

AI-Driven Software Testing for a Test Center of Excellence: A Systematic Mapping and Evidence Synthesis

Ali Ahmad*

System and Information Technology Study Program, Faculty of Science and Technology, University of Darunnajah, Jakarta, Indonesia

ABSTRACT

Open Access

Type: Systematic Literature Review

Article:

Received: September 13, 2026

Revised: September 29, 2026

Accepted: October 20, 2026

Issued: October 30, 2026

© Ahmad. (2026).



This is an open-access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license

***Correspondence Address:**

ali.ahmad@darunnajah.ac.id

Artificial intelligence (AI) is increasingly used to automate and optimize software testing, but the evidence is distributed across heterogeneous testing activities, AI techniques, and evaluation measures. This study maps AI-driven software-testing research and translates the evidence into capabilities relevant to a Test Center of Excellence (TCoE). A systematic evidence workflow was applied to a unified audit corpus of 2,072 records, followed by focused eligibility reconciliation and full-text verification. The final focused evidence workset contains 29 records: 15 primary empirical studies, 9 secondary studies, 1 scope exclusion, and 4 full-text/access exclusions. The primary evidence covers machine learning, reinforcement learning, evolutionary and search-based methods, deep learning, explainable AI, and large language models. The supported testing activities include test generation, selection and prioritization, minimization and optimization, fuzzing, validation, adaptive automation, and maintenance. The synthesis shows that AI should be treated in a TCoE as a governed capability layer rather than as a single automation tool. The proposed capability model links AI techniques to testing activities, measurable outcomes, reusable TCoE capabilities, and governance with human oversight. The study also identifies gaps in reliability, context awareness, test-oracle support, long-term maintenance, CI/CD integration, reproducibility, and standardized evaluation.

Keywords: *Artificial Intelligence, Software Testing, Test Center of Excellence, Test Automation, Test Generation, Machine Learning*

1. INTRODUCTION

Software testing is increasingly expected to operate as a continuous engineering capability rather than as a downstream verification activity. In continuous integration and continuous delivery environments, large numbers of commits and repeated test executions create pressure to reduce execution cost while preserving fault-detection capability. FastLane, for example, used historical test and commit data to make data-driven minimization decisions and reported a reduction of 18.04% in test time while maintaining 99.99% test-outcome accuracy on a large-scale service [1]. This illustrates a central motivation for AI-assisted testing: the objective is not automation for its own sake, but better allocation of limited testing time and resources.

The literature has expanded beyond conventional machine-learning prediction. Deep-learning systems require testing techniques that consider their data-driven and non-traditional behavior. Importance-Driven Deep Learning System Testing introduced a test adequacy perspective based on component importance and semantic diversity [2]. Data distribution-aware test selection similarly treats testing as a selection problem in which only a strategically chosen subset of newly collected data is labeled and used for model enhancement [3]. These studies indicate that AI can be both the object of testing and the mechanism used to perform testing; related work also examines alternative adequacy perspectives for deep-learning system testing[4].

AI has also been applied to test-input prioritization and regression-test optimization. Dang et al. investigated test-input prioritization for machine-learning classifiers and provided an empirical treatment of prioritization in the context of ML systems [5]. Khan et al. examined feature-selection techniques for ML-based regression test prioritization, emphasizing the relationship between feature representation and prioritization performance in continuous-integration settings[6]. Marijan compared machine-learning approaches for test-case prioritization under different test-history sizes and time budgets using industrial continuous-integration data[7].

Search-based and reinforcement-learning approaches extend this optimization perspective toward adaptive test generation. Esnaashari and Damia combined genetic algorithms with

reinforcement learning to accelerate automated test-data generation[8]. Konidena et al. proposed an RL-driven framework for adaptive test-case generation in Agile environments[9]. The evidence therefore suggests a progression from static prioritization toward sequential decision making in which the testing process can respond to execution feedback.

For deep-learning systems, fuzzing and test generation have become important research directions. XMutant uses explainable artificial intelligence to guide fuzzing toward influential input regions and failure-inducing inputs [19]. TAEFuzz addresses image-based deep-learning systems through transferable adversarial examples and automatic fuzzing[10]. These approaches demonstrate that AI can provide search guidance, not merely a classifier for a testing decision.

More recent work extends AI-assisted testing into large language models (LLMs), project-context reasoning, and adaptive maintenance. CITYWALK uses project-dependency awareness and language-specific knowledge to improve LLM-based C++ unit-test generation[11]. Deder et al. study hallucination-reduced unit-test generation through fine-tuning and structural constraints[12]. An adaptive scriptless behavior-driven development framework combines machine learning, reinforcement learning, and self-healing automation to respond to changes in user interfaces and application behavior[13].

Despite this breadth, the evidence remains fragmented. Studies differ in testing activity, AI technique, dataset, evaluation metric, and level of practical maturity. A technical method may demonstrate strong results for a specific testing task without addressing governance, reuse, quality gates, traceability, or organizational adoption. This creates a gap between AI-for-testing research and the operational responsibilities normally associated with a Test Center of Excellence (TCoE).

A TCoE can be understood in this study as an organizational capability layer responsible for reusable testing practices, standards, automation, measurement, governance, and continuous improvement. The novelty of this study is therefore not a new AI algorithm. Instead, it is an evidence-based mapping that organizes heterogeneous AI testing techniques into a TCoE-oriented capability framework

while preserving the distinction between technical testing evidence and direct TCoE evidence.

The objective of this research is to systematically map AI-driven software-testing evidence, identify the testing activities and evaluation measures supported by that evidence, and derive a practical capability framework for integrating AI-enabled testing into TCoE operations. The manuscript is an original synthesis of the reviewed evidence and develops its TCoE interpretation from the author's systematic mapping and evidence-reconciliation process.

CONTRIBUTIONS

1. Evidence mapping contribution. The study provides a reconciled mapping of AI techniques, software-testing activities, and reported evaluation evidence across the 15 primary empirical studies.
2. TCoE capability contribution. The study derives an evidence-based/proposed capability framework that translates heterogeneous AI-testing functions into reusable TCoE capabilities.
3. Governance contribution. The study integrates governance, human oversight, quality gates, traceability, and continuous improvement into the proposed AI-TCoE operating model, without claiming organizational validation.

