

From Pareto to Neural: A Mathematical Survey of Multi-Objective Optimization Algorithms—With Applications to Software Testing

Xufan Zheng ^{1,*}  and Waqas Rasheed ² ¹ College of Science, Northwest Agriculture and Forestry University, Xianyang 712100, China² School of Information Engineering, Yangzhou University, Yangzhou 225012, China; dh23042@stu.yzu.edu.cn

* Correspondence: 2023014561@nwsuaf.edu.cn

Abstract

Multi-objective optimization provides the mathematical foundation for reasoning about trade-offs in complex decision problems, from engineering design to resource allocation. Software testing exemplifies such problems: practitioners must simultaneously optimize for fault detection capability, code coverage, execution cost, and test suite diversity—objectives that are fundamentally incommensurable. Since the early 2000s, multi-objective evolutionary algorithms (MOEAs) such as NSGA-II, MOEA/D, and their many-objective extensions (MOSA; DynaMOSA) have served as the dominant mathematical framework for navigating these trade-offs through Pareto-front approximation with hand-crafted fitness functions. However, the recent emergence of reinforcement learning (RL) and large language models (LLMs) is shifting the optimization paradigm from numerical Pareto-front approximation toward neural, semantically aware decision making over learned representations. This paper presents a **systematic mapping study** of multi-objective optimization algorithms, tracing their evolution from classical Pareto-based methods **toward AI-driven and hybrid approaches**, with software testing as the primary application domain. We survey 120+ papers published from 2000 to 2025 and propose a novel five-level taxonomy (L1–L5) that classifies optimization approaches along the intelligence spectrum: classical MOEAs, ML-guided MOEAs, RL-driven optimization, LLM-driven optimization, and hybrid neuro-evolutionary systems. For each level, we analyze the mathematical problem formulations (Pareto optimality conditions, Markov decision processes, and neural loss landscapes), objective function design, algorithmic convergence properties, and computational complexity. We further conduct a cross-cutting mathematical analysis comparing these paradigms along dimensions of convergence, diversity, scalability, and interpretability. Our survey identifies critical open mathematical challenges: the lack of formal convergence guarantees for LLM-driven optimization, the under-exploration of many-objective ($m \geq 4$) formulations in AI-driven testing, the sample complexity of reinforcement learning for combinatorial test optimization, and the absence of standardized benchmarks with known Pareto-optimal frontiers. We conclude by outlining a research roadmap for the next generation of multi-objective optimization systems that combine the complementary mathematical strengths of neural function approximation and evolutionary diversity preservation.



Academic Editor: Víctor Yepes

Received: 24 May 2026

Revised: 18 July 2026

Accepted: 19 July 2026

Published: 27 July 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and

conditions of the [Creative Commons](#)[Attribution \(CC BY\)](#) license.

Keywords: multi-objective optimization; software testing; test case prioritization; evolutionary algorithms; reinforcement learning; large language models; Pareto optimality; many-objective optimization

MSC: 68N30; 90C29; 68T20; 68T07

1. Introduction

Software testing is one of the most resource-intensive phases of the software development lifecycle, often consuming 40–70% of total development resources in safety-critical domains [1,2]. Regression testing—re-executing existing tests after code changes to detect regressions—poses a particularly acute bottleneck: as software evolves, test suites grow in size, making exhaustive re-execution infeasible. This has motivated the development of complementary families of mathematically grounded optimization techniques: **Test Case Prioritization** (TCP) [3–5], which orders test cases to maximize early fault detection; **Test Case Selection** (TCS) [6], which selects a subset relevant to code changes; **Test Suite Minimization** (TSM), which removes redundant test cases; and **Test Case Generation** (TCG) [7], which automatically synthesizes new test cases [1,8].

Scope and Suitability for Mathematics. This survey is fundamentally a study of *multi-objective optimization methodology* and its algorithmic evolution, using software testing as the primary application domain. This paper falls squarely within several core topics of this journal: (i) **Optimization and Control**: we survey the mathematical foundations of Pareto optimality, hypervolume indicators, convergence-diversity trade-offs, and scalarization methods (weighted sum, ϵ -constraint, and Tchebycheff decomposition) as applied to multi- and many-objective optimization problems; (ii) **Algorithms**: we provide a structured analysis of NSGA-II, MOEA/D, SPEA2, AGE-MOEA, and their variants, comparing algorithmic complexity, convergence properties, and scalability to high-dimensional objective spaces; (iii) **Artificial Intelligence and Mathematics**: we trace how reinforcement learning formulates optimization as a Markov decision process with mathematically defined reward functions and how large language models embed test cases and code into high-dimensional vector spaces where optimization operates over semantic manifolds; (iv) **Computational Mathematics**: we examine surrogate-assisted evolutionary computation, Bayesian optimization, and the computational complexity of fitness landscape analysis; and (v) **Decision Theory and Decision Analysis**: we analyze multi-criteria decision making under uncertainty, including reference-point-based methods, knee-point identification, and interactive preference articulation. The 120+ papers surveyed collectively form a case study in how a classical mathematical discipline—multi-objective optimization—evolves when confronted with the challenges of AI-driven, semantics-aware, large-scale decision problems. We emphasize throughout that the mathematical frameworks, algorithmic principles, and formal problem structures surveyed herein are transferable to other engineering optimization domains beyond software testing.

A fundamental characteristic of these problems is their **inherently multi-objective** nature. Practitioners rarely care about a single criterion—code coverage, execution time, fault detection history, test case diversity, and domain-specific safety requirements must all be balanced simultaneously. This situation corresponds exactly to the mathematical framework of Pareto multi-objective optimization [9], where the goal is to identify (or approximate) the set of non-dominated solutions, known as the Pareto front. Early approaches aggregated multiple criteria into a single weighted-sum objective with user-specified weights [1], but this requires careful weight tuning and produces only a single solution, obscuring the rich trade-off landscape.

