with newer LLM-based pipelines. Yet they have historically struggled with semantic adequacy, readable test construction, and robust oracle generation. By contrast, LLM-based methods can infer richer contextual patterns from source code, comments, and prompts, which makes them attractive for generating tests that appear more natural and developer-aligned. CodeT5+ is relevant to this discussion because open code language models provide the technical foundation for code understanding and generation tasks that underpin many recent LLM-based testing approaches [8]. Comparative evidence between Chat GPT and search-based software testing suggests that LLM-based generation can be promising, but should still be assessed against established automated test-generation baselines [9][10][11]. Nevertheless, recent findings indicate that such semantic rewards are weak unless they are through validation and repair systems. That is, it does not necessarily involve a shift in the literature between the so-called traditional and so-called LLM-based methods, but between the single-stage pipelines of generation and methods of multi-stage pipelines that integrate generation with screening, repair, and verification [12][3].

This more cautious interpretation is supported by recent studies of industry and industry and empirical studies. Elsewhere, as at Meta, TestGen-LLM was not implemented as an unconstrained generator, but as a test-improvement system with controls and filtering, and

with measurable acceptance criteria. One reported assessment showed that 75% of the generated test cases executed successfully, 57 percent from the test cases were reliably passed, and 25 percent expanded their coverage, and 73 percent of recommendations were acceptable to be produced by engineers [13]. These numbers are promising, but also demonstrate a critical drawback: industrial usefulness is not a property of generation itself, but of the quality of the generated tests, passed through the filter of compilation, execution, review and maintenance. This is exactly why a critical analysis of automated unit test generation needs to go beyond mere numbers of test generated or just an assertion of a coverage increase. The actual question is whether or not such tests are now executable, fault-revealing, and can be applied in the real-world contexts of software-development [13][4].

Secondly, there is no solution on the evaluation of generated tests. The coverage metrics continue to feature prominently in much of the literature, but as a proxy of test effectiveness, coverage is progressively becoming a topic of debate by which recent research contends it is an incomplete proxy. [14] Specifically refute this premise and demonstrate that mutation-guided prompting has the potential to enhance the bug-revealing efficacy of the massively generated tests by LLM. They showed that their MuTAP method found up to 28% more faulty human-written code snippets as well as a mutation score of 93.57% on synthetic buggy code showing that structural reach and behavioral fault detection should not be treated as synonymous metrics. This distinction is especially important for the present study because a systematic literature review centered on techniques, algorithms, and tools should not flatten these outcome dimensions into one another. A method that reaches more code is not necessarily a method that produces stronger tests, and a tool that generates executable tests is not necessarily one that produces dependable oracles [14].

The oracle problem remains one of the clearest bottlenecks in the field. Automated unit test generation is not only about producing inputs; it is equally about deciding what counts as correct behavior. Recent assertion-generation studies demonstrate progress, but they also show how far the field still has to go. Earlier information-retrieval-based assertion generation also demonstrates that assertion construction has long been treated as a separate technical problem rather than a simple by-product of test generation [15]. ChIRAAG shows that LLM-informed assertion generation can support rapid test enhancement, but also confirms that assertion quality remains a distinct challenge within automated unit test generation [16]. Assertions remain a central concern in software testing because they determine whether generated tests can meaningfully check expected behavior rather than simply execute code paths [17]. AsserT5, for example, reported exact matches with ground-truth assertions in up to 59.5% of cases, yet on real bugs it suggested fault-finding assertions for only 33 of 138 assertion-detectable bugs [18]. More recent work on LLM-driven oracle generation argues even more directly that many existing approaches still generate regression-style checks over implemented behavior rather than genuinely discriminating between correct and incorrect intended behavior [19]. This means that the apparent fluency of LLM-generated tests can be misleading: readable or even executable tests may still remain weak if their assertions do not capture meaningful behavioral expectations. To that end, verification and oracle quality must be on the agenda as the key dimensions of analysis, instead of being on the margins as refinements.

These developments reveal a greater gap in research. Recent surveys have started to chart the application of LLMs to software testing and test-case generation, but tend to focus on the workflows, datasets, or overall testing applications of the technology instead of the entire range of automated unit test generation techniques, algorithms, tools, and verification systems [3][4]. Surveys about software testing using large language models confirm that the testing based on large language models is rapidly increasing, although it is also disproportionate in the quality of the assessment, repeatability, and practical maturity [3]. This fragmentation

has become a severe drawback as even the most modern systems are becoming a combination of multiple elements simultaneously: the timely engineering system, context selection, mutation-driven optimization, repair loops, assertion support, and execution-based screening. Consequently, the actual unit of progress in the discipline is turning into the pipeline and not the stand-alone generator. The review that emphasizes on the tools, on the LLMs, and/or on the evaluation measures may fail to capture the practical interactions between these aspects [14][20].

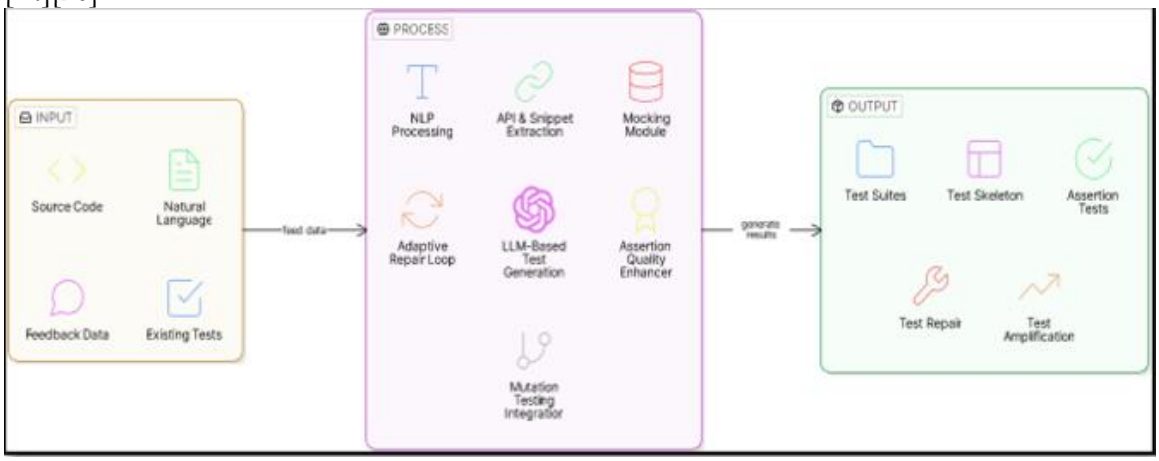

Figure 1. Conceptual pipeline of automated unit test generation, verification and refinement

It is against this backdrop that this study discusses automated unit test generation as a fast moving and multi-phase research field whereby the key challenge is not necessarily to generate more tests automatically, but rather to generate semantically meaningful, executable, fault-revealing and deployable tests. The review thus strategically integrates techniques, algorithms and tools within a single frame of analysis. Techniques define the conceptual approaches to the derivation of tests based on code or natural-language artefacts; algorithms define how the approaches operationalize the conversion of system inputs to test outputs; and tools describe how ideas in practice are implemented, assessed, and made more relevant to work processes. The combination of the synthesized dimensions will answer the duo of issues related to the necessity of a more comprehensive and up-to-date understanding of the field and establish a more effective base of the research purposes and questions to follow.

Research Objectives:

This is a structured literature review that seeks to synthesize in an evidence-based and systematic manner research on the area of automated unit test generation. The objectives of the study are the following:

To recognize and classify the major methods of automated unit test generation, both traditional and the NLP/LLM-based ones.

To analyze how current strategies can be used to test and ensure the quality of tests, especially the mutation testing method, strength of assertion and test refinements after generation.

To examine tools and structures mentioned in the literature along with inputs they claim to support, outputs and validation plans.

To synthesize the set of fundamental algorithms to convert the source artefacts into executable tests and assess their effects on coverage, executability and fault detection.

These goals aim to provide a certain alignment with the research questions and to facilitate an organized examination of methodologies, algorithms, tools and verification mechanisms of the chosen studies.

Novelty and Contributions:

The novelty of the study has to be its combined and evidence-based view on the automated unit test generation. Although recent reviews have considered either the testing of LLM-based or software testing (in general) they tend to do so in isolation, techniques, tools, algorithms and verification mechanisms. Conversely, this paper offers a synthesis that is more coherent and is representative of the multi-level and mutually supporting character of current automated testing processes.

The key contributions of this review are as follows:

Integrated analytical framework: The analysis of automated unit test generation is conducted in four dimensions, all of which are interrelated: techniques, algorithms, tools/frameworks, and verification mechanisms, a more comprehensive analysis than the previous partial surveys.

Pipeline-oriented perspective: The review re-conceptualizes automated unit test generation as a multi-step process, that is, in the form of generation, validation, refinement and repair, but not a one-step code-generation problem.

Evidence-based synthesis of 48 studies: In contrast to narrative reviews, this article maintains a systematic mapping of 48 primary studies, thus maintaining a track of the findings to the evidence.

Critical evaluation of LLM-based approaches: The article points at the mismatch between the generation capability and the quality of tests, demonstrating that the ability to make improvements on coverage and executability does not in any way positively influence the quality of fault detection and oracle.

Industry-aware insights: The review uses new industrial, as well as, empirical evidence to determine the feasibility of automated workflow and unit test generation tools in practice.

In general, this work adds a more detailed and analytical insight into the understanding of automated unit test generation, integrating both the latest developments with an analytical framework based on the analysis and its verification.

Critical Literature Review and Gap Synthesis:

The area of automated unit test generation is still relatively fragmented, despite recent decades compounding it with a sudden spurt of accelerated research, which is the reason a more systematic review should be considered. The most visible trend is the increase of the approaches that are based on LLM and hybrid, however, the increase has not yet yielded a stable or cumulative base of evidence. Recent research frequently has been able to announce promising improvements in coverage, executability or readability, although they strongly vary in terms of benchmark selection, prompt design, selection of context, repair policy and where validation is acceptable. Consequently, technical novelty prevails in the literature but comparability and synthesis is poor. This is what makes a review structured around techniques, algorithms, tools and verification mechanisms an absolute necessity and not an option.

One of the first significant gaps relates to the further over-emphasis on the success of generations as compared to the quality of validation. The more recent talk about mutation-testing progress motivates the necessity to adopt fault-detection-based assessment instead of just covering or merely generating successful tests [21]. This gap is relevant to mutation testing in real-world studies since it offers more compelling evidence of the effectiveness of tests, rather than coverage alone, and it aids in revealing whether the generated tests are in fact capable of identifying faults [22]. The recent literature continues to put a significant emphasis on the successful test production or growth in structural coverage as the primary indicator of improvement. Nonetheless, more robust recent research is starting to indicate coverage as an imperfect and inaccurate proxy measure of effective practice tests. MuTAP is particularly significant in this aspect as it explicitly indicates that the mutation-directed prompting can expose some mistakes, which are overlooked by the baseline LLM prompting and other more

traditional automated-generation methods. The fact that it is claimed to find up to 28% more faulty human-written code snippets, combined with a mutation score of 93.57% on synthetic buggy code, demonstrates that test adequacy must go beyond measuring the amount of code executed, and therefore faulty behavior must also be brought to light. This leaves a gap in research: an avenue has been better working on producing tests rather than demonstrating that tests are behaviorally strong.

A second defect is in the chronic oracle problem. The two interrelated issues with creating automated unit tests have always been to generate test inputs and determine what correct behavior to generate. The unresolved second challenge is evident through the latest studies on assertion and oracle which assert that it is essential to undertake the studies in order to accomplish what has not been achieved so far. A neural assertion-generation model, such as AsserT5, can perform better on ground-truth assertion prediction, being able to correctly identify the assertion in up to 59.5% of cases, but its actual faults-finding capability remains weak, with only 33 out of 138 bugs identifiable by assertion detectors being correctly identified by their supposed assertions. Later work on generation of oracles using LLM-driven oracles even further argues that several more recent methods continue to generate regression-like checks based on implemented behavior, instead of attempted robust representations of desired correctness. These studies taken collectively indicate that even a visually persuasive or performance demonstration test could be weak when assisted by weak or shallow statements. This presents a vital loophole since it implies that the weak assurance of behaviour can still be hidden by the generation fluency. The concern is also supported by the research on semantic-conflict detection since unit tests are the most useful and effective when they indicate the behavioral inconsistencies but not the fact that the code can run [23]. This concern is bolstered by LLMorpheus by demonstrating that mutation-focused LLM approaches could be employed to test and test generated tests to a level of syntactic correctness beyond what is on the surface [24].

A third gap is related to the absence of standardized practices of evaluation and reporting. Comparisons of the performance of LLMs in unit test generation indicate that in the studies involving various benchmarks, prompts, metrics, and reporting practice, it is challenging to consistently compare the model performance [25]. The articles that were published not earlier than five years ago are diverse in terms of their subject languages, benchmark data, model types, template of prompts, context windows, post-processing, and performance indicators. Even those studies that discuss related technical issues, are often hard to compare directly as they optimize different results in different experimental conditions. The contributions made by ChatUniTest, TestART, and MuTAP (and similar systems) are significant, but this is under substantially different configurations, further undermining cumulative interpretation. This issue is even more critical in the LLM era, as API changes and prompt sensitivity changes can be hidden and result in changes in results in the short term. The literature thus does not have the stability of benchmarking that will facilitate cross-study comparisons.

A fourth gap is the lack of extensiveness in assimilating methods, techniques, algorithms, and validation inside a single framework of analysis. A significant portion of the current literature continues to consider one dimension of it at a time: some papers are about prompting, some are about mutation-based evaluation, others about assertion generation and others about tool demonstrations. The resulting specialization has yielded helpful depth, but has also cut up knowledge of the mechanics of automation of unit tests systems as they currently exist. The most recent and powerful methods are gradually uniting various elements simultaneously, such as adaptive context selection, repair iteration, mutation-guided feedback, assertion enhancement, and execution filtering. This implies that the actual unit of advancement becomes more the pipeline, not the solitary generator anymore. This pipeline

perspective is supported in the test-based refinement research by the fact that AI-assisted programming can be more reliable with generation in the presence of executable tests and iterative checking [26]. Test generation using property of cyber-physical systems has demonstrated that testing using LLM is also being investigated in safety sensitive areas where guardrailing (and behavioral restrictions) are crucial [27]. A literature review, which separates these aspects, is potentially missing the real trend in which the field is moving.

Fifth gap is about industrial relevance. The body of evidence of deployment is widening, although by far it is skimpier than benchmark-based assessment. The Meta study is especially important as it demonstrates that it is possible to use it industrially, but also it demonstrates how limited is that possibility. The results of the reported experiments that showed 75% build success, 57% reliable pass rate, 25% coverage increase in a single evaluation, and 73% acceptability of the recommendations by the engineers show that it is practical value that is dependent on stringent filtering, measurability of improvement, and developer trust and not just on generation. This is important as a lot of the academic literature continues to test tools when used in controlled environments without demonstrating the behavior of generated tests in actual repositories, moving systems or long-term maintenance processes. There is thus a stronger technical demonstration as opposed to operational evidence in the literature.

The last, and tactical gap is the instability over time. The generation of unit tests with LLM is changing rapidly, with models, workflows and evaluation patterns emerging more rapidly than the literature can converge on common standards. Recent survey research indicates that not only the volume of research on LLM-based testing is increasing rapidly but that there remains uncertainty regarding the reproducibility, validation and integration into industry. This implies that if one has narrow or stagnant reviews then they become obsolete easily and broad reviews run a danger of being descriptive unless they make an explicit synthesizing point where the field is converging and where it is still weak in methodology. That is why the current review does not merely take the chosen 48 studies as a bunch of new articles, but as a sign of a field in transition: that which is both more competent, but nevertheless unequal in terms of quality of evaluation and the strength of oracles and practical reliability.

In this regard, the major void will cease to be the-lack of automated unit test generation techniques. Instead, it is the lack of a well-integrated comprehension of the interaction between techniques, algorithms, tools and verification mechanisms between the latest literature to generate tests that are not only generated, but are also executable, fault revealing, maintainable and practical. The gap in analysis that is supposed to be fulfilled by this review is that.