2. METHODS

2.1 Research Questions

- RQ1: What AI techniques are used to support software testing?
- RQ2: Which software-testing activities are most frequently supported or automated by AI?
- RQ3: What TCoE capabilities can be derived from AI-enabled software testing research?
- RQ4: How are AI-based testing approaches evaluated and what evidence is reported?
- RQ5: What research gaps and challenges remain for integrating AI-enabled testing into TCoE practice?

2.2 Search and Evidence Workflow

The review followed a controlled evidence workflow developed specifically for

the AI-driven software-testing stream and aligned with established systematic literature review guidance [14], systematic mapping guidance[15], and transparent systematic-review reporting principles[16]. The search protocol separated organizational/TCoE terminology from AI-driven software-testing technology terminology so that the two evidence streams could be connected only after retrieval, deduplication, and eligibility screening. The Stream B protocol covered AI and software testing, AI test generation, AI-driven automation, generative AI/LLMs, and machine-learning optimization of testing activities. The Lens search window was 2015–2026, as defined by the locked review protocol. The unified audit corpus contains 2,072 records. This value is retained as an audit/corpus denominator rather than as a final included-study count. Focused reconciliation produced a 29-record evidence workset. The work set was subjected to primary/secondary classification, full-text verification, evidence extraction, and final access resolution.

Search sources, exact query families, and retrieval audit

The production search was executed in The Lens Scholarly Works on 13 August 2026 using a 2015–2026 publication window. The protocol used two complementary streams. Stream A targeted organizational/TCoE terminology; Stream B targeted AI-driven software-testing terminology. Scopus, Web of Science, IEEE Xplore, ACM Digital Library, ScienceDirect, and SpringerLink were defined as database-specific templates during protocol development, but they were not executed as independent production sources for the final 2,072-record corpus. This distinction is reported explicitly to avoid overstating database coverage.

For reproducibility, the production search used two controlled streams. Stream A used organizational terms for TCoE/test-factory concepts, including the exact terms “Testing Center of Excellence,” “Test Factory,” and “centralized testing.” Stream B used the following exact query families: B1 = (“artificial intelligence” OR “machine learning” OR “deep learning”) AND (“software testing” OR “test automation” OR “automated testing”); B2 = (“artificial intelligence” OR “machine learning” OR “deep learning”) AND (“test case

generation” OR “test generation” OR “automated test generation”); B3 = (“AI-driven testing” OR “AI-based testing” OR “intelligent test automation” OR “intelligent software testing”); B4 = (“generative AI” OR “large language model” OR LLM OR GenAI) AND (“software testing” OR “test automation” OR “test generation”); and B5 = (“machine learning” OR “reinforcement learning”) AND (“test prioritization” OR “test selection” OR “test optimization” OR “test maintenance”). The production execution was performed in The Lens; other databases named in earlier protocol templates were not independently executed for the locked final corpus.

Table 1. Production search streams, exact query families, and raw retrieval counts.

Stream/ query	Exact query or controlled term	Raw/ exported	Role in audit
A1	"Testing Center of Excellence"	14	Direct TCoE terminology
A5	"Test Factory"	195	Organizational/centralized testing terminology
A6	"Centralized testing"	1000	Broad organizational retrieval; export-cap warning
B1–B2	B1: ("artificial intelligence" OR "machine learning" OR "deep learning") AND ("software testing" OR "test automation" OR "automated testing"); B2: same AI block AND ("test case generation" OR "test generation" OR "automated test generation") ("AI-driven testing" OR "AI-based testing" OR "intelligent test automation" OR "intelligent software testing")	1832	AI testing + generation; combined export
B3	"intelligent test automation" OR "intelligent software testing"	373	AI-driven/intelligent testing

Stream/ query	Exact query or controlled term	Raw/ exported	Role in audit
B4	("generative AI" OR "large language model" OR LLM OR GenAI) AND ("software testing" OR "test automation" OR "test generation") ("machine learning" OR "reinforcement learning") AND ("test prioritization" OR "test selection" OR "test optimization" OR "test maintenance")	1490	GenAI/LM testing
B5		1601	Search-result count only; full export unavailable

The available record-level exports from A1, A5, A6, B1–B2, B3, and B4 contained 4,904 records. Conservative computational deduplication used normalized DOI when available, otherwise normalized Lens ID and then normalized title, producing 4,607 candidate records after removal of 297 duplicate/overlap rows. B5 was retained as a documented search-result count because the complete result set could not be exported. A subsequent eligibility-master stage produced 2,072 records for audit-level reconciliation; at this stage 670 were flagged as candidate-includes, 469 as candidate-excludes, and 933 as pending human review. These labels are audit-stage classifications, not final PRISMA inclusion decisions.

Screening responsibility, disagreement handling, and access resolution

Title/abstract screening, evidence reconciliation, and extraction were conducted by the first author using the locked protocol and the study-level audit sheets. The workflow preserved source identifiers, decision status, and exclusion rationale. Because an independently coded second-reviewer dataset was not available, an inter-rater agreement statistic is not reported. Ambiguous records were retained in a pending state until sufficient evidence was available, rather than being forced into inclusion or exclusion. Four records ultimately remained access-limited at the final resolution stage and were excluded only because C4 could not be satisfied.

Separate methodological quality assessment

After eligibility, each included primary study was appraised using five reporting-quality dimensions: (Q1) clarity of the testing problem and AI technique; (Q2) clarity of dataset/system and experimental context; (Q3) clarity of baseline or comparator; (Q4) clarity of evaluation procedure and reported outcome; and (Q5) reproducibility information sufficient to understand implementation or experimental conditions. Each dimension was recorded as Yes, Partial, or No. The quality layer was not used to manufacture a pooled quality score and was not used to exclude an otherwise eligible study solely for incomplete reporting; instead, it was used to qualify the synthesis and identify evidence-strength limitations.

Table 2. Methodological quality assessment checklist applied after eligibility.

Quality dimension	Operational question	Use in synthesis
Q1 Problem/technique clarity	Is the testing problem and AI mechanism sufficiently specified?	Technique/activity classification
Q2 Context clarity	Are system, dataset, benchmark, or experimental context described?	Transferability interpretation
Q3 Comparator clarity	Is a baseline/comparator identified where comparison is claimed?	Comparative claims
Q4 Evaluation clarity	Are metrics, procedure, and outcomes sufficiently reported?	Evidence-strength interpretation
Q5 Reproducibility detail	Are implementation/configuration details sufficient for replication or audit?	Replication/audit claims

2.3 Eligibility Criteria

A study was promoted to the primary evidence set only when five eligibility gates were satisfied: C1, direct software-testing relevance; C2, primary empirical-study status; C3, AI applied to a testing activity; C4, sufficient verified full-text evidence; and C5, extractable empirical evaluation evidence. Eligibility was separated from methodological quality: C1–C5 determined whether a study could enter the evidence set, whereas quality appraisal qualified the confidence and interpretation of extracted evidence. The protocol intentionally avoided promoting records merely because their titles or abstracts contained AI and testing keywords.