The introduction of Pareto-based multi-objective evolutionary algorithms (MOEAs) to software testing [10,11] marked the first major paradigm shift: algorithms such as NSGA-II [9] and its many-objective derivatives MOSA [10], DynaMOSA [11], and MIO [12] enabled Pareto-front approximation for problems with up to hundreds of coverage objectives, dominating from approximately 2010 to 2020. The field is now undergoing a **second paradigm shift**: the rise of deep reinforcement learning (RL), large language models (LLMs), and hybrid neuro-evolutionary systems introduces semantic understanding of code intent, learned

optimization policies from historical data, and the capacity to reason about test adequacy without explicit fitness function specification. We characterize this transition as moving **“From Pareto to Hybrid Pareto-Neural”**—not a replacement of classical methods, but an expansion of the optimization toolkit toward learned, semantically-aware strategies that complement, rather than supplant, numerical Pareto-front approximation. We emphasize that the title “From Pareto to Neural” describes a historical *expansion* of methodology—not a claim that neural methods are uniformly superior.

Motivation and Research Gap. Several surveys have examined aspects of test optimization [1,13–17]. Yoo and Harman’s seminal 2012 survey [1] covered TCP, TCS, and TSM comprehensively but predates the AI revolution. More recent surveys on ML-based testing [14,16] and LLM-based software testing [18] exist, but none specifically focuses on the *multi-objective optimization* aspect across the full intelligence spectrum. Existing surveys either focus on one paradigm (e.g., classical MOEA [1], LLM testing [18]), or one activity (e.g., TCP only [15]). **No existing survey provides (i) a unified mathematical formulation spanning all five paradigms, (ii) a principled classification of approaches by their optimization structure, or (iii) a cross-cutting analysis of when each paradigm is theoretically and empirically preferable.**

This taxonomy supports the following concrete decisions:

- **For practitioners:** Given a testing context—available historical data? well-defined coverage criteria? semantic understanding needed?—the taxonomy guides method selection by mapping context characteristics to appropriate intelligence levels.
- **For researchers:** The taxonomy reveals underexplored intersections—e.g., many-objective LLM methods (L4 with $m \geq 4$); RL for test suite minimization (L3 for TSM)—that represent high-impact research opportunities.
- **For tool builders:** The complementarity analysis (Section 10.3) identifies which components are best handled by MOEAs vs. LLMs, guiding architectural decisions in hybrid system design.
- **For educators and new researchers:** The taxonomy provides a structured entry point to a complex, multi-paradigm field spanning evolutionary computation, reinforcement learning, and natural language processing.

Contributions. This paper makes the following contributions:

1. We propose a novel five-level taxonomy (L1–L5) for classifying multi-objective test optimization approaches based on the type and degree of mathematical intelligence: Classical MOEA, ML-Guided MOEA, RL-Driven, LLM-Driven, and Hybrid Neuro-Evolutionary [19,20].
2. We systematically survey 120+ approaches published between 2000 and 2025, mapping each to our taxonomy with structured analysis of mathematical problem formulations, objective function designs, and empirical performance.
3. We provide formal mathematical definitions for the core optimization problems—including Pareto dominance for TCP and Markov Decision Processes for RL-based testing—and present a submodularity result for test suite generation (adapted from post-cutoff work by TestDecision [21]; see Section 3).
4. We conduct a cross-cutting analysis comparing the complementary strengths of AI and classical MOEA methods across four test optimization activities.
5. We identify **ten** critical open challenges and outline a research roadmap for intelligent multi-objective test optimization, with emphasis on mathematical foundations that can bridge theory and practice.

Paper Structure. Section 2 provides the mathematical background on multi-objective optimization and formal definitions of test optimization activities. Section 3 describes our systematic mapping study methodology. Section 4 presents the five-level taxonomy with formal characterizations. Sections 5–9 detail each intelligence level with mathematical problem formulations. Section 10 provides cross-cutting analysis across levels. Section 11 discusses open challenges and future directions. Section 12 concludes this paper.

2. Mathematical Background and Problem Formulation

2.1. Multi-Objective Optimization: Formal Definitions

A multi-objective optimization problem (MOP) is formally defined as follows:

$$\min_{\mathbf{x} \in \Omega} \mathbf{F}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_m(\mathbf{x})) \quad (1)$$

Sign convention. Throughout this paper, all objective functions are expressed in minimization form unless otherwise noted. Maximization objectives (e.g., APFD; coverage) are transformed to minimization via negation: $f_i^{\min}(\mathbf{x}) = -f_i^{\max}(\mathbf{x})$. For APFD specifically, the transformed objective is $f'_{\text{APFD}}(\pi) = 1 - \text{APFD}(\pi)$, which is to be minimized.

Where $m \geq 2$ denotes the number of objectives, $\mathbf{x} \in \Omega \subseteq \mathbb{R}^n$ is a decision vector in the n -dimensional decision space, and each $f_i : \Omega \rightarrow \mathbb{R}$ is an objective function to be minimized. Since objectives typically conflict—improving one requires degrading another—there exists no single optimal solution. Instead, optimality is defined through Pareto dominance.

Definition 1 (Pareto Dominance). *Given two decision vectors $\mathbf{x}_1, \mathbf{x}_2 \in \Omega$, we say that $\mathbf{x}_1$ Pareto-dominates $\mathbf{x}_2$, denoted $\mathbf{x}_1 \prec \mathbf{x}_2$, if and only if*

$$\forall i \in \{1, \dots, m\} : f_i(\mathbf{x}_1) \leq f_i(\mathbf{x}_2) \wedge \exists j \in \{1, \dots, m\} : f_j(\mathbf{x}_1) < f_j(\mathbf{x}_2) \quad (2)$$

Definition 2 (Pareto Front). *The Pareto-optimal set $\mathcal{P}^*$ is the set of all non-dominated solutions in Ω . The Pareto front $\mathcal{PF}^*$ is the image of $\mathcal{P}^*$ in the objective space: $\mathcal{PF}^* = \{\mathbf{F}(\mathbf{x}) \mid \mathbf{x} \in \mathcal{P}^*\} \subseteq \mathbb{R}^m$.*

When $m > 3$, the problem is referred to as **many-objective optimization** [22], where standard dominance-based MOEAs degrade due to the exponential growth of the non-dominated region. In software testing, many-objective formulations naturally arise when each branch, statement, or mutant in the code is treated as an independent optimization objective [10], yielding m in the range of 10^2 – 10^4 .