The research questions formed are:

RQ1: What Natural Language Processing (NLP) techniques have been applied for automated unit test generation?

RQ2: What methods do existing frameworks use to verify or optimize the quality of generated tests?

RQ3: What tools have been developed and/or used for automated unit test generation, and what inputs and outputs do these tools support?

RQ4: What algorithms have been developed and/or applied in automated unit test generation, and how do they transform system inputs into test outputs?

Research Methodology:

This research has undertaken a Systematic Literature Review (SLR) to locate, review, and integrate recent evidence regarding automated unit test generation techniques, algorithms and tools. The most suitable methodological approach adopted was an SLR since the area has evolved quickly over the last few years and currently includes a diverse combination of conventional search-based techniques, LLM-based generation methods, verification systems,

assertion-generation systems and hybrid repair-based pipelines. The traditional narrative review would not be suitable in this setting as it would have a high risk of selective reading, low reproducibility, and reduced transparency. The SLR approach, in contrast, offers a more systematic ground on the basis of evidence gathering and examination in a manner, which is more rigorous, traceable and consistent with the research questions of the study [28][29][30].

In this study, the use of an SLR was particularly crucial since the title and objectives are multi-dimensional. The review is not based on a single aspect of automated unit test generation (e.g., only LLM prompting or tool development), but rather focuses on techniques, algorithms, tools, and verification mechanisms. This necessitated an approach that could facilitate extensive evidence harvesting and at the same time enable the application of structured screening, quality control and study-level mapping on the 48 primary studies that had been selected. It was also through the methodological design that the relationship between the literature collected and the four research questions that were to be answered in the review were to be preserved.

To strengthen transparency and methodological consistency, the review process was informed by PRISMA 2020 for systematic review reporting, PRISMA-S for search transparency, and SEGRESS for reporting secondary studies in software engineering [30][29][28]. These frameworks were used to improve the clarity of the review workflow, particularly in relation to search design, staged screening, quality assessment, data extraction, and final synthesis. Their use is important because one of the weaknesses of the earlier manuscript version was that the overall methodology was described only in broad terms, without making the full review process sufficiently explicit. The present version therefore clarifies the logic of the review without altering the original SLR structure on which the paper is based.

The methodology was organized into a series of linked stages: development of the review protocol, specification of inclusion and exclusion criteria, systematic database searching, multi-stage study selection, quality assessment of eligible studies, structured data extraction, and narrative as well as descriptive synthesis of the final evidence base. Each of these stages was necessary to ensure that the selected studies were relevant to the topic, sufficiently rigorous in design, and informative enough to answer the research questions defined in Section 1. The complete methodological workflow is presented in Figure 2, which summarizes how the review moved from initial identification of records to final inclusion of the 48 studies used in the synthesis.

Overall, this methodological design was intended to balance breadth, rigor, and traceability. Breadth was needed because automated unit test generation is now spread across multiple technical and evaluative traditions. The rigor was required since the literature in the recent past tends to announce good results in very mixed experimental conditions and thus critical selection and appraisal is necessary. Traceability was needed because the value of this review depends not only on summarizing the field, but on showing clearly how the selected 48 studies were identified, assessed, and mapped to the study's analytical framework. For these reasons, the SLR methodology provides the most defensible foundation for the present review.

To improve methodological transparency and show how the review moved from broad retrieval to the final evidence base, Figure 2 presents the overall systematic literature review workflow used in this study.

As shown in Figure 2, the review proceeded through sequential stages of database searching, duplicate removal, title and abstract screening, full-text eligibility assessment, quality appraisal, and final data extraction and synthesis of the 48 included studies.

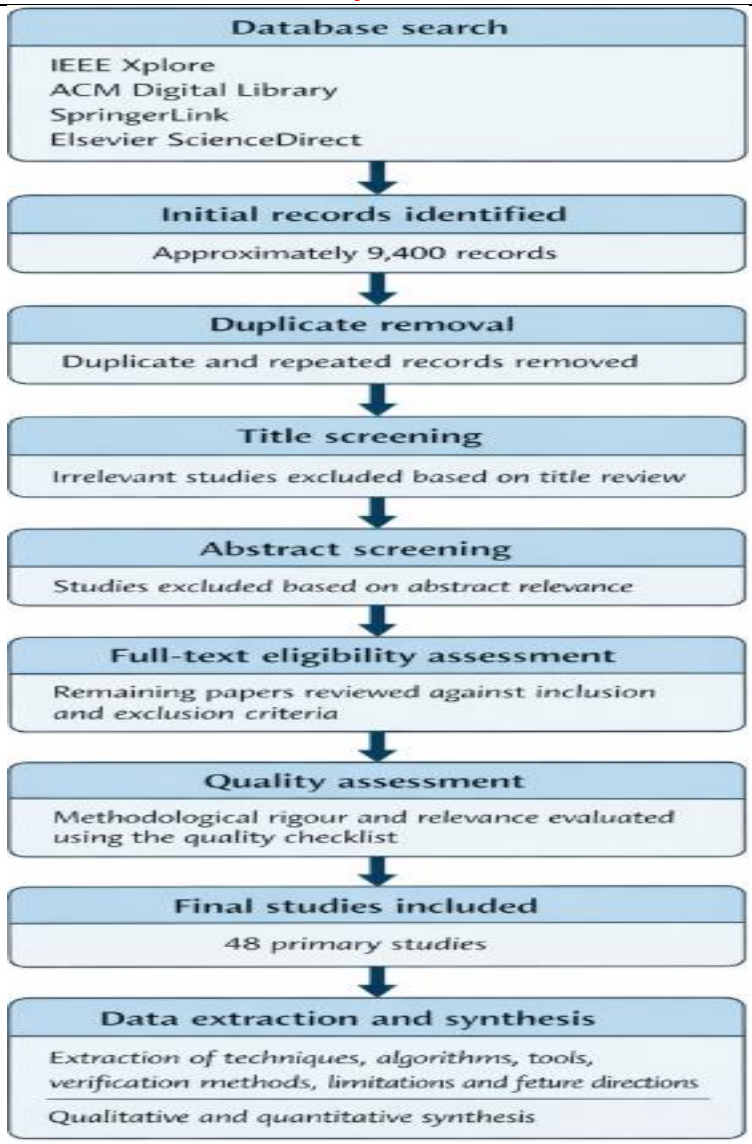

Figure 2. Systematic literature review workflow for study identification, screening, eligibility assessment, quality appraisal, and synthesis

Review Protocol Development:

A review protocol was developed before the main search, screening, and synthesis stages were undertaken. In systematic literature reviews, a protocol is important because it reduces ad hoc decision-making and helps ensure that study selection and interpretation are guided by pre-defined analytical aims rather than by convenience or retrospective adjustment. This was especially necessary in the present study because automated unit test generation is a broad and fast-moving area that spans traditional search-based methods, LLM-supported generation, assertion improvement, repair mechanisms, and verification-oriented pipelines. In the absence of a protocol, the probability of inconsistency in the selection of the study, as well as the poor alignment between the title and the research questions, and the ultimate synthesis, would be higher [28][30].

The protocol employed in this review predetermined the key aspects of the SLR. These were the study scope, the four research questions, the eligibility criteria, the search strategy, the process of study-selection, the logic of quality-assessment, and the data-extraction and synthesis structure. To make sure that the review did not turn into a loose narrative summary of the recent publications it was necessary to set out these elements in advance so that so that

it would remain a structured, evidence-based exploration of automated unit test generation procedures, algorithms, tools and verification mechanisms. This becomes paramount especially considering that the client insists that the 48 articles chosen should stand out clearly in the review format and not be shrouded by the general discussion.

The protocol was also created in such a way that it would maintain a close direction to the analytical concentration of the study. Since the title explicitly mentions techniques, algorithms, and tools, the review design had to ensure that these areas were not disproportionately emphasized. Based on this, the protocol then mapped the review process to the four research questions at the initial stages. RQ1 has been concerned with NLP- and LLM-related methods used in automated unit test generation. RQ2 was on how generated tests are verified, optimized or strengthened. RQ3 analyzed the tools and frameworks presented in the chosen studies along with their input and output supported. RQ4 was concerned with the algorithms for converting source artifacts into executable test outputs. The significance of this mapping was that among the issues raised by reviewers was the fact that, the previous version of the manuscript did not present results section in a manner that was close enough to meet the aim of the study and the title of the study. The protocol therefore served as both a procedural guide and an analytical framework.

Meanwhile, the protocol was to be practical, and not artificially fixed. The research on the automated generation of unit tests is very heterogeneous and the recent research vary in their place of publication, language of interest, benchmark design, usage of models, and logical basis of evaluation. That is why, the protocol had been applied to the review in order to guide it in a consistent, though not mechanic way. Borderline studies were evaluated with great caution as far as their contribution to the review question is concerned and both the methodological and analytical relevance considered to make a decision. This assisted in maintaining breadth without compromising rigor.

In general, the review protocol was the methodological basis of the further steps of the SLR. It made sure that the study was carried out in a clear, sequentially arranged sequence of decisions, and did not lose the original evidence organization, which was premised on the chosen 48 primary studies. The other subsections describe the operationalization of this protocol by use of eligibility criteria, search design, study selection, quality assessment and data extraction.

Inclusion and Exclusion Criteria:

To ensure that only high-quality and relevant researches were included, clear inclusion and exclusion criteria have been outlined.

Relevant to the subject:

Only the ones that discuss automated unit test generation directly, NLP based software testing, or LLM driven testing techniques were included. All the chosen studies should help answer one of the established research questions. Research that was not connected to automate testing or was not related to another set of domains was excluded.

Publication period (2018-2026):

All the included studies were published in 2018-2026. In this era, the development and rapid increase in the use of LLMs in test generation is recorded as evidenced by recent surveys that indicated that post-2018 NLP-based and LLM-assisted test-generation studies exhibited swift growth.

Publisher:

The studies were only peer-reviewed articles published in IEEE, ACM, Springer, and Elsevier online databases. These publishers are known for their strict peer-review processes a strict review process and quality research [31][32][33][34].

Crucial effects:

Studies which are included should have a meaningful contribution, like a new method, tool, algorithm, framework, or empirical assessment, which can be used to show a quantifiable increase in automated unit test generation or verification.

Results-oriented:

The chosen literature should contain empirical validation, as this may be experiments, benchmarks, mutation testing, case studies or user studies. The studies that were not well-validated and were based purely on the idea were filtered out.

Repetition:

When several publications wrote about the same, or almost similar research contributions, only the most complete or latest one was taken to prevent duplication.

Search Process:

A structured and transparent search process was implemented to identify relevant studies for inclusion in this review. In systematic literature reviews, the design of the search strategy is critical because it directly influences the completeness, reproducibility, and credibility of the final evidence base [28][29]. This was particularly important in the present study due to the rapid expansion of research on automated unit test generation, especially with the emergence of NLP- and LLM-based approaches. Without a carefully defined search process, there would be a significant risk of either omitting relevant studies or including a large volume of loosely related work that does not directly contribute to the research questions.

The search strategy was created according to the description of the review protocol in Section 2.1 and was based on the four research questions. In order to cover both old and new events, several scholarly databases were reviewed among them being IEEE Xplore, ACM Digital Library, ScienceDirect and SpringerLink. These databases were chosen as they collectively have excellent coverage of software engineering, artificial intelligence and software testing literature. The multiplicity of sources was to reduce bias in the database, and to include studies published in various publications, such as conferences and journals, both of which are of high significance in software engineering research.

The iterative derivation of the search terms was based on the key concepts of the study: automated unit test generation, techniques, algorithms, tools, and verification. These central ideas were extended with the help of synonyms and related words to make sure that many ideas could be retrieved. Search terms were automated unit test generation, test case generation, unit testing automation, LLM-based testing, NLP test generation, test generation tools, test repair and test assertion generation. These terms were systematically combined using the Boolean operators (AND, OR). As an illustration, (unit test generation AND LLM) and (automated testing AND test case generation AND tools) were some of the combinations which were applied to identify general and specialized studies.

In order to enhance the transparency of the search and illustrate the extent of the initial search, Table 1 presents the major single-term search queries along with the Boolean search queries that are used in the chosen databases and the initial numbers of the results represented.

These retrieval volumes reflect the level of the topic area and not necessarily the ultimate evidence base; these were then reduced by subsequent stages of screening, eligibility and quality screening of the results to create the final corpus of 48 primary studies.

Note: Table 1 displays the initial retrieval volumes of focused keyword and Boolean searches of the four main databases consulted in this review. These records are merely the general results of a search stage, which were then narrowed down to duplicate elimination, title and abstract filtering, full-text inclusion and exclusion, and quality selection to ultimately identify the 48 primary studies selected. Record counts which are shown in the database as values of 1,000,000+ do not indicate the database records that can be downloaded, but the upper-limit values.

Table 1. Search terms and initial retrieval results across selected databases

Sr. #	Search term / query string	Query type	IEEE Xplore	SpringerLink	ScienceDirect	ACM Digital Library
1	Large Language Model	Single term	40,331	3,676,024	785,517	311,322
2	Automated Test Generation	Single term	9,149	187,152	289,945	138,227
3	Unit Test Generation	Single term	19,793	467,785	1,000,000+	196,590
4	Mutation Testing	Single term	8,688	218,495	1,000,000+	22,344
5	Assertion Generation	Single term	588	81,526	62,531	51,259
6	Test Verification	Single term	40,669	351,867	1,000,000+	281,869
7	JavaScript Testing	Single term	2,710	18,011	13,278	16,831
8	LLM-based Testing	Single term	436	9,214	18,511	12,131
9	ChatGPT	Single term	3,234	21,506	36,449	8,479
10	"Large Language Model" AND "Unit Test Generation"	Boolean AND	330	218,687	14	105,963
11	"ChatGPT" AND "Automated Testing"	Boolean AND	141	3,629	119	4,182
12	"Test Verification" OR "Test Optimization"	Boolean OR	159,247	44,777	Not reported	409,463
13	"Mutation Testing" AND "LLM"	Boolean AND	56	602	37	1,084
14	"Fault Detection" AND "Unit Testing"	Boolean AND	3,250	22,936	381	104,608
15	"Assertion Generation" AND "ChatGPT"	Boolean AND	7	329	5	1,781
16	"Test Oracle" AND "Automation"	Boolean AND	241	4,882	292	10,164
17	"JavaScript" AND "Test Generation"	Boolean AND	417	9,087	99	16,539

18	"Dynamic Languages" AND "LLM-based Testing"	Boolean AND	50	1,164	1	7,150
19	"Adaptive Loop" OR "Self-Healing Tests"	Boolean OR	39,214	88,580	2,087	124,044
20	"Mutation Coverage" AND "Automated Unit Test"	Boolean AND	97	4,229	1	4,030

The process of search was not the strictly mechanical one, but the scanning and the improved one with time. The early search results were checked to find out other keywords, other terms and new topics in the literature. Such refinement was needed especially since in current research, terminologies used in reference to either LLM-based or hybrid testing methods are inconsistent. In particular, other studies characterize their work as the generation of tests, whereas others characterize analogous work as the improvement of tests, repairing tests or generating assertions. The inclusion of these variations in the search strategy assisted in making sure that the relevant studies were not being left out because of the differences in the terminology. The necessity of a broad terminology used in search of this area is also supported by NLP-assisted software-testing mapping research, since the area can find studies that describe similar automation issues with different terminology [35].

In order to stay current and keep up with the emphasis on recent events, a publication date filter (2018 2026) was used. This choice was supported by the fact that the creation of automated unit tests has evolved rapidly in the course of this time, especially due to the impact of large language models, which have changed the research field dramatically during this time. The confinement to recent years also serves as a way to comply with the reviewer demand to be more engaged with the recent literature and has the advantage of making sure that the results of the research are dependent on the current trends in methodology and not outdated ones.

After searching databases, multi-stage screening was done. To begin with, duplicates were eliminated. Second, titles and abstracts were filtered with the inclusion and exclusion criteria mentioned in Section 2.1.1. The articles that seemed to be relevant or could be relevant were selected to be reviewed in full. Third, full-text screening was performed to ensure the eligibility in terms of methodological relevance, contribution to automated unit test generation, and correspondence to the research questions. During this process, special focus has been made to the fact that studies that led to techniques, tools, algorithms, and verification mechanisms, were retained, as per the analytical framework of the review.