The final focused workset reconciles to 29 records: 15 primary empirical studies, 9

secondary studies, 1 scope exclusion, and 4 full-text/access exclusions. The four access exclusions remained pending until the final access-resolution stage; after the final attempt, C4 could not be satisfied and the records were excluded with an explicit reason. This treatment prevents inaccessible records from being silently counted as either included or excluded on the basis of weak evidence.

2.4 Data Extraction and Synthesis

The extraction schema captured bibliographic identity, AI technique, testing activity, target system, empirical setting, evaluation metrics, reported outcomes, limitations, and TCoE relevance. Because the studies use heterogeneous evaluation measures, the synthesis is qualitative and matrix-based rather than a pooled meta-analysis, consistent with evidence-synthesis approaches used in software-engineering reviews[17]. Metric families were grouped into coverage, defect/fault detection, execution efficiency, generated-test quality, mutation score, test smells, metamorphic validity, false alarms, precision/specificity, inconsistency, reliability, and maintenance indicators.

The TCoE mapping was derived through a traceable chain: AI technique → testing activity → measurable testing outcome → TCoE capability → governance and human oversight → continuous improvement. This prevents the TCoE framework from treating every AI testing paper as if it were an organizational TCoE study.

TCoE capability derivation procedure

The five TCoE capabilities were derived after study-level coding rather than assigned directly from paper titles. For each primary study, the extraction first recorded the AI technique, then the testing activity, then the measurable testing outcome. These technical elements were mapped to an operational TCoE capability, followed by a governance/human-oversight implication and a continuous-improvement requirement. Capabilities were retained only when the testing function could be operationally connected to a TCoE responsibility; the mapping does not imply that the source study itself evaluated a TCoE. The resulting five capabilities are Automated Test Development, Test Optimization, Adaptive

Automation, Intelligent Maintenance, and Governance and AI-System Evaluation.

The derivation is therefore an evidence-to-organization interpretation: technical evidence → testing activity → measurable outcome → TCoE capability → governance/human oversight → continuous improvement. The last three elements are author-derived organizational implications and are explicitly distinguished from direct empirical findings in the primary studies.

3. RESULTS AND DISCUSSION

3.1 Evidence Reconciliation and Primary-Study Set

The final focused evidence workset contains 29 records: 15 primary empirical studies, 9 secondary studies, 1 scope exclusion, and 4 full-text/access exclusions. The 15 primary studies are: FastLane[1]; Importance-Driven Deep Learning System Testing[2]; data distribution-aware test selection [3]; test-input prioritization [5]; feature-selection-based regression test prioritization[6]; comparative ML test-case prioritization[7]; genetic-algorithm/reinforcement-learning test-data generation [8]; RL-driven adaptive test generation[9]; XMutant [19]; TAEFuzz[10]; CITYWALK[11]; hallucination-reduced unit-test generation[12]; adaptive self-healing BDD automation [13]; multistage-based genetic-algorithm test generation [18]; and multi-objective search for model-based testing [20]. The 9 secondary studies were retained for contextual synthesis but were not included in the primary denominator. One record was excluded for scope, while four were excluded after final access resolution because sufficient verified full text could not be obtained.

The boundary between the 2,072-record audit corpus and the 29-record focused evidence workset is methodologically important. The 2,072 value documents the scale of the retrieved and reconciled corpus; it does not mean that all 2,072 records underwent the same final full-text eligibility process. The locked primary denominator of 15 therefore applies to the focused evidence synthesis.

Table 3. Evidence reconciliation from search retrieval to the final focused workset.

Stage	Records	Interpretation
Raw exported search streams	4,904	Available record-level exports across A1, A5, A6, B1–B2, B3, and B4
Computationally deduplicated candidates	4,607	297 duplicate/overlap rows removed using DOI/Lens ID/title
Eligibility master	2,072	Audit-level master; not final inclusion
Focused evidence workset	29	Final focused reconciliation
Primary empirical studies	15	Final primary denominator
Secondary studies	9	Contextual synthesis only
Scope exclusion	1	Outside final protocol scope
Full-text/access exclusions	4	C4 not satisfied after final access resolution

The four access exclusions were retained explicitly rather than silently omitted: “Optimizing Continuous Integration by Dynamic Test Proportion Selection,” “On the Use of Artificial Intelligence in Software Testing: State of the Art and Application to Satellite Systems,” “Legacy Code Test Generator Using Large Language Model,” and “LLM-Based Unit Test Cases Generation for React, Vue, and Angular Components.” For each, available metadata indicated potential relevance, but verified full-text evidence sufficient for C4/C5 could not be obtained in the final access-resolution attempt.

3.2 RQ1 — AI Techniques Used in Software Testing

Across the 15 primary studies, six AI families were coded using multi-label classification because several studies combine techniques. Machine learning appears in 5/15 studies; reinforcement learning in 4/15; evolutionary/search/adversarial optimization methods in 4/15; deep learning in 4/15; LLM/GenAI in 2/15; and explainable AI in 1/15. The counts are not additive because hybrid studies contribute to more than one family. Machine learning supports prediction, selection, prioritization, and minimization [1], [5], [6], [7]; reinforcement learning supports adaptive decisions and search[8], [9], [13], [20]; evolutionary/search methods support test generation and optimization[8], [10], [18], [20]; deep learning appears both as the testing target and as a component of AI-enabled testing[2],

[3], [10], [19] ;XAI guides failure-oriented fuzzing [19]; and LLMs support context-aware unit-test generation[11], [12]

Deep learning is both a system-under-test technology and a testing target. Importance-driven testing uses internal component importance to assess semantic diversity [2], while data distribution-aware selection addresses distribution shift during DNN enhancement[3]. XMutant and TAEFuzz extend DL testing toward explanation-guided and adversarial robustness-oriented fuzzing[10], [19]. LLM/GenAI approaches expand test automation into executable code and test-artifact generation, including project-context-aware C++ unit testing and hallucination mitigation[11], [12].