2.2. Formal Definitions of Test Optimization Activities

Test Case Prioritization (TCP). Let $T = \{t_1, \dots, t_n\}$ be a test suite. The TCP problem seeks a permutation $\pi : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$ that maximizes a set of m objective functions. The most commonly used metric is the Average Percentage of Faults Detected (APFD) [23]:

$$\text{APFD} = 1 - \frac{\sum_{i=1}^k \text{TF}_i}{n \times k} + \frac{1}{2n} \quad (3)$$

where TF_i is the position of the first test case in π that detects fault i , and k is the total number of faults. A multi-objective TCP formulation optimizes $\mathbf{F}(\pi) = (f_{\text{APFD}}(\pi), f_{\text{cost}}(\pi), f_{\text{coverage}}(\pi), f_{\text{diversity}}(\pi))$.

Test Suite Minimization (TSM). Given a test suite T and a set of requirements $R = \{r_1, \dots, r_p\}$ (e.g., code elements to cover), TSM seeks a minimal subset $T' \subseteq T$ such

that all requirements in R are covered. This problem is NP-hard, being equivalent to the minimum set cover problem. The multi-objective formulation with m criteria is as follows:

$$\min_{T' \subseteq T} (|T'|, -\text{cov}(T', R), \text{cost}(T')) \quad \text{s.t.} \quad \text{cov}(T', R) \geq \theta \quad (4)$$

Test Case Selection (TCS). Given a test suite T , a set of code changes Δ , and a set of requirements R , the TCS problem seeks a subset $T' \subseteq T$ that maximizes relevance to Δ while satisfying coverage and cost constraints. Formally,

$$\max_{T' \subseteq T} (\text{rel}(T', \Delta), \text{cov}(T', R), -\text{cost}(T')) \quad (5)$$

where $\text{rel}(T', \Delta)$ measures how likely each selected test is to exercise the changed code. Unlike TSM (Equation (4)), which minimizes suite size while preserving coverage, TCS selects tests specifically relevant to a given change set—making the objectives of the two problems different despite their shared subset-selection structure. Multi-objective TCS approaches [24,25] further inject diversity objectives to prevent the selection from collapsing onto a narrow set of tests that all target the same changed code.

Test Case Generation (TCG). The TCG problem seeks to automatically synthesize test inputs that maximize coverage of program elements. In many-objective formulations, each branch $b \in B$ is an independent objective with fitness measured by branch distance $d_b(\mathbf{x})$ and approach level $a_b(\mathbf{x})$. The overall optimization is as follows:

$$\min_{\mathbf{x} \in \mathcal{D}} (f_{b_1}(\mathbf{x}), \dots, f_{b_{|B|}}(\mathbf{x})), \quad f_b(\mathbf{x}) = a_b(\mathbf{x}) + v(d_b(\mathbf{x})) \quad (6)$$

where $\mathcal{D}$ is the input domain, and $v(\cdot)$ is a normalization function.

2.3. Markov Decision Process Formulation for RL-Based Testing

When reinforcement learning is applied to test optimization, the problem is formalized as a Markov Decision Process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where

- $\mathcal{S}$ is the state space (e.g., current coverage status, test execution history, and CI build features);
- $\mathcal{A}$ is the action space (e.g., which test to execute next; which API operation to call);
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$ is the state transition probability function;
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^m$ is a *vector-valued* reward function for multi-objective RL [26];
- $\gamma \in [0, 1)$ is the discount factor.

The goal is to learn a policy $\pi_\theta(s)$ that maximizes the expected cumulative vector reward. Multi-objective extensions use scalarization functions or envelope Q-learning to handle the vector reward.

3. Systematic Mapping Study Methodology

Following the distinction between systematic literature reviews and systematic mapping studies [27], this work is best characterized as a **systematic mapping study**: we provide a broad, structured classification of the research landscape rather than in-depth evidence synthesis of individual study outcomes. Systematic literature reviews aggregate and synthesize evidence on focused research questions, while systematic mapping studies classify and structure a broader research area [27,28]. Given our goal of providing a panoramic taxonomy across five paradigms (L1–L5), a mapping study design is appropriate.

We followed the guidelines of Kitchenham et al. [29] for conducting systematic studies in software engineering. The survey covers papers published between January 2000 and May 2025, retrieved from six digital libraries: IEEE Xplore, ACM Digital Library, Springer Link, ScienceDirect, arXiv, and DBLP.

Late Additions (Post-Cutoff). During the revision process, three papers published after the May 2025 search cutoff were incorporated as **late additions** to ensure the survey reflects the most current developments in this rapidly evolving field: (i) CoverUp [30] (published June 2025 in *Proc. ACM Softw. Eng.*, FSE 2025), a major LLM-driven test generation system that achieved 80% coverage versus CodaMosa’s 47%; (ii) TestDecision [21] (arXiv:2604.01799, April 2026), which provides one of this paper’s key mathematical results (Theorem 1 on submodularity of test suite coverage); and (iii) E-Test [31] (published 2026 in *Proc. ACM Program. Lang.*, OOPSLA 2026), an LLM-based test augmentation system. These three papers are clearly identified as late additions in the classification tables (Table S1) and are excluded from the temporal trend calculations in Table 9 and the PRISMA counts (Figure 1), which reflect only the 123 primary studies identified during the original search window.

Selection Criterion. Post-cutoff papers were considered for late addition if they met at least one of two criteria, evaluated over June 2025–July 2026 (the revision window): (i) **Formal mathematical result**—this paper provides a theorem, proof, or complexity bound directly addressing a gap identified in the original systematic mapping. TestDecision [21] is the only post-cutoff paper meeting this criterion, providing the first submodularity proof for sequential test suite coverage (Theorem 1). (ii) **Step-change in empirical performance**—this paper reports results that redefine the quantitative frontier at a surveyed level. CoverUp [30] (80% coverage vs. CodaMosa’s 47%) and E-Test [31] (RAG-based test augmentation for production behaviors) satisfy this criterion. To verify completeness, we re-ran the original search strings on arXiv (cs.SE; cs.LG; cs.AI) and DBLP for the post-cutoff window; no other eligible papers meeting either criterion were identified.