Besides searching databases, backward and forward snowballing, when necessary, was used. Such a methodology aligns with the advice of software-engineering reviews, which acknowledge that snowballing and proper attention to larger literature can enhancement in the coverage of complex and high-paced areas of research [35] [36]. Key study reference lists were also reviewed to find out the other relevant papers and citations of influential studies were also looked into. This was done since there might be a few relevant contributions that might not necessarily be retrieved by just database queries, especially recent conference papers or new preprints.

The ultimate result of this search was 48 primary studies that were identified and selected, and which constitute the evidence base of this review. This was a systematic method by which these studies were not picked just anyhow, but by an open and replicable method that was in tandem with the guidelines of a systematic review. Simultaneously, the search terms

and screening decision were narrowed and refined over time, contributing to the fact that the ultimate list of studies encapsulates the breadth and dynamic nature of the research on automated unit test generation.

Study Selection Procedure:

The final set of studies that are to be included in this review was selected through a structured study selection process, which followed the search and retrieval process outlined in Section 2.1.2. This step was necessary since the wide search findings as indicated in Table 1 needed to be narrowed down to a smaller and analytically meaningful evidence base which matched the research objectives as well as the four research questions.

The selection criteria was based on multi-stage screening methodology following the PRISMA 2020 guidelines. To start with, the records retrieved in the chosen databases were summed and the records with the same studies were eliminated to make sure that the same study was rated once. This was especially significant since studies in software engineering tend to be published in more than one database or on a conference and journal level.

Second, the inclusion and exclusion criteria outlined in Section 2.1.1 were used to conduct screening of title and abstract. This step eliminated studies that were clearly irrelevant, such as ones that centered on general software testing, with no explicit automated unit test generation component, that were purely conceptual papers without technical basis or empirical evidence, or that were not within the specified period of publication. Borderline studies were not excluded because they were sent into full-text review to decrease the chances of missing out on an applicable work.

Third, full-text screening was conducted to determine the real contribution of each remaining study to the analytical aspect of the review: techniques, algorithms, tools/frameworks, and verification or improvement mechanism. Research papers were selected based on the criterion of substantive methodological description, and definitive evidence of evaluation, including benchmark-based testing, mutation analysis, execution success, coverage improvement or empirical validation. This helped to make sure that the end result of the evidence base justified the technicality of the title as well as the quality emphasis of the review.

Selective backward and forward snowballing involving study of references and citation of significant papers, particularly those that covered TLM-based test generation, assertion generation, and mutation-guided improvement were also used. This was aimed at minimizing the chances of not getting the necessary studies because of the indexing limitations in the database or change of terminology.

This multi-stage process resulted in the work being 48 primary studies, the foundation of the analysis in this review. The process aimed not to increase the number of included papers, but to make sure that the resulting corpus was both methodologically relevant and analytically coherent and directly aligned to the research questions and result structure.

The entire study selection process, identification, screening, eligibility assessment, and ultimate inclusion are depicted in Figure 3, with the flow structure being PRISMA-like. This number gives a graphic overview of how the original retrieval set of large size was narrowed down to the corpus of studies that were ultimately taken. This systematic and open-minded selection procedure is vital to keep, especially considering the apprehension of the client that the SLR protocol and study mapping must be distinguishable and traceable in the entire paper instead of being substituted by the all-too-narrative discussion.

All in all, the selection of studies made the study review systematic, transparent, and reproducible, as well as retained the integrity of the original evidence base of 48 studies. Adding structured screening steps with close attention to relevance and contribution, the process gives an arguably sound basis to the further analysis and synthesis of automated unit test generation research.

Figure 3. Detailed study selection process illustrating database retrieval, staged screening, eligibility assessment, and final inclusion of 48 primary studies

Quality Assessment:

To determine the credibility and reliability of the chosen studies, a procedure involving quality assessment was used [37]. All the studies were evaluated concerning the clarity of purpose, correctness of methods, validity of the validation methods used, and clarity of the reported results. Earlier screening phases excluded studies which did not support their claims with empirical evidence or did not have a methodological rigor.

In order to be sure that the most recent and relevant research is included in the search, a publication-year filter was selected 2018-2026 during search. This time period was chosen in order to represent the current developments in the area. Figure 4 demonstrates the distribution of the chosen studies on a yearly basis.

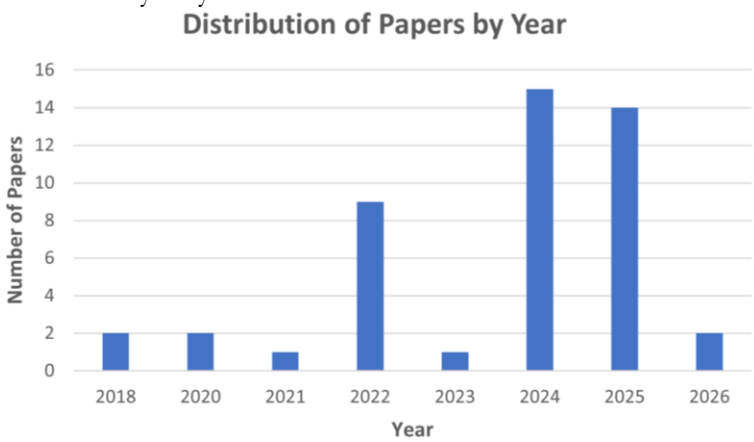

Figure 4. Distribution of selected studies by publication year

As shown in Figure 4, the selected studies are concentrated strongly in the most recent years, particularly from 2022 onwards, reflecting the rapid rise of LLM-based and hybrid approaches in automated unit test generation research.

Data Extraction and Synthesis:

Following study selection and quality assessment, data extraction and synthesis were undertaken systematically to ensure that the final review remained both traceable and analytically coherent. In systematic literature reviews, this stage is critical because it determines how heterogeneous primary studies are translated into structured evidence base capable of answering the review questions in a transparent and reproducible way [28][30]. This was especially essential in the current study due to the fact that the 48 articles of interest have a significant difference in their research focus, technical contribution, formulation of validation strategy, and writing style. While others provide tools or frameworks, others are interested in generating algorithms, others in verifying or mutation-guided refinement, and others in assertion generation or repair. There was thus a need to extract and synthesize this diversity in a structured way so that the evidential value of the diversity could be preserved as opposed to reducing it to a simplified narrative summary. The extracted data elements and synthesis structure are summarized in Table 2.

Table 2. Data extraction and synthesis details for selected studies

S. #	Data Extraction / Synthesis Element	Description of Extracted Information
1	Research Identification	Title, authors, year of publication, type of publication (journal, conference, workshop), and database (IEEE, ACM, Springer, Elsevier).
2	Relevance to Research Questions	Identification of whether the study addresses one or more of the four research questions (RQ1–RQ4).
3	NLP / LLM Techniques Used	Extraction of NLP techniques (e.g., semantic parsing, intent extraction, IR-based methods) and LLM capabilities (e.g., GPT-4, CodeT5+, StarCoder).
4	Verification and improvement Methods	Extraction of validation strategies used in the study, including mutation testing, assertion enhancement, repair loops, mocking, and coverage evaluation.
5	Tool / Framework Identification	Identification of developed tools or reused frameworks, including tool inputs (e.g., source code, test intentions) and outputs (e.g., unit tests, assertions, mutants).
6	Algorithmic Approach	Extraction of core algorithm(s) used (e.g., evolutionary search, probabilistic type inference, RL-based healing, method slicing).
7	Category Mapping	Mapping of each research work to one or more of the five predefined categories listed in Section 2.1.
8	Key Contribution Analysis	Identification of the primary contribution of the study: tool, methodology, algorithm, empirical evaluation, or theoretical advancement.
9	Limitations Identified	Extraction of challenges and limitations reported in the study (e.g., weak assertions, hallucinations, low mutation scores).
10	Future Directions	Extraction of recommendations or open research problems suggested by the authors.

A predefined framework was used to extract the data in accordance with the study title and the four research questions. Bibliographic and contextual details (title, authorship, year of publication, type of publication, and source database) of each paper were initially captured. It was then narrowed down to the key analytical dimensions of the review: automated unit test

generation methods, the underlying algorithm or transformation strategies, tools/frameworks and verification or improvement mechanisms. This was to make sure that the chosen 48 articles were reviewed in a consistent way and that they were related to the scope of the review.

To facilitate correspondence with the research questions, the extraction process also extracted the input artefacts and outputs that were reported in both studies. The artefacts of input were source code, natural-language specifications, prompts, existing test suites and the artefacts of output were executable unit tests, assertions, repaired tests, amplified suites and evaluation metrics. This difference was significant since research described as automated unit test generation can vary in its value: some test generation, some test repair or test improvement, others assertion testing, oracle testing, mutation-assisted refinement.

Where feasible, validation and outcome measures were also extracted. These were coverage, mutation score, pass rate, executability, assertion accuracy and empirical usability. The inclusion of such measures in the records served to differentiate between the studies that only reported the test generation being successful and those that depicted more robust evidence of the quality of test, error detection, or usefulness in practice. Limitations and future directions were also reported to facilitate the balanced synthesis of the strengths and weakness of the evidence base.

Qualitative synthesis and descriptive aggregation were the methods of data synthesis that were used to extract the data. The interpretation of the means various studies conceptualized automated unit test generation and how it could be done technically was done by qualitative synthesis and the summarization of category distributions, research-focus patterns, publication-year trends and the relative importance of methods, algorithms, tools, verification and improvement mechanisms were summarized by descriptive aggregation. The variability of benchmarks, programming languages, versions of models used, prompting strategies, and validation designs across the included studies meant that similar results were not pooled, but were interpreted with caution.

The entire process of data extraction and synthesis made sure that the review was evidence-based, question-oriented, and traceable. It maintained the mapping of the 48 studies (study level) and formed the basis of the results in the subsequent section.

Results:

The following section gives the results of the systematic literature search of the 48 primary papers that were chosen to be analyzed. As per the title of the review and the adjusted methodological structure, the results are grouped in terms of key analytical dimensions of techniques, algorithms, tools/frameworks, and verification or improvement mechanisms. The purpose of this structure is to ensure that one of the most important strengths of the review is that it presents the mapping of the selected studies to these categories, and not a reduction of the evidence base to a more narrative discussion. Meanwhile, the section is updated to better address the reviewer feedback by providing more direct answer as to how the results relate to the research questions and the overall problem statement.

Through the chosen articles, it can be seen that there is a shift of the literature to less discrete and more integrated and pipeline-style of automated unit test generation. The focus on search-based exploration, probabilistic type inference, and model-driven generation (still present in the contemporary work of the review corpus) is still evident in the context of the primary research objective, and generating executable tests or enhancing structural coverage. The later section of evidence basis, however, is becoming influenced more and more by workflows based on LLM that integrate generation and refinement, repair, assertion enhancement or mutation-directed feedback. This change is important as it implies that the field ceases to develop primarily through generating more tests but rather tries to generate tests more applicable, more context sensitive, and more justified in quality [13][38][14].

The second pattern which stands out of the corpus of 48 studies is the asymmetry between generation-oriented contributions and validation-oriented contributions. The highest number of studies stays in tools and frameworks, the next number of studies is dedicated to test-generation methods and algorithms, but to a smaller extent, there are studies dedicated to verification, evaluation, assertion quality, repair, or post-generation improvement. This imbalance is analytically significant as it shows a persistent imbalance in the literature: automated unit test generation has grown more rapidly as a production issue, rather than as a quality-assurance issue. That is to say that the field has improved its ability to produce tests rather than to prove repeatedly that they are fault-revealing, stable and strong enough in terms of oracle quality. This tension is especially obvious with recent developments on mutation-guided refinement and assertion generation, as multiple studies report promising coverage or executability enhancements but still observe limitations in behavioral strength, reproducibility, or validation depth [14][18].

The findings also indicate that automated unit test generation cannot be viewed as a single research field. Not only do the selected studies have different approaches of technical approach, but so do what they consider the main issue to resolve. Other concentrating on the generation of tests directly based on source code or prompts; others concentrating on the extraction of information based on natural language artefacts; others concentrating on mutation-based or coverage-based testing; and others on strengthening assertions, repairing failing tests or enhancing existing test suites. That is why, the studies are not presented in this section of results as the ones that are all solving the same task under the same conditions. Rather, the findings are synthesized in terms of category-based subsections in such a way that they maintain the differences between what is created, how it is created, how it is evaluated and how it is enhanced. This is methodologically significant since recent research in the area is prioritizing the integration of multiple of these functions across the same workflow, particularly in the case of LLM-based systems [3][4]).

The other significant observation that can be observed in the chosen studies is the increasing presence of post-generation control mechanisms. The more powerful new systems seldom are based on one-shot generation. Rather, they frequently add refinement cycles, performance feedback or execution, mutation-driven prompting, repair tactics or assertion augmentation. Such a tendency shows that the dynamics of center of progress in the field have changed. Contributions with the greatest potential to produce the most tests do not always yield the greatest number of tests, but those that combine generation with mechanisms to filter, strengthen or validate results. This is in line with recent industrial and empirical evidence indicating that the practical value inheres in the ability of generated tests to endure realistic conditions of compilation, execution, review, and maintenance, and not with generation fluency per se.

To capture these patterns vividly, the results are provided in a staged manner. The part begins with a review of the chosen literature and how it falls into the primary areas of interest under research. It then looks at tools and structures, then test generation methods and algorithms, verification and evaluation methods, and lastly assertion, repair and test suite improvement methods. This structure enables the review to stay in close connection with the original 48-study evidence structure, as well, answering the reviewer need to have a better mapping between result subsections and the review title with the research questions. In general, the findings indicate that it is a rapidly developing field, although not evenly: it is stronger technically than ever, but still suffers from the lack of validation, oracle quality, comparability, and practical robustness.

Overview of Selected Studies:

In the last corpus of 48 primary studies, one can see that no single technical tradition leading to automated unit test generation is relevant anymore. Rather, the chosen literature

indicates an area of transition, whereby conventional search-based, model-based and probabilistic approaches are becoming more and more coexist with workflows powered by LLM and hybrid ones. This alone is an important discovery since it demonstrates that new developments should not be seen as an outright substitution of old practices by new ones. Instead, the picture indicates that the discipline is shifting to increasingly composite pipelines where test generation, validation, refinement, and assertion support are considered part of a consistent process as opposed to a standalone activity. This larger tendency can be observed in the chosen studies and is one of the most evident reasons why the current review needs to synthesize methods, algorithms, tools, and verification mechanisms as a package and not to comment on them individually.

Table 3. Distribution of selected studies by research focus

Research Focus	References	Number of Studies
Tools & Frameworks	[39][40][41][42][43][44][26][45][46][47][48][49][50][51][52][53][54]	27
Test Generation Techniques & Algorithms	[55][46][25][53][56][57][11][9][58]	14
Test Verification & Evaluation	[59][60][61][24][11]	10
Assertion, Repair & Improvement	[17][40][47][16][18][50][51]	8

As Figure 5 indicates, the greatest percentage of the studies concentrated on tools and frameworks to generate unit tests automatically, then test generation methods and algorithms, verification and evaluation methods, and assertion, repair and test suite enhancement mechanisms.