Table 4. AI technique families in the 15-study primary evidence set.

AI family	Studies (n/15)	Primary testing role	Representative studies
Machine learning	5	Prediction, selection, prioritization, minimization	[1], [5], [6], [7], [13]
Reinforcement learning	4	Adaptive decisions, search, generation, self-healing	[8], [9], [13], [20]
Evolutionary/search/adversarial	4	Test generation, optimization, robustness search	[8], [10], [18], [20]
Deep learning	4	DL-system testing, data selection, robustness/fuzzing	[2], [3], [10], [19]
LLM/GenAI	2	Unit-test generation and context-aware automation	[11], [12]
Explainable AI	1	Guided fuzzing and failure search	[19]

Table 5. Study-level extraction summary for the 15 primary empirical studies.

Study / evidence type	AI family	Testing activity	Evaluation evidence reported	TCoE capability mapping
FastLane [1] Primary empirical	ML	Minimization; selection	Test-time reduction; outcome accuracy	Test Optimization
Importance-Driven DL Testing [2] Primary empirical	DL	Test adequacy; generation	Semantic diversity; robustness-oriented testing	Governance and AI-System Evaluation
Data Distribution-Aware Selection [3] Primary empirical	DL	Selection	Model-enhancement accuracy under distribution shift	Test Optimization

Study / evidence type	AI family	Testing activity	Evaluation evidence reported	TCoE capability mapping
Test Input Prioritization [5] Primary empirical	ML	Prioritization	Prioritization /fault-detection performance	Test Optimization
Feature Selection for RTP [6] Primary empirical	ML	Prioritization	RTP performance ; feature-reduction cost	Test Optimization
Comparative ML Prioritization [7] Primary empirical	ML	Prioritization	Fault-prediction accuracy across history/time budgets	Test Optimization
Esnaashari & Damia [8] Primary empirical	RL + evolutionary	Test-data generation	Coverage; fitness evaluations; generation speed	Automated Test Development / Test Optimization
Konidena et al. [9] Primary empirical	RL	Adaptive generation ; prioritization	Coverage; execution efficiency; defect detection	Adaptive Automation
XMutant [19] Primary empirical	XAI + DL	Fuzzing	Failure rate; AUFC-based efficiency	Automated Test Development / Governance
TAEFuzz [10] Primary empirical	Evolutionary/search + DL	Robustness fuzzing	Failure discovery/robustness evidence	Automated Test Development
CITYWALK [11] Primary empirical	LLM/GenAI	C++ unit-test generation	Generated-test correctness/quality; project comparison	Automated Test Development
Hallucination-Reduced Unit Test Generation [12] Primary empirical	LLM/GenAI	Unit-test generation	Executability/structural robustness	Automated Test Development / Governance
Adaptive Self-Healing BDD [13] Primary empirical	ML + RL	Maintenance; adaptive generation	Script maintenance time; defect detection	Adaptive Automation / Intelligent Maintenance
Multistage GA [18] Primary empirical	Evolutionary/search/adversarial	Test generation ; optimization	Mutation score, accuracy, time, event-based evaluation	Automated Test Development / Test Optimization

Study / evidence type	AI family	Testing activity	Evaluation evidence reported	TCoE capability mapping
Multi-objective Search for MBT [20] Primary empirical	RL + search	Model-based generation	Multi-objective search effectiveness and test-generation outcomes	Automated Test Development / Test Optimization

3.3 RQ2 — Software-Testing Activities Supported by AI

RQ2 shows that test generation is the most frequently represented activity: 8/15 primary studies involve generation. Prioritization appears in 4/15, selection in 3/15, optimization in 2/15, fuzzing in 2/15, while minimization, maintenance/self-healing, and adequacy each appear in 1/15. These are multi-label counts because a study may support more than one activity. Generation includes genetic/RL search, adaptive Agile generation, fuzzing-oriented generation, and LLM-based unit-test generation[8], [9], [10], [11], [12], [18], [20].

Selection and prioritization form the second major activity family. FastLane supports minimization/selection[1]; data distribution-aware selection chooses a subset of test data for DNN enhancement[3]; and ML-based prioritization studies investigate input value, feature representation, test history, and execution budgets[5], [6], [7]. This family is particularly relevant to CI/CD because time budgets constrain the number of tests that can be executed.

Optimization and minimization are related but operationally distinct. Minimization seeks to reduce redundant execution, while optimization seeks better test inputs, test sequences, or decisions under resource constraints[1], [18], [20]. The reviewed evidence supports both uses, with language-model-based minimization providing a complementary example of scalable test-suite reduction[21], suggesting that a TCoE should manage them as reusable optimization capabilities rather than isolated scripts. Fuzzing and robustness testing constitute another activity family. XMutant uses XAI to guide semantic fuzzing[19], while TAEFuzz uses transferable adversarial examples for image-based DL systems[10]. These studies broaden the TCoE view of automation from functional regression testing to AI-system robustness and failure discovery.

Adaptive automation and maintenance are emerging activities. The self-healing BDD framework uses dynamic locators, adaptive generation, and reinforcement learning to respond to application changes[13]. Importance-driven testing contributes an adequacy-oriented activity[2]. These findings suggest that future TCoEs need maintenance and quality-assessment services that continuously detect broken tests, assess whether repair or generation actions are safe, and retain evidence of changes.

Table 6. Study-by-activity frequency mapping (multi-label).

Testing activity	Studies (n/15)	Examples
Test generation	8	[2], [8], [9], [11], [12], [13], [18], [20]
Test prioritization	4	[5], [6], [7], [9]
Test selection	3	[1], [3], [9]
Optimization	2	[18], [20]
Fuzzing/robustness	2	[10], [19]
Minimization	1	[1]
Maintenance/self-healing	1	[13]
DL test adequacy	1	[2]

3.4 RQ3 — TCoE Capability Mapping

The evidence supports five operational TCoE capabilities. The first is Automated Test Development, which covers reusable generation of unit tests, test data, and fuzzing inputs. The second is Test Optimization, covering selection, prioritization, minimization, and search-based optimization. The third is Adaptive Automation, covering RL-based decisions and self-healing execution. The fourth is Intelligent Maintenance, covering test reuse, repair, and adaptation to application changes. The fifth is Governance and AI-System Evaluation, covering validation, traceability, quality gates, risk controls, and human oversight.

Table 7. AI-enabled TCoE capability mapping.