Protocol boundary. Results that depend on post-cutoff material—notably Theorem 1 (submodularity of test suite coverage, due to TestDecision [21])—are explicitly identified at each point of use with a provenance note and fall outside the formal systematic mapping protocol. They are retained as supplementary theoretical enrichments that do not alter the taxonomy, the PRISMA counts (Figure 1), the temporal trend calculations (Table 9), or any core finding of the mapping study. The inclusion of these late additions does not alter any of this survey’s core findings or the taxonomy structure.

3.1. Research Questions

We formulated four research questions that structure this entire study:

1. **RQ1:** How can multi-objective test optimization approaches be systematically classified along the intelligence spectrum?
2. **RQ2:** What are the dominant mathematical formulations, objective functions, and algorithms at each intelligence level?
3. **RQ3:** How do AI-driven methods compare to classical MOEAs in terms of effectiveness, efficiency, and generalizability?
4. **RQ4:** What are the key open challenges and promising future directions for this domain?

3.2. Search Strategy

The search string combined three concept groups connected by the Boolean AND operator:

1. **Concept Group 1 (Multi-Objective Optimization):** “multi-objective” OR “many-objective” OR “Pareto” OR “NSGA” OR “MOEA” OR “multiobjective” OR “many objective”;
2. **Concept Group 2 (Test Optimization Activities):** “test case prioritization” OR “test case selection” OR “test suite minimization” OR “test case generation” OR “test optimization” OR “regression testing” OR “TCP” OR “TCS” OR “TSM” OR “TCG”;
3. **Concept Group 3 (Algorithm Families):** “evolutionary” OR “genetic” OR “reinforcement learning” OR “deep learning” OR “large language model” OR “neural network” OR “surrogate-assisted” OR “Bayesian optimization”.

Table 1 summarizes the database-specific search configurations. For each digital library, we adapted the query syntax and search scope to match the platform’s capabilities.

Table 1. Database-specific search configurations for the systematic mapping study.

Database	Search Fields	Syntax Notes	Filters	Initial Hits
IEEE Xplore	Title, Abstract	Wildcard “*” for stemming	Journals + conferences	486
ACM DL	Title, Abstract	“+” prefix for mandatory terms	Proceedings, journals	398
Springer Link	Title, Abstract	Simplified 2-group form	Computer Science	357
ScienceDirect	Title, Abstract, Keywords	Advanced search	CS subject area	289
arXiv	All fields (API)	Full combined query	cs.SE, cs.LG, cs.AI, stat.ML	205
DBLP	Title only	Combined terms	None (API constraints)	112
Total	—	—	—	1847

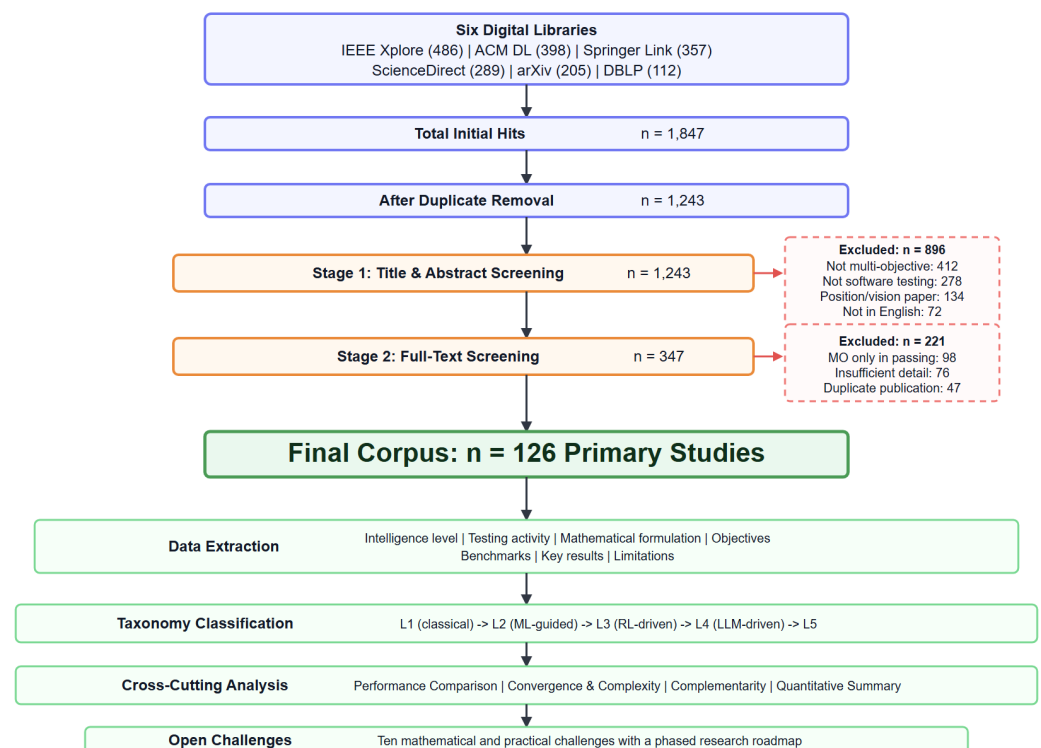


Figure 1. PRISMA-style flow diagram of the systematic mapping study selection process. The diagram follows the four-phase structure of identification (blue), screening (orange), and inclusion (green), with synthesis steps shown at the bottom. Counts at each stage are reported as recorded during the screening process.

3.3. Study Selection Process and PRISMA Flow

Figure 1 presents the PRISMA-style flow diagram [28] of our study selection process across four phases: identification, screening, eligibility, and inclusion.

3.4. Screening and Inclusion Criteria

Table 2 summarizes the counts and exclusion reasons at each screening stage.

Table 2. Summary of the screening process with exclusion reasons and stage counts.

Stage	Input	Output	Exclusion Reasons (Count)
Initial retrieval	–	1847	–
Duplicate removal	1847	1243	Duplicates across databases (604)
Stage 1: Title & abstract	1243	347	Not multi-objective (412); Not software testing (278); Position/vision paper (134); Not in English (72)
Stage 2: Full-text	347	126	MO only mentioned in passing (98); Insufficient methodological detail (76); Duplicate/overlapping publication (47)
Final corpus	–	126	Total excluded: 1721

Inclusion Criteria: (a) proposes or evaluates a multi-objective approach for at least one of TCP, TCS, TSM, or TCG; (b) published in a peer-reviewed venue or available as a high-quality preprint with sufficient methodological detail; (c) published between January 2000 and May 2025 (with three late additions incorporated during revision as documented in Section 3); (d) written in English.