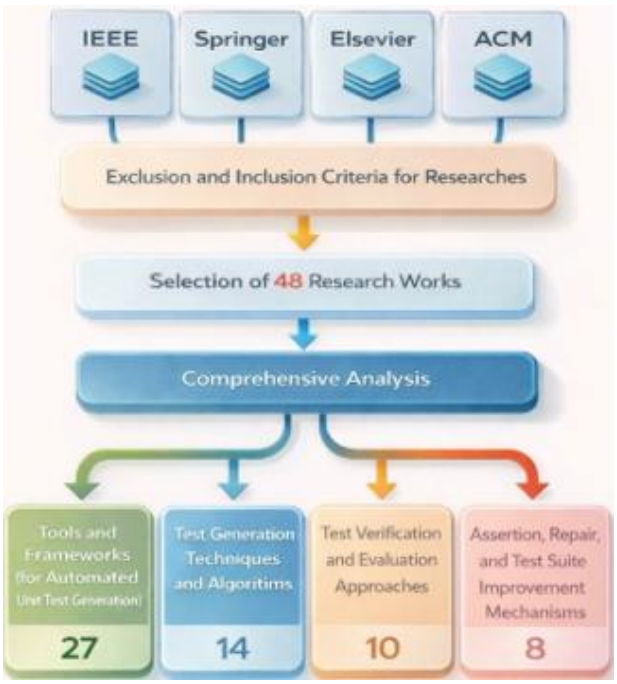

Figure 5. Overview of research selection and category distribution across the 48 included studies

Categorically, the largest proportion of the chosen studies is in the tools and frameworks which represent 27 out of the chosen 48 studies (56.3%). This superiority suggests the literature is highly implementation-based. Several more recent contributions go beyond suggestion of an abstract method and encapsulate their ideas into a form of executable system, toolchain or even workflow at the framework level. Examples are ChatUniTest, TestART, MuTAP, SynTest-JavaScript, Nessie, and AsserT5, which demonstrate the growing importance of operationalization and evaluation as well as conceptual novelty in the field [62][63]. This focus is important, but it also shows an imbalance: the field seems to have gone more into creating generators and support systems than into creating equally mature frameworks to demonstrate the behavioral strength of the generators by the support systems.

The second most numerous group is the test generation techniques and algorithms, which include 14 studies (29.2%). The studies are of particular importance since they show methodological diversity of the field. On the one hand, conventional methods still use the search-based exploration, probabilistic type inference, and logic-based generation to generate executable tests in a constrained or dynamically typed environment. Meanwhile, more recent work uses prompt-based, feedback-driven, and hybrid strategies that utilize richer program context and natural-language indicators, generally relying on LLMs. It is this kind of tension that the state of the current automation unit test generation research can be regarded as: On the one hand, methods of unit test generation are becoming increasingly context-sensitive and semantically expressive; on the other hand, they are increasingly relying on the surrounding control mechanisms like slicing, refinement, or mutation feedback to ensure that the generated unit tests are reliable in practice.

A less analytically critical but smaller number of studies is dedicated to test verification and evaluation (10 studies, 20.8%). The relatively less figure in this category is telling. It indicates that despite the rapid advancement in automated unit test generation, relatively little research focuses on addressing the question of whether generated tests themselves are strong, fault-revealing, and trustworthy. This discrepancy is especially significant since a number of recent contributions directly question the suitability of coverage as a measure of success. Comparative evaluation work and mutation-oriented work demonstrates that generated tests can look to work well structurally, but be weaker than expected in behavioral fault detection. This is to say that the field has grown to be better at producing tests than it is at either proving that those tests have strong oracles or reveal any meaningful errors [59][61]. The comparative lack in size of this category then indicates one of the fundamental weaknesses of the existing literature.

The fourth group includes assertion generation, repair, and test suite improvement processes, which can be illustrated by 8 studies (16.7%). This category is less numerically important, but conceptually significant in that it is where much of the new maturity of the field has been emerging. Research in this field recognizes that first-test generation can be partial, poor or fragile and that useful automation is more and more pegged on the post-first-pass generation. The primary emphasis of assertion-oriented systems like A3Test, AsserT5, and similar oracle-enhancement methods demonstrates that the main difficulty is not to write more test code, but to reinforce the criteria of correctness interwoven into that code [47]. Similarly, the refinement and repair-driven work implies that the next phase of the field might not be as much about fully autonomous generation as about improvement pipelines where the refinement effort is decreased and the quality of practical tests is directly improved.

The other interesting trend that is evident among the chosen studies is the temporal concentration. This last corpus is strongly skewed towards the last few years, particularly since 2022, which corresponds to the fast growth of LLM-based and hybrid methods in automated generation of unit tests. This concentration is not simply a question of volume of publication, but also a change of approach. Current research is increasingly considering test generation as

a larger quality-engineering concern that includes contextual reasoning, refinement loops, assertion generation, and industrial usability than it is a more limited task of code-generation. The emergence of more recent empirical and industrial research, such as into mutation-guided prompting and industrial-scale test improvement, also indicates that the subfield is no longer simply questioning how tests can be generated, but under what circumstances they can be trusted and incorporated into realistic workflows.

Simultaneously, crucial limitations to the evidence base are also revealed in the overview of the chosen studies. The corpus under review contains a lot of technical experimentation, though it is less convincing in terms of evaluation criteria, the level of reporting, and study to study comparability. There is a wide difference in the target programming languages, benchmark settings, and prompt designs, tool environments, and outcome measures across studies. Consequently, the discipline seems to be productive in methods but not yet very consolidated. That is why the current review does not consider the 48 studies as a consistent body of evidence. Rather, they are viewed as a dynamic and unbalanced research landscape where progress can be observed, though still bound by patchy validation practices, intractable oracle-quality issues and relative scarcity of standardization across experimental environments.

The chosen studies combined demonstrate a domain that is not only expanding in technical capability but also in the complexity of methods. The evidence base is dominated by tools and frameworks; this is numerically, yet, verification and post-generation quality mechanisms are still underrepresented in comparison with their significance. This implies that the literature is developing but not with evenness. Automated unit test generators are increasingly more complex and more realistically ambitious, but remain characterized by an ongoing divergence between performance and quality assurance in generation. The degree of that gap gives the explanation to the more circumstantial category-based analyses which follow in the following subsections.

In order to render explicit, the analytical structure of the findings, Table 4 projects every result subsection to the respective research questions and title aspects that the review tackled.

As illustrated in Table 4, every results sub-section is directly associated with either one or more research questions and offered with the title items of techniques, algorithms, and tools. This format makes the results part analytical, traceable and coherent to the overall intent of the review.

Table 4. Mapping of result subsections to research questions and title elements

Result subsection	Main research question(s) addressed	Main title element(s) addressed	Purpose of subsection in the review	Expected evidence base
3.1 Overview of Selected Studies	RQ1, RQ2, RQ3, RQ4	Techniques, Algorithms, Tools	Gives an overview of the corpus-level details of 48 chosen studies, such as their research-centric categories, distribution of the publication years, general shift to the use of LLMs and blended models. It sets the analytical background of subsections which follow in detail.	Study distribution table, research-focus classification, publication-year pattern, category counts
3.2 Tools and Frameworks for Automated Unit Test Generation	RQ3	Tools	Interviews the instruments and frameworks documented in the chosen researches, its reported inputs, outputs, and validation methods. In this subsection, the practical and implementation-focused aspect of automated unit test generation is brought into the limelight.	Tool/framework mapping table, input-output structure, validation methods, implementation orientation
3.3 Test Generation Techniques and Algorithms	RQ1, RQ4	Techniques, Algorithms	Summarizes the ideas and methods of automated unit test generation, such as the traditional search-based, the NLP-assisted, the LLM-based prompting, and the combination of the generation strategies. It describes the process of converting input artefacts into test outputs.	Technique/algorithm table, NLP/LLM mapping table, generation strategies, produced artefacts
3.4 Test Verification and Evaluation Approaches	RQ2	Techniques, Tools	The approaches to test verification, test evaluation, and optimisation, including mutation testing, coverage-based evaluation, empirical comparison, and executability, fault-revealing effectiveness. In this subsection, the chasm between test generation and reliable validation is discussed.	Verification/evaluation table, mutation- and coverage-related evidence, validation metrics
3.5 Assertion, Repair, and Test Suite Improvement Mechanisms	RQ2, RQ3, RQ4	Techniques, Algorithms, Tools	Discusses post-generation techniques that enhance the quality of tests such as assertion generation, oracle enhancement, repair, healing, amplification and refinement strategies. In this subsection, the increasing treatment of	Assertion/repair/improvement table, oracle-support studies, refinement mechanisms

			automated testing as an iterative improvement process is demonstrated.	
4. Answers to Research Questions	RQ1, RQ2, RQ3, RQ4	Techniques, Algorithms, Tools	Combines the evidence reported in the entire Section 3 to direct responses to the four research questions. This last mapping phase is to ensure that the results are clearly aligned to the objectives, research questions and title of the review.	Cross-reference to all result tables and category-based evidence

Tools and Frameworks for Automated Unit Test Generation:

The corpus of 48 studies reveals that the biggest category of research oriented is represented by tools and frameworks, comprising 27 studies (56.3%), which shows that automated unit test generation is not merely a methodological research domain but also highly implementation oriented. This trend is significant since it implies that the research on the topic has shifted away from informing about abstract generation conceptions and has tried more and more to bring them operationalisations as a system, pipeline, or workflow element. Simultaneously, this numerical hegemony must not be viewed as a very positive phenomenon without any reservations. There seems to be a lag in the literature in advancing generators and systems to support them as much as there has been in defining an equally mature standard of validation of the quality, robustness and industrial stability of their products. The said imbalance is among the most evident themes that come out of the chosen works.

Table 5. Tools and frameworks for automated unit test generation

Sr. #	Paper	Input	Tool	Output	Validation
1	Olianas et al., 2022	UML Model	MATTER	Test Scripts	Experimental Validation
2	Arteca et al., 2022	Source Code	Nessie	Test Cases	Experimental Validation
3	Olsthoorn et al., 2024	Source Code	SynTest-JavaScript	Unit Test Cases	Experimental Validation
4	Stallenberg et al., 2022	Source Code	GuessWhat	Unit Test Cases	Experimental Validation
5	Guo et al., 2024	Source Code	PC-TRT	Test Cases	Experimental Validation
6	Chen et al., 2024	Source Code	ChatUniTest	Unit Test Suites	Experimental Validation
7	Gu et al., 2024	Source Code	TestART	Test Suite	Experimental Validation
8	Tip et al., 2025	Source Code	LLMorpheus	Mutants	Experimental Validation
9	Abdi et al., 2022	Existing Tests	Small-Amp	Amplified Tests	Experimental Validation
10	Zhu et al., 2023	Unit Test Code	StubCoder	Stub Code	Experimental Validation
11	Alagarsamy et al., 2024	Source Code	A3Test	Assertion Tests	Experimental Validation
12	Mali et al., 2024	Natural Language	ChIRAAG	Generated Assertions	Empirical Evaluation
13	Primbs et al., 2025	Source Code	AsserT5	Generated Assertions	Empirical Evaluation
14	Khandaker et al., 2025	Source Code	AugmenTest	Generated Assertions	Experimental Validation
15	Mani & Attaranasl, 2025	Failing Tests	Adaptive Test Healing	Repaired Test Cases	Empirical Evaluation
16	Schäfer et al., 2024	Source Code	TestPilot	Test Suite	Experimental Validation
17	Dakhel et al., 2024	Source Code	MuTAP	Test Suite	Experimental Validation
18	Zhang et al., 2025	Source Code	TestCTRL	Test Suite	Experimental Validation

The first difference that can be observed throughout the tools reviewed is between traditionally/structurally guided systems and LLM-enabled systems or hybrid systems. The classical one, which is represented by the chosen literature and is summarised in the tools table, consists of MATTER, Nessie, SynTest-JavaScript, GuessWhat, PC-TRT. These systems are typically run on structured artefacts like source code, program models, or already existing test assets and their results are usually executable unit tests, scripts or reused test cases. Their main advantages are that they explicitly explore logic, procedural openness and comparatively

consistent validation in terms of structure measure like branch, path or coverage evaluation. Like that, they are still relevant since they prove that controllability and reproducibility is still relevant in automated unit test generation, particularly when the issue of dynamic program behaviour and reliability of the execution of a program are involved.

The newest evidence base demonstrates a definite transition to the frameworks, which are based on LLM and hybrid, and enlarge the input and the output space of automated testing tools. Newer systems are increasingly based on prompts, snippets of contextual code, execution feedback, mutation data and existing test resources as alternatives to only using source code or static artefacts. They also produce more than just regular unit tests; they have repaired test suites, assertions that are mutation sensitive, and artefacts to provide refinement to downstream testing. The use of ChatUniTest, TestART and MuTAP does demonstrate this shift since these tools conceptualize automated unit test generation as a controlled generation-validation-repair cycle as opposed to a single prompt-to-test event.

The other tool cluster is important in generation of assertions, generation of oracle and generation improvement. Systems like A3Test, ChIRAAG and AsserT5, AugmenTest, Small-Amp, Adaptive Test Healing and StubCoder demonstrate that tool-oriented research is not only worrying about creating new tests, but also about enhancing the quality of the existing or created tests. This directly aids the issues covered in RQ2 and RQ3, as the usefulness of a generated test is directly related to whether or not it has meaningful assertions, identifies defects, reliably executes, and can be maintained. AsserT5 is particularly of interest in that it shows advances in the production of assertions but also indicates that the production of assertion accuracy remains a persistent challenge of applying it to the overall detection of bugs.

There is also a more general direction towards hybridisation in the tools reviewed. Numerous new structures, as outlined in Table 5, are no longer easily placed into any one category like generator, verifier, or repair tool. Rather, they are integrated with functions like generation, filtering, slicing, repair, assertion support, mutation-based improvement as well as empirical evaluation. This is an indication that the design of the tools is taking to consider the real complexity of automated unit test generation. A test that is generated should have some value, that is, it should compile, be executable, with meaningful checks and be resilient to real-life integration restrictions.

In spite of this development, there are still limitations that are apparent in the literature of the tools. Validation is also still a disparate area, with most schemes reporting coverage, pass rate or executability, but fewer schemes showing good evidence on fault revelation, oracle strength or long-term maintainability. The aspect of dependency management and maintaining the stability of the environment is also a recurring challenge, particularly in LLM-based tools. Moreover, benchmark-centred evaluation and actual deployment situations still have a gap. Industrial experience of test-improvement workflows based on the LLM demonstrates that useful value is based on limited generation, filtering, quantifiable improvement and developer control instead of unlimited automation.

In general, the most glaring aspect of research in automated unit test generation is now tools and frameworks, which also reveal the main methodological conflict of the discipline. Systems and workflows are abundant in the literature, but the most effective tools are becoming more and more successful when generation is coupled with repair or mutation-guided validation, or oracle improvement. This observation forms the basis of the next subsection that will shift this perspective down to tool-level systems to the test generation techniques and algorithms on which the systems rely.

Test Generation Techniques and Algorithms:

In this subsection, the technical methods of determining the automated unit tests on the 48 studies selected are analyzed. Although Section 3.2 was concerned with tools and frameworks, the underlying generation logic (how tests are generated based on source code,

natural-language artefacts, prompts, execution feedback or even existing test assets) is the focus of this subsection. This difference is significant since the methods in tools may or may not be packaged, but the quality and usefulness of the generated tests is heavily dependent on the generation strategy and algorithmic form of the generated tests.

The papers under consideration indicate that the methods of test generation can be divided into two quite broad but more and more overlapping categories, namely, the traditional structurally guided approaches and the approaches based on LLM or hybrid approaches. Search-based exploration, evolutionary optimization, symbolic or model-guided reasoning, probabilistic inference and path- or coverage-oriented generation are examples of traditional methods. These methods are still relevant due to their explicit controls, reproducibility and exploration of programs in a systematic way. They are however restricted in generating natural test code, context sensitive, and robust assertions without the support mechanisms.

The second large type is the techniques of generation that use LLM support, such as zero-shot prompting, few-shot prompting, prompt templation, context augmentation, iterative prompting and hybrid refinement [64]. These techniques rely on semantic information of source code, prompts, and textual artefacts in order to create more context-sensitive tests. Yet, the evidence indicates that not much prompting is likely to be successful in the absence of restraint. Slicing, prompt engineering, execution feedback, mutation guidance, assertion enhancement, or repair loops tend to support the stronger LLM-based ways of generation, and make generation more consistent and more reflective of the expected program behavior.

This observation is relevant to both RQ1 and RQ4 as it indicates that it is not only through the use of sophisticated models that the generation of automated unit tests can develop but also through the organization of the generation. Guided prompting through mutation, refinement-based generation and repair workflows show that first-pass test generation can be just a step. The usefulness of a technique is due to its participation linking the context of code, which prompts logic, validation and correction. As such, the chosen works can be interpreted not merely as the evidence of the tool's development, but rather as the evidence of the more fundamental approach towards the methodological changes, in favor of the hybrid and verification-conscious generation pipelines. The identified test generation techniques and algorithms are summarized in Table 6.