TCoE capability	Evidence mapping	Operational responsibility
Automated Test Development	Generation, fuzzing, unit tests	Reusable test assets and standards
Test Optimization	Selection, prioritization, minimization	Resource-aware CI/CD execution
Adaptive Automation	RL and self-healing	Response to changing systems
Intelligent Maintenance	Repair, reuse, adaptation	Reduced test-maintenance effort
Governance and Evaluation	Validation, metrics, traceability	Human oversight and quality gates

These capabilities should not be interpreted as five separate tools. A mature

TCoE would integrate them into a pipeline in which AI generates or recommends testing actions, evidence is collected from execution, outcomes are measured, and governance controls determine whether recommendations can be promoted into operational test assets.

3.5 RQ4 — Evaluation Evidence

Evaluation evidence is heterogeneous and was therefore normalized into metric families rather than pooled into a single score. The primary studies report measures spanning execution efficiency/time, prediction or prioritization performance, coverage and test adequacy, fault/failure detection, robustness, generated-test quality/executability, and maintenance/repair effectiveness. Examples include FastLane time savings and outcome accuracy[1], model-enhancement accuracy under distribution shift[3], prioritization performance and feature-reduction cost [5], [6], [7], search efficiency and coverage for automated test-data generation[8], failure rate and efficiency for XMutant[19], robustness-oriented failure discovery for TAEFuzz[10], generated-test quality for CITYWALK[11], and script-maintenance time and defect-detection rate for self-healing automation[13].

LLM-based test generation introduces additional quality dimensions. CITYWALK evaluates executable and high-coverage C++ unit tests using project-dependency information [11], while hallucination-reduced generation focuses on structural integrity and executable test quality [12]. Self-healing automation introduces maintenance cost, suite stability, and repair effectiveness as evaluation concerns[13]. Because the archived extraction did not normalize every study-specific metric into a validated common counting dictionary, the manuscript does not fabricate study-frequency counts for individual metric names. Instead, it reports the metric families represented in the primary evidence and distinguishes those reported measures from an author-derived TCoE metric portfolio. For TCoE use, the evidence supports four portfolio dimensions: effectiveness (faults detected, coverage, validity), efficiency (execution time and resource use), reliability (false alarms, robustness, stability), and maintainability (repair effort, reuse, adaptation). No pooled effect size or cross-study performance score is claimed.

3.6 RQ5 — Research Gaps and Challenges

The first gap is reliability of AI-generated tests. LLM-based generation can produce syntactically plausible but semantically weak or hallucinated tests, motivating work on hallucination reduction and structural validation [12]. The second gap is context awareness. CITYWALK demonstrates the value of project-dependency and language-specific knowledge [11], indicating that generic prompts alone may be insufficient for complex systems.

The third gap is long-term maintenance. Self-healing automation provides evidence that AI can respond to UI and application changes[13], but long-term reliability, repair safety, and regression risk require further study. The fourth gap concerns test oracles and validation. Generated tests still require mechanisms for determining whether observed behavior is correct, especially for AI systems and nondeterministic applications.

The fifth gap is governance. AI-generated or AI-prioritized tests affect release decisions, yet the reviewed technical studies rarely provide a complete organizational governance model. A TCoE must therefore retain human accountability, traceability, approval rules, and quality gates. The sixth gap is CI/CD integration and reproducibility. Although several studies use CI or industrial data[1], [7], broader evidence on cross-project portability, benchmark standardization, and reproducible artifacts remains limited.

Finally, evaluation standards are fragmented. The absence of common benchmark suites and common reporting dimensions makes it difficult to compare techniques across testing activities. A TCoE-oriented research agenda should therefore prioritize reusable benchmarks, transparent datasets, artifact availability, standardized metric definitions, and longitudinal evaluation.

3.7 Integrated AI-TCoE Capability Framework

The synthesis produces an evidence-derived/proposed operating model with six stages; it is not presented as an empirically validated organizational framework; no reviewed study is treated as direct organizational validation of the TCoE model. First, an AI technique provides a capability such as prediction, generation, optimization,

explanation, or adaptation. Second, the capability is connected to a testing activity such as generation, prioritization, fuzzing, or maintenance. Third, execution produces measurable evidence. Fourth, the evidence is assessed against TCoE quality gates. Fifth, human oversight determines whether the AI recommendation or generated artifact can be promoted. Sixth, the outcome is stored as organizational knowledge and used for continuous improvement.

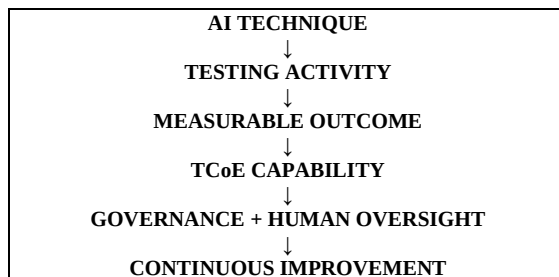


Figure 1. *Integrated AI-TCoE capability operating model.*

This model is intentionally governance-oriented. AI should not directly replace the TCoE's responsibility for test policy, release criteria, risk acceptance, evidence traceability, and auditability. Instead, AI becomes a capability layer that increases the speed and adaptability of testing while the TCoE supplies the organizational controls needed to make those capabilities dependable.

3.8 Threats to Validity

Search bias may omit relevant studies outside the selected databases or query vocabulary. The review mitigates this by maintaining separate search streams and preserving the audit corpus. Publication and access bias may also affect evidence availability; four focused records were excluded after final access resolution because sufficient verified full text could not be obtained.

Classification bias is possible when a paper could be interpreted as technical evidence, secondary evidence, or organizational evidence. The C1-C5 protocol and explicit primary/secondary classification reduce this risk. Screening and extraction were conducted by a single reviewer, so independent inter-rater agreement could not be estimated; this is a limitation of the review. Metric heterogeneity limits quantitative aggregation, so the synthesis uses metric families rather than pooled effect sizes.

The TCoE capability framework is an evidence-derived interpretation rather than a claim that all reviewed studies explicitly evaluated TCoEs. This distinction is essential: technical evidence is mapped to organizational capability only where the testing function can be operationally connected to TCoE responsibilities. No reviewed study is used as evidence that an organizational TCoE itself was implemented or validated; the organizational mapping is an author-derived synthesis of the reported testing functions.

Generalizability is also limited by the small primary evidence base (15 studies), heterogeneous contexts, and uneven reporting detail. The five TCoE capabilities should therefore be treated as an evidence-informed organizational synthesis rather than as a universal organizational structure. Future validation requires multi-site case studies and longitudinal industrial evaluation.