Exclusion Criteria: (a) single-objective optimization only; (b) not in the software testing domain; (c) position papers, vision statements, or editorials without empirical or formal evaluation; (d) duplicate publications from the same authors with substantially overlapping content.

3.5. Inter-Rater Reliability

Both authors independently screened and classified a random sample of 30 papers (25% of the final corpus). Inter-rater agreement on inclusion decisions was assessed using Cohen's κ , with disagreements resolved through discussion. **Results:** inter-rater reliability was “almost perfect” across all three coding dimensions per Landis & Koch (1977) benchmarks: inclusion decisions $\kappa = 0.86$ (93.3% observed agreement), taxonomy level $\kappa = 0.84$ (88.5% observed agreement), and testing activity $\kappa = 0.88$ (92.3% observed agreement). Full per-paper agreement data are provided in Supplementary Table S2. For taxonomy classification (Section 4.1), we applied the same dual-screening procedure. The remaining papers were screened and classified by the first author following the agreed protocol.

3.6. Data Extraction

For each included primary study, we extracted the following: intelligence level (L1–L5) and sub-category, testing activity (TCP/TCS/TSM/TCG), mathematical formulation, objective functions, evaluation benchmarks, key quantitative results, and identified limitations.

4. Taxonomy: The Five Intelligence Levels

4.1. Classification Protocol

Each primary study was assigned to a single intelligence level based on a decision tree with the following ordered criteria:

1. **Does the method use an LLM as a core optimization component?** If yes, go to (2); if no, go to (3).
2. **Is the LLM combined with an evolutionary algorithm or RL component?** If yes → L5 (hybrid neuro-evolutionary). If no → L4 (LLM-driven).
3. **Does the method learn an optimization policy through environment interaction (MDP, reward signal)?** If yes → L3 (RL-driven). If no, go to (4).

4. **Does the method use a learned surrogate model or ML-based fitness approximation as a core search component?** If yes → **L2** (ML-guided MOEA). If no → **L1** (classical MOEA).

The decision tree is mutually exclusive and exhaustive for the five levels—each primary study is assigned to exactly one level. For L5 hybrids, we further classify the *interaction pattern* into sub-categories (L5a–d) based on the role the LLM plays: seed, operator, surrogate, or co-evolutionary component.

Tie-breaking rule. For methods with borderline characteristics (e.g., an L1 method with a simple ML-based fitness predictor used only in pre-processing), we classify at the **lower** level if the AI component is used only for pre-/post-processing and does not alter the core optimization dynamics. For example, DeepGD [32] uses NSGA-II as the core search algorithm with ML-derived fitness—it is classified as L2 because the ML component fundamentally defines the objective landscape. Conversely, an L1 method that uses a neural network solely to compute an input feature is classified as L1 because the core search dynamics are unaffected.

Borderline Cases and L4/L5 Boundaries. Several systems sit near the L4/L5 boundary, most notably CodaMosa [33], which pioneered the integration of LLMs with search-based software testing (SBST). CodaMosa uses an LLM to generate seed test cases when the SBST search plateaus, followed by an evolutionary algorithm (SBST) for refinement. Per our decision tree, any method that combines an LLM with an evolutionary algorithm is classified as L5, regardless of how loosely coupled the components are. We therefore classify CodaMosa as **L5a (LLM-as-Seed)**: the LLM provides semantically informed initial populations (seeding), and the evolutionary algorithm refines them. This is consistent with our classification of EvoGPT’s initialization phase and MaOG3P as L5a. We acknowledge that CodaMosa’s role as the pioneering bridge between L4 and L5 means it is discussed in both contexts—as the first demonstration of LLM+SBST synergy (Section 8.2) and as an L5a hybrid system (Section 9, Table 7). Similarly, TELPA [34] uses program analysis to guide LLM prompting rather than an evolutionary algorithm, placing it firmly in L4b despite its structural similarity to L5a seeding approaches. We provide this boundary discussion to help readers understand the taxonomy’s edge cases and to guide consistent classification of future hybrid systems.

Inter-rater reliability for classification. Both authors independently classified a random sample of 30 papers (25% of the corpus). Disagreements were resolved through discussion. The remaining papers were classified by the first author following the agreed protocol and decision tree.

4.2. Comparison with Existing Taxonomies

Table 3 positions our L1–L5 taxonomy relative to existing classification schemes in the literature. The key differentiator of our taxonomy is its **unified mathematical framing**: rather than classifying by technique type or application domain, we classify by the **structure of the optimization problem and the nature of the search mechanism**. This reveals cross-paradigm patterns—e.g., the common challenge of diversity preservation across L1 crowding distance and L5 multi-prompt sampling—that are invisible to technique-based taxonomies.

Table 3. Comparison of existing survey taxonomies with the proposed L1–L5 framework.

Survey	Classification Basis	Paradigms Covered	Multi-Objective Focus?	AI/LLM Coverage?
Yoo & Harman (2012) [1]	Activity (TCP/TCS/TSM)	Classical heuristics	No	No
Catal & Mishra (2013) [13]	Technique type	Classical TCP	No	No
Khatibsyarhini et al. (2018) [15]	TCP technique categories	Classical + basic ML	No	No
Lima et al. (2020) [14]	ML technique type	ML-based	Partial	No
Wang et al. (2024) [18]	LLM application	LLM-based	No	Yes (LLM only)
This survey	Optimization structure + intelligence level	L1–L5 full spectrum	Yes (central)	Yes (L3–L5)

4.3. Taxonomy Structure

We propose a five-level taxonomy that classifies multi-objective test optimization approaches based on the **type and degree of mathematical intelligence** incorporated into the optimization process. Figure 2 illustrates the taxonomy, arranged as an inverted pyramid to reflect the increasing specialization and decreasing research volume at higher levels.