The other significant trend is the significance of intermediate transformation strategies. There are a number of studies that do not directly start with input artefacts to eventual tests. They instead include intermediate steps include method slicing, type inference, related artefact retrieval, reinforcement signal, or context filtering. These strategies are important as they minimize ambiguities and instability which tend to characterize direct generation. Some such as method slicing can be used to specialize attention to the code related to a particular problem, in effect to enhance both performance in generation and coverage. The same concept is implied by Retrieval-assisted generation, which implies that the artefacts generated or other examples can enhance outputs generated by a model or generator by alleviating the task of the model or generator to learn all the information given the raw context alone. These methods support a larger conclusion: the most powerful methods of the modern automated unit test generation are not necessarily direct generators, but rather well-thought-out mediated transformation pipelines.

The chosen articles also indicate that techniques based on NLP are still up-to-date even outside mainstream prompting of LLM. Despite the recent popularity of LLMs, a few studies are also based on more traditional NLP techniques like requirement parsing, information retrieval, and structured textual analysis when test cases or assertions need to be gleaned out of requirements, user stories, or mixed natural-language artefacts. This has an analytical significance in that it extends the concept of automated unit test generation into the context of more than just a source-code-only setting. It also demonstrates that language-based

automation in testing did not start with LLMs; instead, LLMs generalize and magnify an older tradition of work that is interested in such transformation of textual data into formal or semi-formal test artefacts. In this regard, the literature analyzed proposes continuity and disruption: the recent LLM practices open up the field, but at the same time, the practice occurs within an extended history of NLP-assisted software-testing studies. The NLP-based techniques extracted from the selected studies are summarized in Table 7.

Table 6. Test generation techniques and algorithms reported in selected studies

Sr. #	References	Input Artifact(s)	Technique / Algorithm	Generation Strategy	Produced Artifacts
1	[46][25]	Source code, prompts	LLM prompting (zero-shot / few-shot)	Prompt-based generation	Unit tests
2	[58]	Source code, coverage feedback	Chain-of-thought prompting + feedback	Prompting + iterative refinement	Unit tests
3	[58][11]	Source code	Large-scale prompt evaluation	Comparative prompting evaluation	Unit tests, metrics
4	[5][6]	Source code	Search-based / evolutionary generation	Evolutionary search / optimization	Unit tests
5	[65]	Source code	Probabilistic type inference (dynamic languages)	Type inference + exploration	Unit tests
6	[63]	Source code	Search-based unit test generation (JavaScript)	Search-based exploration	Unit tests
7	[3]	Source code	Method slicing for high coverage	Slicing + generation	High-coverage unit tests
8	[58]	Source code, coverage feedback	Reinforcement learning from coverage feedback	RL-guided generation	Unit test suites
9	[15]	Source code, related artifacts	Information retrieval assisted generation	Retrieval + synthesis	Assertions / tests

Critically, the evidence base continues to have a number of weaknesses. To begin with, the successes of generation are reported in many studies without sufficiently well-evidenced oracle quality, fault exposure, or maintainability. Second, algorithmic comparisons are challenging due to the different styles of studies, conditions of prompt, model version, and benchmark sets, and validation metrics. Third, certain sections of the literature have an ongoing predilection to consider readability or executability as more strongly predictive of success than the predictive validity of these attributes. Instead of just converting inputs into test code, it is the fact that the guiding mutations or assertions must be done in a manner that maintains behavioural sense and practical utility, which is made possible by recent mutation-guided and assertion-oriented studies. The value of current generation techniques remains unaffected by these weaknesses, but they do imply that such claims of progress should not be as bold as some studies suggest.

Table 7. NLP techniques applied in automated unit test generation studies

Sr. #	Paper	Input	NLP Technique	Output	Validation
1	Alagarsamy et al., 2024	Source Code	PLBART, Masked Language Model	Assertion-Enhanced Test	Experimental Validation
2	Hajri et al., 2020	Natural Language	Requirements Parsing	Test Case Classification	Case Study
3	Garousi et al., 2020	Natural Language Requirements	POS Tagging, Parsing, NER	Test Cases	Theoretical/ Literature Review
5	Primbs et al., 2025	Source Code	Transformer Model	Test Assertions	Empirical Evaluation
6	Abrahamsson et al., 2023	Natural Language Requirements	Prompt Engineering	Source Code, Test Cases	Experimental Validation
7	Stallenberg et al., 2022	Source Code	Probabilistic Type Inference	Unit Test Cases	Experimental Validation
8	Planötscher et al., 2026	User Stories, Requirements Text	TF-IDF, POS Tagging, LLM Prompting	Requirements Models, User Stories	Theoretical / Literature Review

In general, the chosen articles demonstrate that the methods and algorithms of the test generation are increasingly semantically competent, pipeline-based, and rely on feedback and refinements. Concrete methods have not lost their relevance as they are controllable and provide a structural assurance whereas, the methods designed using LLM are expanding the range of what can be created by richer contexts. The most significant tendency, though, is the fact that neither of the two takes the upper hand, rather, they are becoming more and more integrated. Automated unit test generation is perhaps more appropriately viewed as a contest between old approaches and new approaches, but it represents a changing methodology where structural exploration, semantic inference and iterative correction are progressively integrated. This tendency also provides the reason why the following subsection, about verification and evaluation, is so vital: the more advanced the generation of the techniques, the more the necessity to evaluate what is actually generated with the help of these techniques becomes central.

Test Verification and Evaluation Approaches:

In this subsection, the answer to RQ2 will be discussed, through the way the chosen papers confirm, assess, and, in other instances, optimize the quality of automatically generated unit tests. The corpus of 48 studies comprises ten studies (20.8%) that concentrate predominantly on verification and evaluation, which is a smaller number of studies numerically but a very significant study analytically. According to Table 3, this imbalance indicates that the field has been more advanced in generating tests compared to establishing whether the tests that have been generated are actually effective, revealing faults and reliable in practice.

Through the chosen studies, mutation testing is revealed as one of the most formidable verification techniques since it is an evaluation that goes beyond mere executability and coverage. Mutation testing tests whether generated tests can detect injected faults (as opposed to simply determining the presence of generated tests running or covering code). This is particularly helpful in the generation of plausible or executable unit tests where the plausible or executable tests might be behaviorally weak. Recent research applies mutation testing as a post-generation evaluation system as well as an active system to provide feedback to be

improved or repaired. Indicatively, a mutation-guided prompting demonstrates that verification has the ability to affect the quality of the tests and not only measure it.

Coverage-based evaluation, together with mutation testing, is frequently used, particularly in the context of the search-based generation, JavaScript-based testing and benchmark-based tool comparison. Statement, branch and path coverage are reasonably helpful measures of the breadth of execution but are relatively simple to measure between tools. Nevertheless, the chosen literature also demonstrates that coverage is not to be considered as adequate evidence of the quality of tests. A suite generated can have good coverage but a poor assertion or fails to detect faults. Thus, coverage can be considered as some useful but imperfect measure of validation.

In recent studies, the reviewed articles rely on empirical comparative assessment as well, especially in the studies based on the use of LLM. The models are compared in such studies and strategies, refinement workflows, or tool settings are prompted based on such metrics like coverage, executability, mutation score, and pass rate. This can be used to compare cross-method, although the evidence is again hard to compose as studies vary in terms of the choice of benchmark, programming language, model version, prompt design, dataset characteristics and configuration to evaluate. Consequently, empirical assessment is on the rise though standardized assessment is minimal.

Other studies also use alternative or semantic evaluation strategies to uncover behavioral inconsistencies, semantic conflicts or more profound inconsistencies between generated tests and intended program behavior. These are not widely used as either mutation or coverage-based evaluation, but are significant as they transcend syntactic correctness and structural adequacy. They are also not used widely suggesting that the field does not yet have a widespread consensus on how to measure the strength of semantics on scale, particularly in settings based on LLM where outputs can be fluent and weak behavioural assurance can be concealed.

All in all, the evidence of verification and evaluation indicates that, the generation of unit tests by means of automation is increasingly becoming quality-conscious, although, validation practices are skewed. Empirical comparison, semantic evaluation and mutation testing have more evidence than just coverage, but are not always utilized in the 48 studies. The implication of this observation is that the need to standardise multi-metric assessment models further, which is discussed in the recommendations and future-work sections. The verification and evaluation approaches reported in the selected studies are summarized in Table 8.

One more crucial pattern is the one which is related to the process of verification and optimization. In some of the recent works, verification has ceased to be considered as an independent downstream. It is rather integrated into a feedback process of a control loop where bad test results are improved out by mutation feedback, repair, filtering or by improving assertions. This is a good indication of the development of a methodology in the field. It implies that assessment is turning out to be more fruitful when it is not employed to assess produced examinations, but to enhance them. Hybrid systems like mutation-guided refinement workflows and repair-oriented frameworks demonstrate that valuable automation is more and more based on such integration. Most recent approaches are thus the strongest as they do not consider verification as a passive layer of measurement, but rather as an active part of unit test generation pipelines where automated unit tests are created.

Although this has been achieved, a number of limitations are still evident in the literature related to the verification. To start with, the quality of oracle is a poorly tackled aspect compared to structural measures. Even a test that the compiler compiles successfully, executes successfully and has high coverage can fail to identify meaningful bugs, because of shallow assertions or behavioural non-congruency. Second, there is an uneven distribution of industrial

realism. A few of the tools would be good in benchmark conditions, but would have less indication of longevity or realistic maintainability in real-life development environments. Third, there is still no consensus in the literature on the desired metrics to be prioritised when there are multiple desirable properties in conflict like high-coverage, powerful fault revelation, readable output, and stability of the execution. The points of these limitations are important in the sense that they demonstrate that the main issue of the field is not how to come up with tests, but how to determine confidence that the generated tests are worth retaining.

Table 8. Test verification and evaluation approaches reported in selected studies

Sr. #	References	Verification / Evaluation Approach	Metrics Used	Evaluation Scale	Datasets / Subjects
1	[59][60][21]	Mutation testing	Mutation score, killed mutants	Large-scale	Open-source projects, industrial systems
2	[14][43]	Mutation-guided refinement	Mutation score, repair success	Medium-scale	Open-source projects
3	[48][63][25]	Coverage-based evaluation	Statement, branch, path coverage	Medium-scale	Benchmark programs, OSS projects
4	[48]	Differential testing	Behavioral inconsistencies	Medium-scale	Multiple implementations
5	[23]	Semantic conflict detection	Behavioral conflicts	Small-scale	Open-source projects
6	[46][11][25]	Empirical comparative evaluation	Coverage, mutation score, executability	Large-scale	Multiple OSS repositories
7	[59]	Industry-oriented evaluation	Practical usability, scalability	Large-scale	Industrial systems

Altogether, it seems that the studies chosen demonstrate that verification and evaluation is turned more advanced, yet less developed than the very generation. Mutation testing has become the most potent critical response to excessively optimistic arguments founded on coverage or executability, whereas the use of hybrid refinement processes is gradually showing the importance of the incorporation of verification into the generation process. Despite this, evidence base is still skewed in terms of reporting depth, metric consistent, and practical comparability. That is to say that verification has become recognised as a key focus of automated unit test generation research, but has not as yet reached maturity in terms of an equally mature methodology as the generation literature it is supposed to evaluate. The same tension is the reason why the following subsection, about assertion, repair and test suite improvement mechanisms, is needed: when we have the weaknesses revealed in verification, the improvement mechanisms are the response of the field to those weaknesses.

Assertion, Repair, and Test Suite Improvement Mechanisms:

In this subsection, post-generation mechanisms are looked at to enhance automatically generated unit tests once they have been initially generated. This category contains 8 studies that represent 16.7% of the 48-study corpus, which is smaller compared to the tools/frameworks category and the generation-focused category but plays a significant role in RQ2, RQ3 and RQ4. This category, as illustrated in Table 3, is an indication of the general

increasing awareness of the fact that first-pass test generation is not always adequate. Generated tests can be executable and yet have low oracle quality, scanty behavioral checking or brittle in realistic execution. Thus, assertion generation, oracle augmentation, amplification, repair, and refinement can be considered as the key components of automated unit test generation and not as its add-ons.

One of the most significant strands in this category is with regard to assertion generation and oracle enhancement. A3Test, ChIRAAG, AsserT5 and AugmenTest, are tools that are aimed at adding strength to the assertion layer of the generated tests instead of merely generating test inputs. This is significant since the usefulness of a developed test is not merely on whether it executes or not but on its capability of making differentiation between the right and wrong behavior. AsserT5 is of particular interest as it demonstrates that the quality of assertion can be considered a quantifiable research problem; nevertheless, the larger evidence base indicates that there is no guarantee that the higher the quality of assertion, the more direct correlation with the effective fault detection in the real world. This supports one of the key conclusions of the review: it is not always a good idea to consider executable or fluent tests to be reliable tests.

The second one is related to the amplification of tests and improvement of suites. Every system like Small-Amp and AugmenTest demonstrates that automated systems can enhance the existing or generated test suites and not merely generate tests. Such a pragmatic direction is pragmatic since even many existing software projects in the real world contain test suites which require extension, strengthening or adapting. Test amplification can be particularly advantageous where the reuse of existing tests can be used as a starting point in enhancing the behavioral coverage, assertion strength or fault-detection ability.

Test repair and healing is the third strand. Repair-oriented and adaptive test healing Repair-oriented approaches, such as adaptive test healing, is the extension of automated unit test generation into the maintenance phase, where failing or flaky automated unit tests are determined and repaired by feedback, or by learning-based methods. This matters as flaws in the creation of tests or present testing are usually identified in implementation, integration or subsequent development of code. Repair-oriented research is thus more conducive to a more realistic perspective of test automation: useful systems need not simply produce tests in the first place, but also contribute to stabilizing and sustaining them in the long run.

The co-evolutionary and mutation-guided refinement is another theme. Methods like the MuTAP and TestART indicate that a method of improvement can be integrated into an iterative loop and improvements to the test are informed by evaluation feedback, which is then used in subsequent test generation or repair. Verification and improvement are not isolated steps which are done at the end of the system in these systems; they are incorporated within the process of generation. It underlies the more general pipeline-based meaning that is worked out in the rest of Section 3.2 through 3.4, in which more robust generation of automated unit tests is based on the disciplined interplay of generation, validation, repair, and refinement.

Altogether, assertion, repair and test suite improvement mechanisms demonstrate the direction the field is taking to leave one-step test generation behind to more reliable and maintainable testing pipelines. This category would be numerically insignificant but analytically significant since it would rectify the drawbacks that verification and evaluation have revealed them to be weak oracles, unstable execution, incomplete suites, and limited fault revelation. This result also reflects on subsequent implications and recommendations, wherein, enhanced oracle support, repair mechanisms and multi-stage assessment are cited as major priorities in the future research in automated unit test generation studies.

Table 9. Assertion, repair, and test suite improvement mechanisms

Sr. #	References	Improvement Mechanism	Tool / Approach	Effect on Test Suite	Validation Method
1	[47]	Assertion generation	A3Test	Adds and strengthens assertions	Experimental Validation
2	[16]	Assertion generation	ChIRAAG	Generates assertions using LLMs	Empirical evaluation
3	[18]	Assertion generation	AsserT5	Generates test assertions	Benchmark evaluation
4	[50]	Oracle augmentation	AugmenTest	Enhances tests with LLM-driven oracles	Experimental Validation
5	[40]	Test amplification	Small-Amp	Expands existing test suites	Experimental Validation
6	[51]	Test repair and healing	Adaptive test healing	Repairs failing or flaky tests	Empirical evaluation
7	[14][4]	Co-evolutionary refinement	Mutation-guided refinement	Improves fault detection	Experimental Validation

Critically, there are also a number of limitations evident in the studies in this category. First, assertion and oracle enhancement are evidently getting better yet they are underrepresented in comparison to its significance. The evidence indicates that the problem of oracle quality remains among the most challenging unsolved ones in the field of automated generation of unit tests even with the emergence of more recent assertion-based tools. Second, repair and amplification studies have shown to be promising based on benchmark or even medium-scale tests, yet long-term maintainability and realistic adoption by developer's remains to be tested. Third, in reporting, the boarders of these mechanisms are at times blurred. The description of refinement, augmentation, and repair can overlap each other, and this fact makes it more difficult than usual to cross-study the research. These shortcomings should not be used to undermine the significance of this literature, but they do point to the fact that the work of improvement focus is still in its nascent phase as a research line.