3.9 Study Characteristics and Evidence Distribution

The primary evidence set is heterogeneous in both technical focus and experimental context. Studies on test minimization and prioritization are closely connected to continuous integration and large-scale execution environments [1], [6], [7]. Studies on deep-learning testing focus more strongly on adequacy, data distribution, robustness, and failure discovery[2], [3], [10], [19]. LLM-oriented studies emphasize the quality and executability of generated test artifacts, while self-healing research emphasizes adaptation and maintenance[11], [12], [13]. This distribution matters for TCoE design because different AI techniques address different bottlenecks in the testing lifecycle.

At the level of testing artifacts, the evidence can be organized into three broad classes. The first class concerns test inputs and test data, including generated inputs, selected data, prioritized inputs, and fuzzing inputs. The second class concerns executable test artifacts, including unit tests and automated test scripts. The third class concerns test decisions, including which tests to execute, in which order, and under what time budget. A TCoE can therefore treat AI as a portfolio of services that operate on inputs, artifacts, and decisions rather than as a single platform feature.

An important observation is that the evaluation target changes with the testing

activity. A test-generation approach may be judged by coverage, validity, executability, or fault detection, whereas a test-minimization approach may be judged by time savings and prediction accuracy. A self-healing approach must additionally be assessed for repair correctness and stability. Consequently, a TCoE dashboard should not collapse these dimensions into a single AI-performance indicator. Instead, each AI capability should have an explicit quality contract that defines its intended outcome and acceptable failure modes.

3.10 Implications for TCoE Architecture

The evidence supports a layered TCoE architecture. At the bottom is the test-asset layer, containing source code, requirements, historical test executions, defect records, model artifacts, and project documentation. Above it is an AI service layer that provides prediction, generation, optimization, explanation, and adaptation. A measurement layer records execution outcomes and quality indicators. A governance layer controls access, approval, traceability, and risk. Finally, a knowledge layer stores validated prompts, models, test patterns, benchmark results, and lessons learned for reuse across teams.

Such an architecture addresses a recurring weakness in AI-testing research: the gap between an experimental technique and an operational capability. For example, an ML model that prioritizes regression tests is useful to a TCoE only when the model can consume reliable historical data, operate within CI/CD time constraints, expose performance indicators, and provide a fallback when prediction confidence is low[1], [5], [7]. Similarly, an LLM-based test generator requires project context, executable validation, and mechanisms for rejecting hallucinated or low-quality tests [11], [12].

The governance layer is particularly important for generated artifacts. A practical TCoE should distinguish between AI-generated recommendations, AI-generated test assets that have passed automated validation, and test assets approved for release-critical execution. This creates an evidence chain from generation to validation and approval. It also supports auditability when AI recommendations influence release decisions.

Human oversight should therefore be designed as an explicit control point rather than

as an informal practice. The human role is strongest where consequences are high, evidence is ambiguous, or the AI system has low confidence. In lower-risk repetitive activities, automated gates can be used to reduce manual effort. This risk-based allocation of human involvement is consistent with the observed diversity of AI-testing applications and with the need for TCoEs to maintain organizational accountability.

3.11 Comparison with the Existing Evidence Base

Recent secondary literature confirms that AI-testing research is broad but fragmented. Escalante-Viteri and Mauricio provide a decade-scale taxonomy of AI algorithms, testing problems, variables, and metrics[22]. Faraji and Pombo survey AI-driven test automation across planning, generation, execution, repair, and maintenance [23]. Ricca et al. focus on more than 3,600 grey-literature sources and 100 AI-driven test-automation tools, complemented by expert interviews[24]. Garousi et al. review 56 AI-powered testing tools and emphasize that full autonomy remains distant[25]. Tasarsu et al. focus specifically on LLM-based test-case generation and analyze 38 peer-reviewed studies[26]. Lahami and Krichen provide a 2026 architectural SLR of 37 GenAI-driven testing studies, emphasizing testing activities, architecture, pipelines, and evaluation metrics[27]. These studies establish a strong technical and tool-oriented evidence base, but they do not use TCoE capability/governance as the primary synthesis unit.

Compared with conventional rule-based automation, AI-enabled approaches provide a potential advantage when the testing environment changes or when the decision space is too large for manually specified rules. However, the evidence also shows that this flexibility introduces new failure modes. Prediction errors can omit useful tests, generated tests can be invalid or hallucinated, adaptive repairs can introduce unintended behavior, and AI-system testing itself requires specialized adequacy and robustness measures[2], [12], [13], [19]. Therefore, AI adoption should be accompanied by validation and fallback mechanisms.

The present review differs in emphasis rather than claiming a competing algorithmic

taxonomy. Its contribution is the explicit translation chain from AI technique to testing activity, measurable outcome, TCoE capability, governance/human oversight, and continuous improvement. The five capabilities are author-derived from the mapped technical evidence; they are not claimed to have been directly observed in all 15 primary studies. This boundary is important because a strong technical result does not by itself demonstrate organizational scalability, governance readiness, or TCoE effectiveness.

Table 8. Positioning against recent AI-driven software-testing reviews and surveys.

Review	Scope / evidence base	Primary synthesis focus	Boundary relative to this study
Escalant e-Viteri & Mauricio (2025) [22]	2014–2024 AI/software e-testing literature	Taxonomy of algorithms, problems, variables, metrics	Does not center TCoE capability/governance
Faraji & Pombo (2025) [23]	AI4SE-oriented research + industrial tools 3,600+ grey sources;	Techniques, tools, challenges across test cycle AI-assisted test-automation	Broader automation survey; not TCoE evidence mapping
Ricca et al. (2025) [24]	342 filtered documents; 100 tools; 5 experts	problems, solutions, tools	Grey-literature/practice emphasis rather than primary-study evidence synthesis
Garousi et al. (2025) [25]	56 AI-powered testing tools	Tool capabilities and limits of autonomy LLM datasets, techniques, languages, integration	Tool review; not study-level TCoE capability derivation
Tasarsu et al. (2026) [26]	38 peer-reviewed LLM test-generation studies	Testing activities, architecture, pipelines, metrics	LLM-specific; narrower than full AI/TCoE scope
Lahami & Krichen (2026) [27]	37 GenAI-driven testing primary studies	Evidence-to-TCoE capability, governance, human oversight	Architecture-centric; does not derive TCoE governance capabilities
This study	15 primaries + 9 secondaries in a 29-record focused workset		Proposed organizational synthesis; not empirically validated

3.12 Practical TCoE Implementation Roadmap

Phase 1 — Establish the evidence baseline (author-derived recommendation). The

TCoE should inventory current testing activities, historical execution data, defect records, CI/CD constraints, and existing automation assets. AI should not be introduced before the data and measurement baseline is understood.