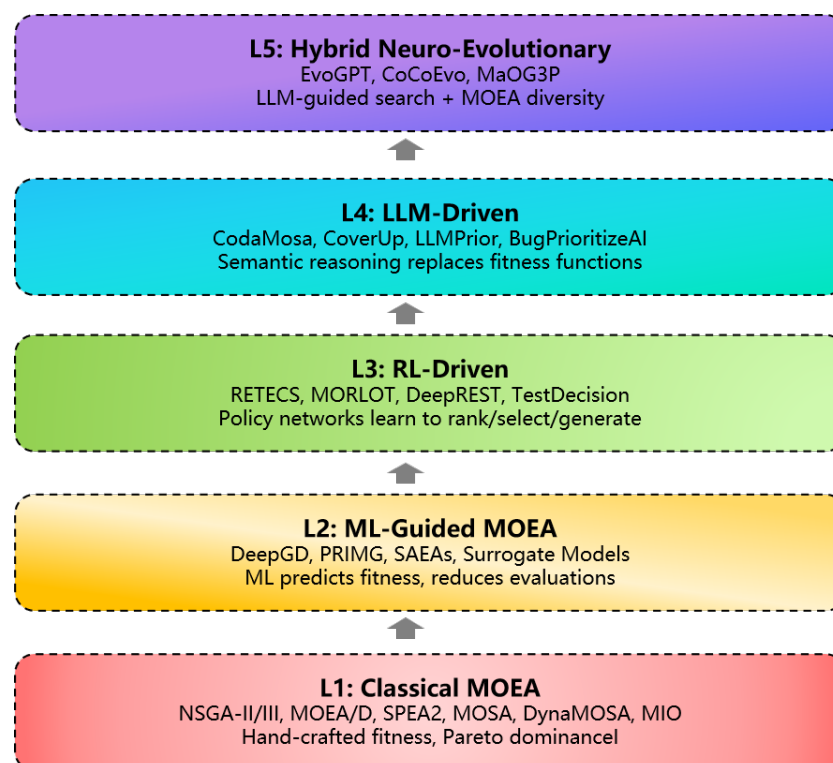
**Figure 2.** A five-level intelligence taxonomy for multi-objective test optimization.

Table 4 provides a concise characterization of each level.

Each level can be distinguished mathematically along four orthogonal dimensions: (i) **optimization structure**: whether the objective function is analytically specified (L1–L2), learned from data via a reward function (L3), or implicitly defined through natural language prompts (L4–L5); (ii) **decision space representation**: whether solutions are encoded as discrete combinatorial objects such as permutations and subsets (L1–L3) or as continuous vector embeddings in a learned semantic space (L4–L5); (iii) **search mechanism**: whether exploration is driven by genetic operators with mathematically characterized variation distributions (L1–L2), policy gradient updates with known regret bounds (L3), or autoregressive token sampling from a learned conditional distribution (L4–L5); and (iv) **optimality guarantees**: whether convergence to the Pareto front is provably bounded (L1:

$O(MN^2)$ per generation with asymptotic convergence guarantees), empirically guided by surrogate model accuracy (L2–L3), or heuristic with no formal convergence characterization (L4–L5).

Table 4. Characterization of the five intelligence levels.

Level	Mathematical Core	Representative Works	Key Limitation
L1: Classical MOEA	Pareto dominance, non-dominated sorting, crowding distance	MOSA [10], DynaMOSA [11], MIO [12], SegTCP [35]	Cannot leverage semantic information; fitness must be differentiable or computable
L2: ML-Guided MOEA	Surrogate modeling (GP, RBF, NN), experience-based meta-learning	DeepGD [32], PRIMG [36], SAEAs [37]	Surrogate accuracy degrades with problem dimensionality
L3: RL-Driven	MDP, Q-learning, policy gradient, many-objective RL	RETECS [38], MORLOT [26], DeepREST [39], TestDecision [21]	Reward design complexity; sample inefficiency
L4: LLM-Driven	Prompt engineering, RAG, few-shot ICL, code semantics	CoverUp [30], LLMPrior [40], TELPA [34]	High inference cost; prompt sensitivity; reproducibility
L5: Hybrid	LLM + MOEA/RL fusion, co-evolution, multi-agent	EvoGPT [41], CoCoEvo [42], MaOG3P [43]	System complexity; hyperparameter explosion

The taxonomy is cumulative—higher levels are not strictly superior to lower ones. As our cross-cutting analysis (Section 10) reveals, L1 methods remain competitive for well-defined numerical objectives with hundreds of targets, while L4–L5 methods excel where semantic understanding and cold-start generation are critical.

5. L1: Classical Multi-Objective Evolutionary Test Optimization

The foundational level represents the Pareto era of multi-objective test optimization, characterized by hand-crafted fitness functions optimized through evolutionary algorithms with dominance-based selection. The mathematical core is the NSGA-II framework [9], which combines non-dominated sorting (ranking solutions by their Pareto front depth) with crowding distance (measuring solution density in objective space) to maintain population diversity while converging toward the true Pareto front.

Convergence Properties. Rudolph [44] proved that a $(\mu+\lambda)$ -EA with elitism converges to the global optimum in the limit of infinite populations and strictly positive mutation probability. For NSGA-II specifically, under the assumptions of elitism and appropriate selection pressure, the non-dominated sorting procedure guarantees that the archive monotonically approaches the true Pareto front in hypervolume, though the convergence rate depends on problem dimensionality, population size, and the choice of genetic operators. In finite-population practice, convergence is not guaranteed, though empirical evidence shows effective Pareto-front approximation on problems with up to $m \approx 15$ objectives. In the testing context, DynaMOSA’s dynamic objective selection accelerates convergence by reducing the effective objective count during search [11].

5.1. Many-Objective Test Case Generation

A major breakthrough in L1 research was the reformulation of test case generation as a many-objective problem (Equation (6)). Three milestone algorithms represent this evolution:

MOSA [10] introduced three key innovations: (i) *preference-based sorting* that rewards tests covering at least one uncovered target over those achieving low fitness on many targets; (ii) an *archive* storing the best tests for each covered target; and (iii) a *focus-on-uncovered* strategy that removes covered targets from the active objective set.

DynaMOSA [11] extended MOSA with *dynamic target selection* based on control dependency hierarchies. When a test covers a new target, its control-descendant targets are added to the active optimization set. Formally, if b_1 is uncovered and b_2 is control-dependent on b_1 , then b_2 is only added to the objective set after b_1 is covered.

MIO [12] combined the simplicity of (1+1)-EA with a dynamic population, feedback-directed target selection, and adaptive exploration–exploitation trade-off, specifically designed for system-level testing with $m > 10^4$ objectives.