In general, the studies chosen in this article demonstrate that assertion in the literature, repair, and test suite improvement mechanisms are one of the conceptually significant aspects even though they are numerically less than the rest of the categories. They are important as they reflect on the direct reaction of the field to the chronic maladaptations revealed by automated generation: feeble assertions, uncertain execution, incomplete suites, and little revelation of faults. The category does not in this sense just concern itself with post-processing. It is an indication of a larger change in the automated generation of unit tests towards generation of stronger, more stable and more defensible tests. That is the change that is among the most explicit signs that the field is coming of age, although it is by no means a methodologically stable one.

Answers to Research Questions:

This part will directly answer the four research questions out of the evidence that was synthesised out of the 48 primary studies that were selected. Unlike the previous version, the following answers are clearly and clearly matched with the revised structure of the results, and with the recovered category-based evidence on Sections 3.1–3.5. It is not just to generalise what the literature reports, but to engage critically that evidence in terms of techniques,

algorithms, tools and verification mechanisms, which combine to constitute the analytical core of this review.

RQ1: What Natural Language Processing (NLP) techniques have been applied for automated unit test generation?

The chosen papers demonstrate that the approaches to NLP-related techniques in automated unit test generation have recently become a wide range, spanning both traditional methods of language processing and the recently popular semantic generation process based on LLMs. In the 48-study corpus, prompt-based and transformer-based methods, such as zero-shot prompting, few-shot prompting, prompt engineering, and context-aware generation of LLMs based on source code, prompts, or a mixture of natural-language artefacts, dominate the recent literature. Such techniques are particularly evident in more recent work since they enable tests to be produced with more semantic fluency and context sensitivity as compared to many of the traditional structurally guided approaches. Nonetheless, it is also demonstrated that the application of LLM is hardly ever good enough. The more powerful ones are those that combine language-based generation with downstream control systems like slicing, refinement, mutation-directed feedback, or repair iteration.

Simultaneously, the review discovered that the automated unit test generation is not turned into an exclusive LLM-based. More classical NLP methods like requirements parsing, part-of-speech tagging, named entity recognition, retrieval-based support, and structured text interpretation are still in use or still are based on more classical methods where the artefacts of test are not read directly out of source code but are instead read out of requirements, user stories, or mixed natural-language specifications. This finding is significant as it demonstrates that the NLP addition to automated unit test generation is stratified, as opposed to flat. The approaches based on LLM have greatly extended the field, although they are founded on a more traditional practice of converting textual artefacts into more formal testing outputs. Consequently, the review recommends NLP in this area be conceptualized as not merely prompt-driven generation, but rather a broader set of language-processing systems which facilitate test derivation, assertion construction and contextual reasoning. The evidence presented in the summary of the evidence in Section 3.3, in particular, the technique-oriented overview and the target NLP mapping table, supports this interpretation.

More importantly, the studies reviewed also report that inconsistency of validation and strength of behavior is not merely the major weakness of current NLP-based approaches, but rather the lack of expressive power. Most of the language-based techniques generate tests that seem to make sense, are readable, or executable; however, the evidence is not as consistent when compared to oracle quality, mutation-based fault revelation and practical robustness. In this regard, the review concludes that the NLP-related techniques are now at the center of current automated unit test generation studies, although they remain in the immature state of relying heavily upon the effectiveness of the verification and refinement processes that enclose them.

RQ2: What methods do existing frameworks use to verify or optimize the quality of generated tests?

The review discovered that the current frameworks verify and optimize generated tests by a mix of mutation testing, coverage-based analysis, executability and pass-rate verification, empirical comparison, assertion-strengthening, and iterative repair or refinement processes. Of these, mutation testing is the most analytically meaningful as it determines the capability of created tests to reveal errors instead of just run program paths. A number of the most powerful recent works do not just use mutation as a metric of a post-generation, but as a directive mechanism to improve generated tests, a trend of wider adoption of verification-conscious pipelines as opposed to pipelines that are one-step generation systems.

The coverage-based evaluation is also prevalent throughout the chosen works, especially in the cases when a framework is built upon structural exploration, search-based generation, or comparison of the results of LLM benchmarked. The coverage of statements, branches and paths is still considered to be useful metrics of the extent of execution, yet the evidence examined shows conclusively that they are not adequate measures in and of themselves. One common trend in the corpus is that the frameworks can attain promising coverage improvements and at the same time, have a poorer performance during mutation-oriented testing or oracle quality. This helps to substantiate one of the main conclusions of the review: the literature has now improved in producing tests, as opposed to proving that the produced tests are behaviorally strong. This imbalance also can be seen on the level of categories since tools and frameworks significantly outnumber the studies that are specifically based on verification and evaluation.

The review also discovered that there is an increasing tendency of optimization by post-generation mechanisms as opposed to generation. Frameworks like mutation-guided refinement pipelines, assertion-augmentation tools and repair-oriented systems enhance the generated tests by refining assertions, repairing failures or instead feeding evaluation feedback to subsequent iterations. This is a methodological factor as it demonstrates that quality optimization is no longer considered as a last evaluation phase. Instead it is also becoming a part of the generation process itself. Critically, though, there are still no uniform evaluation standards of frameworks in the literature. The variety of benchmarks, target languages, prompt design, model versions, and reported metrics makes comparisons of optimization claims between studies challenging. The answer to RQ2, then, is that present frameworks do validate and optimize generated tests with an ever-increasing variety of more and more sophisticated mechanisms, but the general maturity of that verification ecosystem is still uneven.

RQ3: What tools have been developed and/or used for automated unit test generation, and what inputs and outputs do these tools support?

The review established that tools and frameworks constitute the biggest proportion of evidence base, indicating 27 of the 48 studies that were chosen (56.3%), which substantiates that the discipline has a solid implementation-driven nature. The selected articles describe a great variety of tools, both traditional and language-specific like MATTER, Nessie, SynTest-JavaScript, GuessWhat, and PC-TRT, and newer tools powered by LLM, as well as hybrid ones, such as ChatUniTest, TestART, MuTAP, A3Test, ChIRAAG, AsserT5, AugmenTest, Small-Amp and Adaptive Test Such a variety demonstrates that a unit test can be generated automatically regardless of the kind of platform or input source. The tools literature, instead, represent a discipline that is expanding in terms of both technical and operational aspects.

The inputs that these tools can support and across the reviewed corpus are source code, UML or model artefacts, natural-language prompts, natural-language specifications, APIs, and existing test suites, feedback signals, execution traces, and mutation information. Their products are also very varied and even far much more than conventional unit tests. The output can be in the form of unit test suites, assertions, amplified test suites, fixed or mended tests, mutants, stub code or any other artefacts of the test depending on the tool. It is one of the most obvious indications that the field is becoming multi-stage pipelines: tools are no longer just providing raw tests, but more and more are providing subsequent stages of refinement, validation, and repair or oracle enhancement.

Simultaneously, the review established that such a wide range of tools available is not to be taken as a sign of a lack of methodological maturity per default. The tools are non-homogeneous in terms of target language, context modelling, and validation strategy and the level of realistic practicality. Others are mostly benchmark-based, others are narrowly focused on specific artefacts or languages and others rely extensively on prompt engineering, mutation feedback or filtering to get results acceptable enough. The most dominant overarching trend

is then not the fact that there are an increasing number of tools available but that helpful tools are increasingly becoming more hybrid, verification-aware and context-based. Critically, this translates to the fact that the proliferation of tools in itself is not the primary success measure. Whether these tools generate artefacts that can be executed, are fault exposing, maintainable and compatible with realistic developer processes is the more important question.

RQ4: What algorithms have been developed and/or applied in automated unit test generation, and how do they transform system inputs into test outputs?

The chosen papers demonstrate that automated generation of unit tests currently is based on a wide variety of algorithms that convert the inputs into outputs in dramatically different ways of reasoning. Classical methods remain based on search based, evolutionary, model-based, feedback-directed methods and probabilistic inference to investigate input spaces, make inferences about executable values and optimize structural adequacy. These algorithms are still significant as they allow explicit control of the exploration, and can have more predictable execution behavior in a constrained environment. They are particularly applicable to situations where test generation relies on type inference, path exploration, or systematic optimization of code coverage and other measures of adequacy it implies.

More recent works are based on more specialized algorithms that use LLM-based algorithms, or hybrid algorithms, that work by prompt-driven generation, contextual inference, iterative prompting, coverage-aware feedback, mutation-directed refinement, retrieval support, and repair loops. In such methods, there is not necessarily a direct relationship between the input and the test output of the system. Some of the studies add in between algorithm steps like method slicing, related artefact retrieval, prompt enrichment, refinement based on feedback or assertion augmentation, and then the final output. It is an important discovery since it implies that the most powerful new algorithms are not necessarily bigger models with the outdated algorithms being downgraded. Instead, they are designed pipelines which interchange between input artefacts and test outputs based on several stages of transformation.

The review thus concludes that the present-day development of algorithms is not as substitution as hybridization oriented. The traditional algorithms continue to provide strength in the control, search discipline, and structural adequacy whereas the LLM based algorithms provide semantic flexibility and extended contextual reasoning. Recent algorithms that combine all these strengths and not individual ones are the most promising. This conclusion is noteworthy as it specifically focuses on the direction that automated unit test generation is currently taking: the generation is not most convincingly being made in pure model-driven generation, but rather in workflows where the generation is assisted by slicing, mutation feedback, repair iteration, assertion support, or execution-based filtering. Critically, however, there is a comparability issue in the literature since there is a wide variance in the benchmark conditions, language targets, and evaluation methods that are used in algorithmic studies. The response to RQ4 is thus that the field has a wide and ever more hybrid collection of algorithms, but they are much as strong in terms of their ability to organize the transformation of input artefacts into executable and behaviorally useful test outputs.

Overall synthesis of the four research questions:

A combination of the responses to the four research questions would give a consistent conclusion. The generation of automated unit tests has made significant progress, particularly with the emergence of LLM-based and hybrid approaches, but the definition of progress in the field has evolved, as well. There is no longer a need to pose the question of whether it is possible to create tests automatically. What is more crucial is whether they can be created in a context-sensitive, executable, fault-revealing, maintainable and practically trustworthy way. The evidence base of 48 studies demonstrates that there is a definite direction towards that aim, though, the current evidence base indicates an imbalance in that generation methods and

tools are evolving more rapidly than standards of validation, oracle quality, and cross-study comparability. This is why the finest modern strategies are becoming more and more those ones which are integrative, combining generation with verification, refinement and repair as opposed to considering them distinct issues.

Discussion:

This review reveals that automated unit test generation can no longer be viewed as a problem with a limited focus on code-generation. Rather, the data of the 48 chosen articles show a tendency towards a larger quality-engineering paradigm where test generation, verification, refinement, assertion support and repair are becoming more intertwined with each other. This interpretation is in line with the conceptual pipeline presented in Figure 1 and results structure depicted in Table 4. Previous research on automated testing tended to use the generation of executable tests or the coverage of tests as its primary measure of progress, however the synthesis in Sections 3.13.5 indicates that usefulness in automated unit test generation is not limited to the generation of tests, but also to the executability, fault-revealing, maintainability and coverage by meaningful oracles of those tests.

The results on the bias in the generation-oriented research and validation-oriented research are one of the most evident. As illustrated in Table 3, the tools and frameworks constitute the biggest section of the review, with 27 out of 48 out of the 48 studies selected (56.3%), with only 10 studies (20.8) studies involving verification and evaluation. This spread is an indication that the field has been moving faster in the development of systems to produce tests rather than coming up with equally mature means of establishing the quality of such tests. This result directly helps to justify the issue covered by the RQ2, as coverage, executability, and pass rate are valuable metrics, but they cannot be used independently. A generated test can work correctly to compile, and to run successfully, but still fail to find faults in it in the event that its assertions are weak or in the event that they merely verify the existing behavior, not the intended behavior.

This issue has been brought into the limelight by the emergence of LLM-based test generation which is primarily addressed in RQ1 and RQ4. LLMs have enhanced the capability of automated systems to produce context-sensitive and readable test code based on the source code, prompts and other artefacts of natural language. However, as the evidence reviewed demonstrates, the generation based on LLM does not necessarily resolve such long-standing issues as the quality of oracles, their dependency management, and the reproducibility, as well as the comparability of benchmarking. Recent research on mutation-guided prompting, assertion generation, and test improvement with Industrial LLMs demonstrate that the best results are obtained with generation and validation, filtering, repair, or refinement. This implies that the shift towards controlled and verification-conscious pipelines is not merely the most crucial recent change, but the use of the LLMs.

It was also revealed during the review that the traditional and the approaches based on the use of LLMs cannot be considered mutually exclusive. This aspect is based on the synthesis algorithm in Section 3.3 and on the direct response to RQ4. The search-based, model-based, probabilistic and structurally guided methods still have advantages in exploring the paths, reproducibility and control of the execution. In comparison, the approaches based on LLM offer more contextual generation and semantic flexibility. The most promising direction seems to be the hybridization, that is, structural techniques are integrated with the LLM-based reasoning and are used in multi stage workflows. These workflows can be used to generate, check the execution, provide mutation feedback, refine the assertions, and repair them in one pipeline.

The other issue of discussion is the on-going oracle problem. The problem is reflected in the critical literature review, the verification results in Section 3.4 and in the assertion/improvement mechanisms in Section 3.5. According to a number of recent studies,

there has been a positive development in assertion generation and oracle augmentation, but there is no doubt that this is one of the weakest areas of automated unit test generation. This is important since the usefulness of a generated test at practice will largely depend on its ability to identify misbehavior rather than simply on whether or not it can run code. The assertion-based testing and mutation testing thus give more evidence of the quality of the test as compared to coverage. Future automated unit test generation studies ought to consider oracle quality as an evaluation aspect in and of itself and not as a secondary enhancement measure.

The results also show that the industrial maturity is still low and this directly relates with the implications as discussed in the next sub section. Experience in industry, including the use of test generation (LLM-aided test improvement) in real development scenarios, demonstrates that automated test generation can be of practical use. Nevertheless, this value is based on the strict filtering, reviewing by developers, reliability of the execution and maintainability. So, automated unit test generation cannot be regarded as a completely independent and autonomous substitute of developer judgement yet. Rather it can be thought of as an aide that can alleviate manual effort when incorporated into engineered engineering processes.

The discussion, in general, advocates a critical yet positive view of the field. Generation of unit tests has achieved apparent advancements with the help of LLM-enabled and hybrid systems. But, according to the responses to RQ1-RQ4, the field is not yet balanced in the strength of validation, oracle quality, benchmark standardization, and even the industrial adoption. The most justifiable conclusion is that the future development will rely less on the generation ability and more on the unification of the generation with verification, refinement and repair and realistic testing. This pipeline-based approach offers the best root in making automated unit test generation more reliable and practical to real software-engineering.

Study Implications:

The results of this review have significant implications on research, practice and development of tools. On a conceptual level, the results of the 48 studies selected indicate that automated unit test generation can no longer be conceptualized as a one-step task of code-generation. Rather, as indicated by the pipeline-based results in the Sections 3.2-3.5 it must be interpreted as a quality-engineering process whereby generation, verification, repair and oracle enhancement are interrelated. This is significant since coverage, executability and output fluency cannot be enough to have reliable test quality.

The second implication is related to standards of evaluation. The review indicates that the field still heavily depends on structural measures, like coverage, whereas RQ2 and Section 3.4 reveal that coverage is just a partial measure of practical effectiveness. A more rigorous test ought to be a combination of several tests, such as mutation score, fault-revealing ability, oracle adequacy, executability, pass rate, and post-generation reliability. Thus, the studies of the future must go beyond the question concerning the possibility of the generation of a test and examine the possibility of the generated test to detect faults in realistic conditions in a meaningful way.