Phase 2 — Introduce low-risk optimization (author-derived recommendation). Test selection, prioritization, and minimization are suitable early candidates because their outputs can often be compared with an existing deterministic test process. Metrics such as execution time, fault detection, and prediction accuracy can be monitored continuously[1], [5], [7].

Phase 3 — Expand generation and adaptive automation (author-derived recommendation). Once measurement and quality gates are mature, the TCoE can introduce automated test-data generation, fuzzing, unit-test generation, and adaptive automation[8], [13]. Generated artifacts should pass syntax, execution, coverage, and policy checks before operational adoption.

Phase 4 — Establish governance and continuous learning (author-derived recommendation). Approved AI capabilities should be versioned, monitored, periodically revalidated, and associated with explicit ownership. Changes in applications, datasets, models, or prompts should trigger appropriate re-evaluation. The TCoE should maintain a reusable knowledge base of successful patterns and documented failure cases.

Table 9. Practical TCoE implementation stages for AI-enabled testing.

TCoE stage	AI capability	Key metric	Control
Discover	Data-driven test selection	Selection effectiveness	Data quality check
Optimize	Prioritization/minimization	Time saved, fault detection	Fallback policy
Generate	Unit-test/test-data generation	Validity, coverage	Automated validation
Adapt	Self-healing/RL	Stability, repair success	Human approval for high risk
Govern	AI portfolio management	Reliability, drift, reuse	Audit and version control

3.13 Research Agenda for AI-Enabled TCoE

Based on the evidence gaps, future research should move beyond isolated

algorithm comparisons toward longitudinal evaluation of AI testing capabilities in realistic development environments. A useful benchmark should capture changes in source code, test suites, requirements, defect histories, CI/CD execution budgets, and model versions. Such a benchmark would allow researchers to evaluate not only whether an AI technique improves one metric, but whether the improvement remains stable as the system evolves.

A second research direction is standardized reporting. Studies should report the testing activity, baseline, dataset or project context, training data, AI model configuration, evaluation protocol, and failure cases in sufficient detail to support replication. For LLM-based approaches, prompts, context sources, model versions, temperature or sampling settings, generated-test filtering, and validation rules are particularly important. Without this information, comparisons between LLM testing approaches remain difficult.

A third direction is evidence-aware automation. AI-generated tests should carry provenance information that identifies the source requirements, code context, model or prompt version, validation results, and approval status. This would allow a TCoE to distinguish experimental artifacts from production-approved testing assets. It would also support audit trails when a generated test contributes to a release decision.

A fourth direction is adaptive governance. AI systems and software systems both change over time, so static approval is insufficient. A TCoE should monitor performance drift, changes in the application under test, changes in training data, and changes in generated-test quality. Future studies should investigate thresholds that trigger retraining, revalidation, human review, or temporary rollback of an AI testing capability.

A final direction is organizational evaluation. Most technical evidence focuses on algorithmic or testing outcomes, while TCoE adoption also depends on cost, skill requirements, integration effort, user trust, maintainability, and organizational readiness. Future systematic reviews and empirical studies should therefore connect technical metrics with operational outcomes such as cycle-time reduction, defect escape rate, test-maintenance effort, and reuse across projects.

CONCLUSION

This systematic mapping and evidence synthesis shows that AI-enabled software testing spans test generation, selection, prioritization, minimization, optimization, fuzzing, validation, adaptive automation, and maintenance. The final focused evidence workset contains 29 records: 15 primary empirical studies, 9 secondary studies, 1 scope exclusion, and 4 full-text/access exclusions. The 15-study primary base is heterogeneous and small, so the quantitative mapping is descriptive and multi-label rather than a pooled performance synthesis.

The main contribution is an evidence-derived/proposed AI-TCoE capability framework that connects AI techniques to testing activities, measurable outcomes, reusable testing capabilities, and governance with human oversight. The framework should not be interpreted as empirically validated organizational performance evidence. Its value is as a traceable synthesis that helps translate heterogeneous technical results into candidate TCoE capabilities while preserving the boundary between direct study evidence and author-derived recommendations.

Future research should establish standardized evaluation metrics, reproducible cross-project benchmarks, longitudinal studies of maintenance and self-healing, stronger test-oracle mechanisms, and governance models for AI-generated and AI-prioritized testing assets. Empirical validation of the proposed TCoE model through multi-site case studies, longitudinal industrial evaluation, and comparative organizational studies is a priority. These directions can move AI-enabled testing from task-specific demonstrations toward dependable enterprise testing capabilities.

DECLARATIONS

Author Contributions: The author was responsible for the conceptualization, methodology, investigation, data curation, formal analysis, and preparation of the manuscript. The author also reviewed and revised the manuscript, verified the supporting references and research findings, and approved the final version of the manuscript.

Funding: This research received no external funding.

Conflict of Interest: The author declares no conflict of interest.

Generative Artificial Intelligence Use Statement: ChatGPT (OpenAI) was used as an AI-assisted tool during the preparation of this manuscript primarily to support preliminary literature search strategy, conceptual mapping of research structures, structural outlining, and language refinement (including grammar correction and proofreading). The human author conducted all core analysis, evidence evaluation, interpretation of findings, and manuscript writing. The author reviewed and verified all AI-assisted outputs and retains full responsibility for the originality, accuracy, integrity, verification of sources, and conclusions presented. ChatGPT is not listed as an author or co-author.

Ethics Approval and Informed Consent: Not applicable. This study is a systematic mapping study and evidence synthesis based on published literature. It did not involve human participants, direct collection of personal data, or interventions requiring ethical approval or informed consent.

Data Availability: The data supporting the findings of this study are available from the author upon reasonable request, subject to any applicable restrictions on access or sharing.