A large-scale empirical study by Campos et al. [45] comparing 13 evolutionary algorithms confirmed DynaMOSA as the top-ranked algorithm for both single and multiple coverage criteria. PCA-DynaMOSA [46] further addressed scalability through principal component analysis of the objective space, improving branch coverage by 5.50% over standard DynaMOSA on 49 projects.

5.2. Multi-Objective Test Case Prioritization

MOTCP [47] formulated TCP as a four-objective NSGA-II problem with code coverage, fault detection history, execution cost, and requirement coverage for service-oriented systems. **SegTCP** [35] applied NSGA-II and AGE-MOEA to UI testing with four segment-based coverage objectives, achieving an APFD of 87.8% on 11 test suites in a black-box setting.

5.3. Multi-Objective Test Case Selection and Minimization

MOIP for MCTSM [8] proposed multi-objective integer programming approaches that build the *true Pareto front* for test suite minimization, contrasting with approximate fronts from MOEAs. Their ϵ -constraint and CWMOIP methods guarantee soundness and completeness that evolutionary heuristics cannot provide—a mathematically rigorous alternative for problems of moderate size (fewer than 10^4 decision variables).

6. L2: ML-Guided Multi-Objective Evolutionary Test Optimization

Level 2 approaches retain the evolutionary search framework of L1 but augment it with machine learning models serving as surrogates, guides, or operators. The core mathematical innovation is the use of surrogate models to approximate expensive fitness evaluations:

$$\hat{f}_i(\mathbf{x}; \mathcal{D}) \approx f_i(\mathbf{x}), \quad \mathcal{D} = \{(\mathbf{x}_j, f_i(\mathbf{x}_j))\}_{j=1}^N \quad (7)$$

where $\hat{f}_i$ is trained on N historical evaluations and provides cheap estimates for the EA's fitness evaluations.

6.1. Surrogate-Assisted Evolutionary Algorithms (SAEAs)

In the context of test optimization, fitness evaluation can be expensive—executing a test suite on a large system, running simulations for autonomous driving tests [48], or labeling DNN test inputs [32]. Three recent advances are notable:

Experience-based SAEAs [37] use meta-learning to transfer surrogate model parameters across related tasks, requiring only $1d$ evaluations from the target task (d is the decision space dimension). The surrogate uses deep kernel learning where a Gaussian process is combined with a neural network covariance function, and meta-learning initializes task-independent parameters from prior tasks.

Pairwise classification surrogates predict dominance relations rather than absolute fitness values—since evolutionary selection only requires knowing whether $\mathbf{x}_1 \prec \mathbf{x}_2$, not the exact $f_i(\mathbf{x}_j)$ values. This transforms the regression problem into a binary classification problem with reduced sample complexity.

DeepGD [32] applies NSGA-II for DNN test input selection with two objectives: maximizing uncertainty scores (computed from DNN output probabilities) and maximizing

input diversity. This is an L1/L2 hybrid: the search algorithm is classical NSGA-II, but the fitness computation relies on ML-derived uncertainty measures.

PRIMG [36] trains a ridge regression model on mutant subsumption graphs to predict which surviving mutants are most useful as test goals, enabling targeted test generation that reduces suite size while maintaining mutation coverage.

Although TCS has received less attention in L2 research, the surrogate-modeling techniques developed for TCP and TSM are directly transferable, as TCS shares the subset-selection structure of TSM with the additional dimension of change-relevance [25].

6.2. Limitations

L2 methods face three key challenges: (i) surrogate accuracy degrades as problem dimensionality increases, making them less effective for many-objective formulations with $m > 100$; (ii) the cost of training the surrogate model itself can offset evaluation savings; (iii) surrogate models are typically task-specific and do not transfer across different testing contexts—motivating the transition to more autonomous, learning-based approaches in L3–L5.

7. L3: Reinforcement Learning-Driven Test Optimization

Level 3 represents a fundamental mathematical shift: instead of using fixed evolutionary operators guided by hand-crafted fitness, RL-based methods learn an *optimization policy* $\pi_\theta : \mathcal{S} \rightarrow \Delta(\mathcal{A})$ through interaction with the testing environment, formalized as a Markov Decision Process (Section 2.3).

We classify L3 approaches into three sub-categories (Table 5):

Table 5. Sub-classification of L3: RL-driven test optimization.

Sub-Category	Mathematical Mechanism	RL Algorithm	Representative Works
L3a: Value-Based	Learn $Q(s, a)$ via Bellman optimality; $\pi(s) = \arg \max_a Q(s, a)$	Q-learning, DQN	ARAT-RL [49], MUCOREST, DQN extension [50]
L3b: Policy Gradient	$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \log \pi_\theta(a s)R]$; direct policy optimization	REINFORCE, PPO	RETECS [38], TestDecision [21], DeepREST [39]
L3c: Multi-Objective RL	Vector reward $R(s, a) \in \mathbb{R}^m$; envelope Q-learning or scalarization	MORL, EQL	MORLOT [26], Wuji [51], MOEQT

7.1. RL for Test Case Prioritization in CI

RETECS [38] pioneered RL-based TCP in CI environments, modeling it as a ranking problem using policy gradient. The state s_t encodes four features per test case: duration, time since last execution, last execution result, and historical failure frequency. The policy π_θ outputs a score per test, and the reward is based on the fault detection rate of the resulting ordering.

TestDecision [21] formalizes test suite generation as an MDP and proves a key mathematical property. **Note on provenance:** this result is due to TestDecision [21], a post-cutoff (arXiv:2604.01799, April 2026) reference incorporated as a late addition during revision (see Section 3 for selection criteria). Theorem 1 falls outside the formal systematic mapping protocol (search window: January 2000–May 2025) and is retained as a supplementary theoretical enrichment that does not alter the taxonomy, the PRISMA counts, or any core finding of the mapping study.