Tool and framework developers can also have implications of the findings. The best recent systems are not based on one-shot generation as demonstrated in Sections 3.2 and 3.5. They are a combination of immediate design, control of the context, validation of execution, filtering, repair, assertion strengthening or mutation-directed refinement. This indicates that further development of tools needs to focus on end-to-end workflow robustness, as opposed to the ability to generate standalone. Using more powerful orchestration of pipeline stages, rather than larger or newer models, is likely to provide better automated unit testing tools.

Industry wise, the review is an opportunity and a warning. The improvement of tests facilitated by the use of LLM can offer quantifiable value, although not until the results of the generated outputs are filtered, validated, reviewed, and subsequently incorporated into the

controlled engineering processes. Hence, the generated tests are to be viewed as the ones to be improved instead of presumed to be trustworthy results. This is particularly significant in real projects where the value of seemingly effective test generation can be diminished by weak assertions, flaky behavior, dependency issues and maintainability costs.

Last but not least, the review supports the ongoing relevance of mixed solutions and increased accountability of reporting. Conventional search-based, model-based, and structure-based approaches are useful due to their ability to have control, reproducibility, and systematic exploration, whereas the use of LLM-based approaches adds semantic freedom and contextual reasoning. The next step toward advancement is thus expected to be the integration of these strengths and reporting studies in a more transparent manner by providing prompts, model configurations, datasets, filtering logic and evaluation conditions. This would enhance reproducibility, cross study comparability as well as cumulative value of future studies.

Implications of the Study:

Even though this review had a well-designed SLR design and maintained well-mapped evidence base of 48 primary studies, a number of limitations must be considered. To begin with, only those studies which were found in the chosen workflow of the digital-library and the ultimate inclusion criteria explained in Section 2.1.1 were reviewed. This increased focus and traceability, and might have missed any related preprints, technical reports, industrial white papers or very recent studies that were not indexed at the moment of search. As such, it can be assumed that the review is a stringent synthesis of a specified body of evidence, not an attempt to enumerate all the literature on automated unit test generation.

Second, there is a great difference in programming language, choice of benchmark, version of the model, prompt design, evaluation metric, and reporting depth of the studies included. This heterogeneity is both an expression of the field breadth, and a constraint on direct comparability. The coverage, mutation score, assertion accuracy, pass rate or executability are often reported in studies under various experimental conditions. It is due to this reason that the review took a structured method of qualitative synthesis and descriptive aggregation as opposed to an actual statistical meta-analysis. This weakness can be aligned with the issue of the field level that was found during the review: the generation of automated unit tests is still innovative yet not even in terms of evaluation standardization.

Third, the quality of reporting and reproducibility of the original studies influences the review. Papers vary in their description of prompts, datasets, tool settings, benchmark construction and evaluation procedures. The quality assessment was used to define the strength of the evidence, however, incomplete reporting of some of the primary studies makes it difficult to replicate or fairly compare the findings of these studies.

A fourth weakness is with regard to the dynamism of the research in the field of software testing on the basis of LLM. New models, which require strategies, toolchains, and evaluation practices, are quickly changing, and it means that even the latest reviews will be outdated soon. Thus, the findings of this review are to be interpreted as a synthesis of the field during a specific review time frame, and not necessarily as a final or definite report on automated unit test generation.

These limitations should be overcome in future work in a number of ways. To begin with, the discipline should have additional standardized benchmarking and assessment frameworks which integrate coverage, mutation testing, executability, oracle quality and maintainability in practice. Second, more focus needs to be on the generation and quality of oracles, which are weak ones as a consistent limitation is observed in the reviewed literature. Third, the research in the future should come up with more powerful hybrid pipelines combining the processes of generation, verification, repair and refinement. Fourth, deployment-oriented and longitudinal studies are required to investigate whether or not generated tests are maintained, edited, trusted, fixed or abandoned in the actual processes of

software-engineering. Lastly, evidence base should be periodically updated in future SLRs since the tools and evaluation criteria based on LLM are rapidly changing.

Limitations and Future Work:

Although this review adopted a structured SLR design and retained a clearly mapped evidence base of 48 primary studies, several limitations should be acknowledged. First, the review was limited to studies retrieved through the selected digital-library workflow and the final inclusion criteria described in Section 2.1.1. This strengthened focus and traceability, but it may have excluded relevant preprints, technical reports, industrial white papers, or very recent studies not fully indexed at the time of searching. Therefore, the review should be interpreted as a rigorous synthesis of a defined evidence set rather than a complete census of all work on automated unit test generation.

Second, the included studies vary substantially in programming language, benchmark choice, model version, prompt design, evaluation metric, and reporting depth. This heterogeneity reflects the breadth of the field, but it also limits direct comparability. Studies reporting coverage, mutation score, assertion accuracy, pass rate, or executability often do so under different experimental conditions. For this reason, the review adopted structured qualitative synthesis and descriptive aggregation rather than a strict statistical meta-analysis. This limitation is consistent with the field-level problem identified throughout the review: automated unit test generation remains innovative but uneven in evaluation standardization.

Third, the review is affected by the reporting quality and reproducibility of the primary studies themselves. Not all papers describe prompts, datasets, tool settings, benchmark construction, or evaluation procedures with the same level of detail. Although quality assessment was used to interpret the strength of the evidence, incomplete reporting in some primary studies limits the extent to which their findings can be replicated or compared fairly.

A fourth limitation concerns the fast-changing nature of LLM-based software testing research. New models, prompting strategies, toolchains, and evaluation practices are emerging rapidly, which means that even recent reviews can become outdated quickly. Therefore, the conclusions of this review should be understood as a synthesis of the field within a defined review period, rather than as a final or fixed account of automated unit test generation.

Future work should address these limitations in several directions. First, the field needs more standardized benchmarking and evaluation frameworks that combine coverage, mutation testing, executability, oracle quality, and practical maintainability. Second, greater attention should be given to test oracle generation and assertion quality, as weak oracles remain a persistent limitation across the reviewed studies. Third, future research should develop stronger hybrid pipelines that integrate generation, verification, repair, and refinement. Fourth, more deployment-oriented and longitudinal studies are needed to examine whether generated tests are retained, modified, trusted, repaired, or discarded in real software-engineering workflows. Finally, future SLRs should periodically update the evidence base because LLM-based tools and evaluation standards continue to evolve rapidly.

Recommendations:

On the basis of the results of this review, a number of suggestions can be offered to researchers, developers of tools and practitioners. First, there should be a shift in the future work whereby the quality of the pipeline should be emphasized rather than the volume of generation. As depicted in Sections 3.23.5, the most recent systems are not only stronger as they generate more tests, but they are stronger when they are coupled with filtering as they generate and mutate the tests, assertion support, repair and validation. Thus, the contribution to be made in future is to be judged by the ability to create tests that are executable, fault-revealing, maintainable and realistic to be used.

Second, scholars ought to enhance the evaluation discipline. Coverage needs to be included in the evaluation, but not regarded as an adequate measure of a practical test of

quality. More detailed and balanced evaluation designs, which integrate coverage and mutation testing, oracle adequacy, executability, pass rate and, where feasible, maintenance-oriented or user-oriented results, should be used in future research. This would enhance comparability of studies and enhance future secondary review of studies.

Third, the quality of test oracle generation and assertion must be given a higher priority in the future in research and tools development. It is repeatedly demonstrated in the review that weak assertions and superficial oracles continue to be significant limitations, particularly in the case of approaches based on LLM. A3Test, ChIRAAG, AsserT5 and AugmenTest are steps in the right direction, but they indicate that oracle quality is yet to be fully achieved. Systems of the future must consider assertion and oracle support as design goals and not as a post processing option.

Fourth, developers of tools need to plan a controlled adoption and not an uncontrollable automation. Industrial experience indicates that automated unit test generation can offer value, albeit with output filtering, reviewing, and validation and integration into disciplined engineering processes. The next generation tools should thus incorporate the aspects of trust building like validation of the executions, dependency management, support of repair, explainable filtering, and compatibility with the current development pipelines.

Fifth, hybrid methods should be preferred in the future work over exclusive use of methods. The studies reviewed do not follow a straightforward replacement story whereby, the search-based, model-based, or structurally guided techniques are replaced by the LLM-based systems. Rather, the traditional methods bring about the element of control, reproducibility, and systematic exploration whereas the LLM-based methods bring the element of semantic flexibility and contextual reasoning. By integrating these strengths, there are likely to be more trusted automated testing pipelines.

Lastly, longitudinal and deployment-based research should be made more in the future. There are a number of studies that are still benchmark-based and this is handy in the initial assessment but inadequate in determining long term practical value. Further research is required to investigate the retention, modifications, trust, mending or discarding of generated tests, in actual software-engineering processes. This would assist the discipline to leave the technical possibility to the long-term adoption and reliable quality-engineering practice.

Conclusion:

The current literature review has identified the state of the art in automated unit test generation by searching and reviewing 48 primary studies: 48 studies in total were identified, covering the 4 interconnected aspects of techniques, algorithms, tools and verification mechanisms. This review has indicated that over the past few years, the field has developed significantly, especially with the emergence of approaches based on LLM and hybrid models, although the development is not even. The most evident strength of the literature is the increasing number of tools and frameworks, taking the largest portion of the selected studies (27/48; 56.3%), next in line are studies devoted to test generation techniques and algorithms (14/48; 29.2%) followed by the aspects of verification and evaluation (10/48; 20.8%) and assertion, repair, and improvement mechanisms (8/48; 16.7%). The importance of these distributions is that they demonstrate that automated unit test generation is becoming more of an implementation-focused issue, but remains relatively immature in terms of validation and quality assurance of the generated test.

The review also discovered the field is ceasing to be developed primarily by isolated generation strategies. Rather, increasingly popular recent studies are using hybrid and pipeline-based workflows, which incorporate generation and refinement, mutation-directed feedback, assertion refinement, or repair. This is among the conclusions of the review that are the most important. It indicates that generation is not the way forward in the future of automated unit test generation, but generation in the combination with mechanisms to enhance behavioural

reliability and practical usefulness. In this regard, recent work based on LLM has certainly provided the semantic and contextual power of automated testing, but has not eliminated the necessity of structured validation or solved long-standing oracle quality weaknesses.

One of the main findings of this overview is that the lack of balance between the performance of the generation, and the power of quality assurance remains a current issue in the field. The evidence is becoming more and more reported, executable, and readable, but is less convincing when compared to more robust evidence in the form of mutation-based fault revelation, assertion adequacy and practical reliability. This implies that automated unit test generation is now more technically feasible, but not quite as reliable. The review thus recommends a less optimistic view of the present developments: the literature shows that there is a definite improvement, but that the improvement is best where generation is coupled with evaluation and improvement as opposed to being an end in itself.

Directly related to the research questions, the review concluded that NLP and LLM methods currently are at the center of test generation, but do not exclude the older search-based, model-based and probabilistic methods, instead complementing them. Current frameworks check and optimize generated tests primarily with the help of mutation testing, coverage-based testing, assertion support, repair, and refinement processes, although the evaluation is unexplored across studies. Various tools and frameworks have now been created, which can take a wide variety of inputs (source code, models, prompts and existing tests), and can give a wide variety of output (unit test suites, assertions, repaired tests, and amplified suites). Lastly, the algorithmic space is becoming more hybrid and the trends are currently based on prompt engineering, context filtering, slicing, feedback loops and refinement strategies and no longer on a one-step transformation of input to output.

When combined with the above findings, this suggests that automated unit test generation is a rapidly changing, but not fully mature, field. It is shifting away towards more general automation and quality engineering, away towards isolated techniques and pipelines, and more towards a more semantically aware and context sensitive approach to generation. It is however bound by reproducibility, standardization of evaluation, strength of oracle and realism of industry. That is why it is not that the problem has been solved, but rather the field has already reached the point where its further development will be conditioned by the success with which it will manage to combine the generation ability with the verification discipline, the post-generation improvement and the practical needs of its deployment.

This review as a whole can be seen to have value in maintaining the structure of evidence of the chosen 48 studies and provide a critical synthesis of how the present-day work is transforming the field of automated unit test generation. Clearly, the field is more than ever ready to advance, but the successive advances must be gauged not by the number of tests it can generate, but by their significance, their ability to reveal faults, and their longevity and reliability in actual software-engineering situations. This is the overall finding of the current review.

Acknowledgement:

The authors also acknowledge the use of Artificial Intelligence (AI) tools for language refinement and proofreading to improve the clarity and readability of the manuscript. All research design, analysis, and conclusions presented in this paper were developed and validated by the authors.

References:

[1] P. McMinn, "Search-based software test data generation: A survey," *Softw. Test. Verif. Reliab.*, vol. 14, no. 2, pp. 105–156, Jun. 2004, doi: 10.1002/STVR.294;ISSUE:ISSUE:DOI.

[2] S. M. Bindu Bhargavi and V. Suma, "A Survey of the Software Test Methods and Identification of Critical Success Factors for Automation," *SN Comput. Sci. 2022 36*,

vol. 3, no. 6, pp. 449-, Aug. 2022, doi: 10.1007/s42979-022-01297-5.

[3] J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, and Q. Wang, "Software Testing With Large Language Models: Survey, Landscape, and Vision," *IEEE Trans. Softw. Eng.*, vol. 50, no. 4, pp. 911–936, Apr. 2024, doi: 10.1109/TSE.2024.3368208.

[4] Murat Tasarsu, Ahmet Vedat Tokmak & Cagatay Catal, "Test case generation using large language models: a systematic literature review," *Cluster Comput.*, vol. 29, no. 227, 2026, [Online]. Available: https://link.springer.com/article/10.1007/s10586-026-06021-z

[5] Samar Ali Abdallah, Ramadan Moawad, "An optimization approach for automated unit test generation tools using multi-objective evolutionary algorithms," *Futur. Comput. Informatics J.*, vol. 3, no. 2, pp. 178–190, 2018, doi: https://doi.org/10.1016/j.fcij.2018.02.004.

[6] Ohood Al-Masri, Wedad Abdulraqeeb Al-Sorori, "Object-Oriented Test Case Generation Using Teaching Learning-Based Optimization (TLBO) Algorithm," *IEEE Access*, 2022, doi: 10.1109/ACCESS.2022.3214841.

[7] I. Hajri, A. Goknil, F. Pastore, and L. C. Briand, "Automating system test case classification and prioritization for use case-driven testing in product lines," *Empir. Softw. Eng.*, vol. 25, no. 5, pp. 3711–3769, Sep. 2020, doi: 10.1007/s10664-020-09853-4.

[8] Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi D.Q. Bui, Junnan Li, Steven C.H. Hoi, "CodeT5+: Open Code Large Language Models for Code Understanding and Generation," *arXiv:2305.07922*, 2023, [Online]. Available: https://arxiv.org/abs/2305.07922

[9] G. Yi, Z. Chen, Z. Chen, W. E. Wong, and N. Chau, "Exploring the Capability of ChatGPT in Test Generation," *Proc. - 2023 IEEE 23rd Int. Conf. Softw. Qual. Reliab. Secur. Companion, QRS-C 2023*, pp. 72–80, 2023, doi: 10.1109/QRS-C60940.2023.00013.

[10] S. Zekarias Esubalew and B. G. Assefa, "Reimagining Unit Test Generation With AI: A Journey From Evolutionary Models to Transformers," *IEEE Access*, vol. 13, pp. 154908–154929, 2025, doi: 10.1109/ACCESS.2025.3597049.

[11] Yutian Tang, Zhijie Liu, Zhichao Zhou, Xiapu Luo, "ChatGPT vs SBST: A Comparative Assessment of Unit Test Suite Generation," *arXiv:2307.00588*, 2023, [Online]. Available: https://arxiv.org/abs/2307.00588

[12] "Hybrid Concolic Testing with Large Language Models for Guided Path Exploration." Accessed: May 05, 2026. [Online]. Available: https://arxiv.org/html/2601.12274v1

[13] Nadia Alshahwan, Jubin Chheda, Anastasia Finegenova, Beliz Gokkaya, "Automated Unit Test Improvement using Large Language Models at Meta," *arXiv:2402.09171*, 2024, [Online]. Available: https://arxiv.org/abs/2402.09171

[14] Arghavan Moradi Dakhel, Amin Nikanjam, "Effective test generation using pre-trained Large Language Models and mutation testing," *Inf. Softw. Technol.*, p. 107468, 2024, doi: https://doi.org/10.1016/j.infsof.2024.107468.