REFERENCES

- [1] A. A. Philip, R. Bhagwan, R. Kumar, C. S. Maddila, and N. Nagappan, "FastLane: Test minimization for rapidly deployed large-scale online services," *Proc 41st IEEEACM Int Conf Softw. Eng. ICSE*, pp. 408–418, 2019, doi: 10.1109/ICSE.2019.00054.
- [2] S. Gerasimou, H. F. Eniser, A. Sen, and A. Cakan, "Importance-driven deep learning system testing," *Proc 42nd IEEEACM Int Conf Softw. Eng. ICSE*, pp. 1123–1134, 2020, doi: 10.1145/3377811.3380391.
- [3] Q. Hu *et al.*, "An empirical study on data distribution-aware test selection for deep learning enhancement," *ACM Trans Softw. Eng. Methodol.*, 2022, doi: 10.1145/3511598.
- [4] J. Kim, R. Feldt, and S. Yoo, "Evaluating Surprise Adequacy for Deep Learning System Testing," *ACM Trans Softw. Eng. Methodol.*, 2023, doi: 10.1145/3546947.
- [5] X. Dang, Y. Li, M. Papadakis, J. Klein, T. F. Bissyandé, and Y. Le Traon, "Test input prioritization for machine learning classifiers," *IEEE Trans Softw. Eng.*, pp. 413–442, 2024, doi: 10.1109/TSE.2024.3350019.
- [6] M. A. Khan *et al.*, "On the effectiveness of feature selection techniques in the context of ML-based regression test prioritization," *IEEE Access*, pp. 131556–131575, 2024, doi: 10.1109/ACCESS.2024.3459656.
- [7] D. Marijan, "Comparative study of machine learning test case prioritization for continuous integration testing," *Softw. Qual. J.*, 2023, doi: 10.1007/s11219-023-09646-0.
- [8] M. Esnaashari and A. H. Damia, "Automation of software test data generation using genetic algorithm and reinforcement learning," *Expert Syst. Appl.*, 2021, doi: 10.1016/j.eswa.2021.115446.
- [9] B. K. Konidena, A. R. Bairi, and T. Pichaimani, "Reinforcement learning-driven adaptive test case generation in Agile development," *Am. J. Data Sci. Artif. Intell. Innov.*, pp. 241–273, 2021.
- [10] S. Ji *et al.*, "TAEFuzz: Automatic fuzzing for image-based deep learning systems via transferable adversarial examples," *ACM Trans Softw. Eng. Methodol.*, 2025, doi: 10.1145/3714463.
- [11] Y. Zhang *et al.*, "CITYWALK: Enhancing LLM-based C++ unit test generation via project-dependency awareness and language-specific knowledge," *ACM Trans Softw. Eng. Methodol.*, 2026, doi: 10.1145/3763791.
- [12] M. Deder, S. Sahin, M. Yilmazer, and M. Karaköse, "Hallucination-reduced and robust accuracy unit test generation," *Int. J. Manag. Data Anal.*, pp. 191–205, 2026.
- [13] S. Senthil Murugan and M. Balamurugan, "An adaptive scriptless behavior-driven development automation framework with self-healing intelligence for evolving software applications," *J. Internet Serv.*

- Inf. Secur.*, pp. 291–310, 2026, doi: 10.58346/JISIS.2026.I1.017.
- [14] B. A. Kitchenham and S. Charters, “Guidelines for performing systematic literature reviews in software engineering,” 2007.
- [15] K. Petersen, S. Vakkalank, and L. Kuzniarz, “Guidelines for conducting systematic mapping studies in software engineering: An update,” *Inf. Softw. Technol.*, pp. 1–18, 2015, doi: 10.1016/j.infsof.2015.03.007.
- [16] M. J. Page, J. E. McKenzie, P. M. Bossuyt, and et al., “The PRISMA 2020 statement: An updated guideline for reporting systematic reviews,” *BMJ*, 2021, doi: 10.1136/bmj.n71.
- [17] D. S. Cruzes and T. Dybå, “Synthesizing evidence in software engineering research,” in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, Bolzano-Bozen Italy: ACM, Sep. 2010, pp. 1–10. doi: 10.1145/1852786.1852788.
- [18] A. Bala and R. S. Chhillar, “Automatic test cases generation using multistage-based genetic algorithm for object oriented testing,” *Int. J. Recent Technol. Eng.*, pp. 561–573, 2019, doi: 10.35940/ijrte.b1099.0982s1019.
- [19] X. Chen, M. Biagiola, V. Riccio, M. d’Amorim, and A. Stocco, “XMutant: XAI-based fuzzing for deep learning systems,” *Empir. Softw. Eng.*, 2026, doi: 10.1007/s10664-025-10792-1.
- [20] R. Wang, C. Artho, L. M. Kristensen, and V. Stolz, “Multi-objective search for model-based testing,” *Proc 2020 IEEE 20th Int Conf Softw. Qual. Reliab. Secur. QRS*, pp. 130–141, 2020, doi: 10.1109/QRS51102.2020.00029.
- [21] R. Pan, T. A. Ghaleb, and L. C. Briand, “LTM: Scalable and Black-Box Similarity-Based Test Suite Minimization Based on Language Models,” *IEEE Trans Softw. Eng.*, pp. 3053–3070, 2024, doi: 10.1109/TSE.2024.3469582.
- [22] A. Escalante-Viteri and D. Mauricio, “Artificial intelligence in software testing: A systematic review of a decade of evolution and taxonomy,” *Algorithms*, 2025, doi: 10.3390/a18110717.
- [23] A. Faraji and N. Pombo, “AI-driven software test automation: An AI4SE-oriented survey of techniques, tools, and challenges,” *IEEE Access*, pp. 183296–183313, 2025, doi: 10.1109/ACCESS.2025.3623944.
- [24] F. Ricca, A. Marchetto, and A. Stocco, “A multi-year grey literature review on AI-assisted test automation,” *Inf. Softw. Technol.*, 2025, doi: 10.1016/j.infsof.2025.107799.
- [25] V. Garousi, N. Joy, and D. Taibi, “AI-powered software testing tools: Full autonomy remains a distant goal,” *IEEE Softw.*, pp. 98–111, 2025, doi: 10.1109/MS.2025.3557895.
- [26] M. Tasarsu, A. V. Tokmak, and C. Catal, “Test case generation using large language models: A systematic literature review,” *Clust. Comput.*, 2026, doi: 10.1007/s10586-026-06021-z.
- [27] M. Lahami and M. Krichen, “Architectural perspectives on GenAI-driven software testing systems: A systematic literature review,” *Inf. Softw. Technol.*, 2026, doi: 10.1016/j.infsof.2026.108246.