[15] Hao Yu, Yiling Lou, "Automated assertion generation via information retrieval and its integration with deep learning," *Proc. - Int. Conf. Softw. Eng.*, 2022, [Online]. Available: https://dl.acm.org/doi/10.1145/3510003.3510149

[16] Bhabesh Mali, Karthik Maddala, Vatsal Gupta, Sweeya Reddy, Chandan Karfa, Ramesh Karri, "ChIRAAG: ChatGPT Informed Rapid and Automated Assertion Generation," *arXiv:2402.00093*, 2024, [Online]. Available: https://arxiv.org/abs/2402.00093

[17] Masoumeh Taromirad & Per Runeson, "Assertions in software testing: survey,

landscape, and trends," *Int. J. Softw. Tools Technol. Transf.*, vol. 27, 2025, [Online]. Available: https://link.springer.com/article/10.1007/s10009-025-00794-1

[18] Severin Primbs, Benedikt Fein, "AsserT5: Test Assertion Generation Using a Fine-Tuned Code Language Model," *arXiv:2502.02708v1*, 2025, [Online]. Available: https://arxiv.org/html/2502.02708v1

[19] Adam Bodicoat, Gunel Jahangirova, Valerio Terragni, "Understanding LLM-Driven Test Oracle Generation," *arXiv:2601.05542*, 2026, [Online]. Available: https://arxiv.org/abs/2601.05542

[20] Zhonghao Guo, Sinong Chen, "PC-TRT: A Test Case Reuse and generation Tool to achieve high path coverage for Unit Test," *SoftwareX*, vol. 28, p. 101918, 2024, doi: https://doi.org/10.1016/j.softx.2024.101918.

[21] "Mutation Testing Advances: An Analysis and Survey | Request PDF." Accessed: May 05, 2026. [Online]. Available: https://www.researchgate.net/publication/325014033_Mutation_Testing_Advances_An_Analysis_and_Survey

[22] Jordy Navarro, Ronald Ibarra, "Automatic test case generation using natural language processing: A systematic mapping study," *Inf. Softw. Technol.*, vol. 189, p. 107929, 2026, doi: https://doi.org/10.1016/j.infsof.2025.107929.

[23] Léuson Da Silva, Paulo Borba, "Detecting semantic conflicts with unit tests," *J. Syst. Softw.*, vol. 214, p. 112070, 2024, doi: https://doi.org/10.1016/j.jss.2024.112070.

[24] Frank Tip, Jonathan Bell, Max Schaefer, "LLMorpheus: Mutation Testing using Large Language Models," *arXiv:2404.09952*, 2025, [Online]. Available: https://arxiv.org/abs/2404.09952

[25] Lin Yang, Chen Yang, "On the Evaluation of Large Language Models in Unit Test Generation," *Proc. - 2024 39th ACM/IEEE Int. Conf. Autom. Softw. Eng. ASE 2024*, 2024, [Online]. Available: https://dl.acm.org/doi/10.1145/3691620.3695529

[26] Bernhard K. Aichernig & Klaus Havelund, "AI-Assisted Programming with Test-Based Refinement," *Bridg. Gap Between AI Real.*, pp. 385–411, 2024, [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-031-73741-1_24

[27] Khashayar Etemadi, Marjan Sirjani, Mahshid Helali Moghadam, Per Strandberg, Paul Pettersson, "LLM-based Property-based Test Generation for Guardrailing Cyber-Physical Systems," *arXiv:2505.23549*, vol. 6, 2025, [Online]. Available: https://arxiv.org/abs/2505.23549

[28] M. J. Page *et al.*, "The PRISMA 2020 statement: An updated guideline for reporting systematic reviews," *BMJ*, vol. 372, Mar. 2021, doi: 10.1136/BMJ.N71.

[29] Melissa L. Rethlefsen, Shona Kirtley, Siw Waffenschmidt, Ana Patricia Ayala, "PRISMA-S: an extension to the PRISMA Statement for Reporting Literature Searches in Systematic Reviews," *Syst. Rev.*, vol. 10, no. 39, 2021, [Online]. Available: https://link.springer.com/article/10.1186/s13643-020-01542-z

[30] B. Kitchenham, L. Madeyski and D. Budgen, "SEGRESS: Software Engineering Guidelines for REporting Secondary Studies," *IEEE Trans. Softw. Eng.*, vol. 49, no. 3, pp. 1273–1298, 2023, doi: 10.1109/TSE.2022.3174092.

[31] "IEEE Xplore." Accessed: Mar. 17, 2026. [Online]. Available: https://ieeexplore.ieee.org/Xplore/home.jsp

[32] "ACM Digital Library." Accessed: Mar. 17, 2026. [Online]. Available: https://dl.acm.org/

[33] "ScienceDirect." Accessed: Jul. 14, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050921014599/pdf?crasolve=1&r=8a3770a69f8a22b6&ts=1721022260273&rtype=https&vrr=UKN&redir=UKN&redir_fr=UKN&redir_arc=UKN&vhash=UKN&host=d3d3LnNjaWVuY2Vka

XJlY3QuY29t&tsoh=d3d3LnNjaWVuY2VkaXJlY3QuY29t&rh=d3d3LnNjaWVuY2VkaXJlY3QuY29t&re=X2JsYW5rXw%3D%3D&ns_h=d3d3LnNjaWVuY2VkaXJlY3QuY29t&ns_e=X2JsYW5rXw%3D%3D&rh_fd=rrr)n%5Ed%60i%5E%60_dm%60%5Eo)%5Ejh&tsoh_fd=rrr)n%5Ed%60i%5E%60_dm%60%5Eo)%5Ejh&iv=08f224d777421a845023c0e7a0edff9f&token=6161326237393330613833616637643431633138346232306336393163653062303334383064383136616135396562363933383239386463613636323930626230633366633931353030666666656238356436363937393431626331343337656232653736353830396336373764383032336432336433306633613861386133383836363532636433623436383833306466393630613637&text=fba922b59a9bed370419e04a1d372945763243e91f8b493c67da9d3c79f7496c4ed77b3ef8c7eeaf4d5c9799d135643e3b7865bd95934282a43b7bae8ca575f5b31e8477cc66c8a8740d97437a3c8fb1fe2d816e372e315c864a995be80e6c5d24b4c2e3c21b7568b4df92d1581edc94f1b5c43125875ad047acef9b23846b20becd6aa3601d24cd451cf7c3eaa8b6cd37e48effd8175f23f440c91ef3ec5b989a5772eb991aa48c28287a9145f20ebb9ce51591627ed3e36002c0419e05bc83d28407c6a81274136401abaf84c36f91de76876cefdf49640d0f6a22a98f63ed979ba82f78ccabf03353f1599d11051aaee6f21a29b48dc102822504447a5bcef0ad59a5bbf0717f8f0b608cabbcfba290c693536799cd65770b4e42b498df263a8ff44638fc09af6f348df0ccd65323&original=3f6d64353d353131303132376134356431313937303965303266386563343764336136333526706964 3d312d73322e302d5331383737303530393231303134353939 2d6d61696e2e706466

[34] "Home | Springer Nature Link." Accessed: Mar. 17, 2026. [Online]. Available: https://link.springer.com/

[35] Vahid Garousi, Sara Bauer, "NLP-assisted software testing: A systematic mapping of the literature," *Inf. Softw. Technol.*, vol. 126, p. 106321, 2020, doi: https://doi.org/10.1016/j.infsof.2020.106321.

[36] Vahid Garousi, Michael Felderer, "Guidelines for including grey literature and conducting multivocal literature reviews in software engineering," *Inf. Softw. Technol.*, vol. 106, pp. 101–121, 2019, doi: https://doi.org/10.1016/j.infsof.2018.09.006.

[37] Muhammad Usman, Nauman bin Ali, Claes Wohlin, "A Quality Assessment Instrument for Systematic Literature Reviews in Software Engineerin," *arXiv:2109.10134*, 2021, [Online]. Available: https://arxiv.org/abs/2109.10134

[38] Yinghao Chen, Zehao Hu, "ChatUniTest: A Framework for LLM-Based Test Generation," *FSE Companion - Companion Proc. 32nd ACM Int. Conf. Found. Softw. Eng.*, 2024, [Online]. Available: https://dl.acm.org/doi/10.1145/3663529.3663801

[39] Dario Olianas, Maurizio Leotta & Filippo Ricca, "MATTER: A tool for generating end-to-end IoT test scripts," *Softw. Qual. J.*, vol. 30, 2022, [Online]. Available: https://link.springer.com/article/10.1007/s11219-021-09565-y

[40] Mehrdad Abdi, Henrique Rocha, Serge Demeyer & Alexandre Bergel, "Small-Amp: Test amplification in a dynamically typed language," *Empir. Softw. Eng.*, vol. 27, 2022, [Online]. Available: https://link.springer.com/article/10.1007/s10664-022-10169-8

[41] Pekka Abrahamsson, Tatu Anttila, Jyri Hakala, Juulia Ketola, Anna Knappe, Daniel Lahtinen, Väinö Liukko, Timo Poranen, "ChatGPT as a Fullstack Web Developer - Early Results," *Agil. Process. Softw. Eng. Extrem. Program. – Work.*, pp. 201–109, 2023, [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-031-48550-3_20

[42] Ciprian Paduraru, Adelina Staicu, "LLM-based methods for the creation of unit tests in game development," *Procedia Comput. Sci.*, vol. 246, pp. 2459–2468, 2024, doi: https://doi.org/10.1016/j.procs.2024.09.473.

[43] Siqi Gu, Quanjun Zhang, Kecheng Li, Chunrong Fang, Fangyuan Tian, Liuchuan Zhu, Jianyi Zhou, Zhenyu Chen, "TestART: Improving LLM-based Unit Testing via

Co-evolution of Automated Generation and Repair Iteration," *arXiv:2408.03095*, 2024, [Online]. Available: https://arxiv.org/abs/2408.03095

[44] Juliano Cesar Pancher, Jorge Melegati & Eduardo Martins Guerra, "Exploratory Test-Driven Development Study with ChatGPT in Different Scenarios," *Agil. Process. Softw. Eng. Extrem. Program.*, 2025, [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-031-94544-1_10

[45] Moritz Mock, Jorge Melegati & Barbara Russo, "Generative AI for Test Driven Development: Preliminary Results," *Agil. Process. Softw. Eng. Extrem. Program. – Work.*, pp. 24–32, 2025, [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-031-72781-8_3

[46] Max Schäfer, Sarah Nadi, Aryaz Eghbali, Frank Tip, "An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation," *arXiv:2302.06527*, 2023, [Online]. Available: https://arxiv.org/abs/2302.06527

[47] Saranya Alagarsamy, Chakkrit Tantithamthavorn, "A3Test: Assertion-Augmented Automated Test case generation," *Inf. Softw. Technol.*, vol. 176, p. 107565, 2024, doi: https://doi.org/10.1016/j.infsof.2024.107565.

[48] J. Park, S. An, D. Youn, G. Kim, and S. Ryu, "JEST: N+1-version differential testing of both javascript engines and specification," *Proc. - Int. Conf. Softw. Eng.*, pp. 13–24, Nov. 2021, doi: 10.1109/ICSE43902.2021.00015.

[49] Christopher Foster, Abhishek Gulati, "Mutation-Guided LLM-based Test Generation at Meta," *Proc. ACM SIGSOFT Symp. Found. Softw. Eng.*, 2025, [Online]. Available: https://dl.acm.org/doi/10.1145/3696630.3728544

[50] Shaker Mahmud Khandaker, Fitsum Kifetew, Davide Prandi, Angelo Susi, "AugmenTest: Enhancing Tests with LLM-Driven Oracles," *arXiv:2501.17461*, 2025, [Online]. Available: https://arxiv.org/abs/2501.17461

[51] N. Mani and S. Attaranasl, "Adaptive Test Healing using LLM/GPT and Reinforcement Learning," *2025 IEEE Int. Conf. Softw. Testing, Verif. Valid. Work. ICSTW 2025*, pp. 9–16, 2025, doi: 10.1109/ICSTW64639.2025.10962516.

[52] Andrea Lops, Fedelucio Narducci, Azzurra Ragone, Michelantonio Trizio, Claudio Bartolini, "A System for Automated Unit Test Generation Using Large Language Models and Assessment of Generated Test Suites," *arXiv:2408.07846*, 2024, [Online]. Available: https://arxiv.org/abs/2408.07846

[53] Zejun Wang, Kaibo Liu, "HITS: High-coverage LLM-based Unit Test Generation via Method Slicing," *Proc. - 2024 39th ACM/IEEE Int. Conf. Autom. Softw. Eng. ASE 2024*, 2024, [Online]. Available: https://dl.acm.org/doi/10.1145/3691620.3695501

[54] Hengcheng Zhu, Lili Wei, "StubCoder: Automated Generation and Repair of Stub Code for Mock Objects," *ACM Trans. Softw. Eng. Methodol.*, 2023, [Online]. Available: https://dl.acm.org/doi/10.1145/3617171

[55] Daniel Planötscher, "NLP and GenAI in Agile Project Management: A Systematic Mapping Study," *Agil. Process. Softw. Eng. Extrem. Program. – Work.*, pp. 41–49, 2025, [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-032-05799-0_5

[56] Wendkuuni C. Ouedraogo, Kader Kabore, "LLMs and Prompting for Unit Test Generation: A Large-Scale Evaluation," *Proc. - 2024 39th ACM/IEEE Int. Conf. Autom. Softw. Eng. ASE 2024*, 2024, [Online]. Available: https://dl.acm.org/doi/10.1145/3691620.3695330

[57] Giovanni Grano, Simone Scalabrino, "An empirical investigation on the readability of manual and generated test cases," *Proc. - Int. Conf. Softw. Eng.*, 2018, [Online]. Available: https://dl.acm.org/doi/10.1145/3196321.3196363

[58] ZhangJunwei, HuXing, "Automated Unit Test Generation via Chain-of-Thought

Prompt and Reinforcement Learning from Coverage Feedback," *ACM Trans. Softw. Eng. Methodol.*, 2026, [Online]. Available: https://dl.acm.org/doi/10.1145/3745765

[59] Goran Petrovic, Marko Ivankovic, "Practical Mutation Testing at Scale: A view from Google," *IEEE Trans. Softw. Eng.*, 2022, [Online]. Available: https://dl.acm.org/doi/10.1109/TSE.2021.3107634

[60] Ana B. Sánchez, Pedro Delgado-Pérez, "Mutation testing in the wild: findings from GitHub," *Empir. Softw. Eng.*, 2022, [Online]. Available: https://dl.acm.org/doi/abs/10.1007/s10664-022-10177-8#:~:text=Mutation testing exploits artificial faults,research events on the topic.

[61] Ana B. Sanchez, Jose A. Parejo, "Mutation Testing in Practice: Insights From Open-Source Software Developers," *IEEE Trans. Softw. Eng.*, vol. 50, pp. 1130–1143, 2024, doi: 10.1109/TSE.2024.3377378.

[62] Ellen Arteca, Sebastian Harner, "Nessie: automatically testing JavaScript APIs with asynchronous callbacks," *Proc. - Int. Conf. Softw. Eng.*, 2022, [Online]. Available: https://dl.acm.org/doi/10.1145/3510003.3510106

[63] Mitchell Olsthoorn, Dimitri Stallenberg, "Syntest-JavaScript: Automated Unit-Level Test Case Generation for JavaScript," *Proc. - 2024 IEEE/ACM Int. Work. Search-Based Fuzz. Testing, SBFT 2024*, 2024, [Online]. Available: https://dl.acm.org/doi/10.1145/3643659.3643928

[64] P. A. & T. A. Kari Sainio, "Prompt Patterns for Agile Software Project Managers: First Results," *Softw. Bus.*, pp. 190–204, 2024, [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-031-53227-6_14

[65] D. Stallenberg, M. Olsthoorn, and A. Panichella, "Guess What: Test Case Generation for Javascript with Unsupervised Probabilistic Type Inference," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 13711 LNCS, pp. 67–82, 2022, doi: 10.1007/978-3-031-21251-2_5.