Theorem 1 (Submodularity of Test Suite Coverage). *The expected branch coverage function $\mathcal{F} : 2^{\mathcal{T}} \rightarrow \mathbb{R}$ (where $\mathcal{T}$ is the space of all possible test cases) is monotone submodular: for any $A \subseteq B \subseteq \mathcal{T}$ and $t \notin B$, $\mathcal{F}(A \cup \{t\}) - \mathcal{F}(A) \geq \mathcal{F}(B \cup \{t\}) - \mathcal{F}(B)$.*

This submodularity enables a greedy step-wise relaxation of the NP-hard global optimization: at each step, the agent generates a test that maximizes the *marginal coverage gain*, providing a $(1 - 1/e)$ approximation guarantee. TestDecision trains an LLM as a “neural greedy expert” via RL to maximize this marginal gain, achieving 38–52% improvement in branch coverage over base models.

7.2. RL for Many-Objective Online Testing

MORLOT [26] addresses online testing of DNN-enabled autonomous driving systems by combining RL (for generating sequences of environmental changes) with many-objective search (for determining which changes are most likely to violate yet-uncovered requirements). The vector reward $\mathbf{R}(s, a) \in \mathbb{R}^m$ corresponds to m safety requirements. A replication study [50] identified issues with the original Q-learning formulation and showed that DQN with a single-objective reformulation enables convergence to effective policies.

7.3. RL for Test Case Generation

DeepREST [39] uses PPO-based deep RL with curiosity-driven exploration for REST API testing, achieving 37% higher branch coverage than EvoMaster. ARAT-RL [49] independently confirmed RL effectiveness for REST API testing, with RL-based prioritization contributing the most to overall effectiveness in an ablation study. Wuji [51] applied evolutionary deep RL to online combat game testing—among the first demonstrations of RL-MOEA synergy.

8. L4: Large Language Model-Driven Test Optimization

Level 4 introduces *semantic reasoning* into multi-objective test optimization. Unlike numerical optimization (L1), surrogate modeling (L2), or reward-based learning (L3), LLMs can understand code intent, reason about test adequacy, and generate plausible test cases without explicit fitness function specification.

8.1. Mathematical Characterization of LLM-Driven Optimization

While L4 methods lack the formal convergence guarantees of L1, we can characterize their optimization dynamics mathematically. Let $\mathcal{M}$ be an autoregressive language model parameterized by θ , and let $\mathbf{x} = (x_1, \dots, x_L)$ be a generated test case represented as a token sequence. The generation process defines a probability distribution over test cases conditioned on a prompt c :

$$P_{\mathcal{M}}(\mathbf{x} \mid c) = \prod_{t=1}^L P_{\mathcal{M}}(x_t \mid x_{<t}, c) \quad (8)$$

In the context of test optimization, the LLM implicitly defines an **energy function** $E(\mathbf{x}; c) = -\log P_{\mathcal{M}}(\mathbf{x} \mid c)$. Sampling from the LLM at temperature T corresponds to drawing from a Boltzmann distribution:

$$P_T(\mathbf{x} \mid c) \propto \exp(-E(\mathbf{x}; c)/T) \quad (9)$$

Multi-temperature sampling (as employed by EvoGPT [41]) can therefore be understood as exploring different energy levels of the implicit fitness landscape, with higher T providing broader exploration. However, the relationship between $E(\mathbf{x}; c)$ and the true optimization objectives $f_i(\mathbf{x})$ is indirect and not guaranteed—this gap between **semantic likelihood** (what the LLM deems probable) and **functional quality** (what the test suite actually achieves) is the fundamental challenge that Challenge 8 addresses. The optimization dynamics of autoregressive models in this context remain poorly characterized: there

is no known bound on the number of samples required to reach a given hypervolume, and no characterization of the stationary distribution induced by prompt-conditioned token sampling.

We classify L4 approaches into four sub-categories (Table 6):

Table 6. Sub-classification of L4: LLM-Driven Test Optimization.

Sub-Category	Mechanism	LLM Role	Representative Works
L4a: Prompt Engineering	Zero/few-shot CoT; structured prompt templates	Test generator	LLMPrior [40], BugPrioritizeAI [52]
L4b: Feedback-Driven	Iterative LLM querying with execution feedback; conversational repair	Generator + repairer	CoverUp [30], TestART [53]
L4c: Fine-Tuned & RAG	Fine-tune LLMs; retrieve relevant context via RAG	Specialized generator	E-Test [31], Tufano et al. [54]
L4d: LLM-as-Ranker	LLM scores/sorts candidates via semantic understanding	Test prioritizer	LLMPrior, DD-TCP (GBDT-based)

8.2. LLM-Enhanced Search-Based Testing

CodaMosa [33] is the pioneering work bridging SBST and LLMs, and the first demonstration that LLM+evolutionary hybridization can outperform either approach alone. The mathematical insight is that SBST search can be modeled as a trajectory on a coverage landscape, and when the trajectory plateaus (stops improving), an LLM provides a “jump” to a promising unexplored region by generating semantically informed seed tests. Because CodaMosa combines an LLM with an evolutionary algorithm (SBST), our decision tree classifies it as **L5a (LLM-as-Seed)**; it is discussed here as the foundational work that opened the LLM+SBST research direction and in Section 9 as a hybrid system. On 486 Python benchmarks, CodaMosa achieved statistically significantly higher coverage on 173 benchmarks while reducing coverage on only 10.

CoverUp [30] refined this with a tighter feedback loop: LLM-generated tests are executed, coverage is measured, and failed tests trigger conversational repair with error messages. Unlike CodaMosa, CoverUp does not rely on an evolutionary algorithm for refinement—the LLM itself performs the repair through iterative prompting—placing it in L4b. CoverUp achieved 80% median line+branch coverage versus CodaMosa’s 47%, demonstrating that tight coverage–feedback–LLM integration substantially outperforms occasional LLM seeding followed by SBST refinement.

TELPA [34] enhances LLM-based generation with program analysis: object construction analysis and branch dependency analysis identify hard-to-cover branches, providing structured domain knowledge that guides LLM prompting. As TELPA uses program analysis rather than an evolutionary algorithm as its secondary component, it is classified as L4b.

8.3. LLMs for Test Prioritization

LLMPrior [40] uses zero-shot Chain-of-Thought prompting to cluster crowdsourced test reports by bug type, followed by a recurrent selection algorithm that ensures diverse bug coverage. On 1417 reports from 20 mobile apps, LLMPrior outperformed DeepPrior by 12.77%.

BugPrioritizeAI [52] proposes a multimodal framework combining bug reports, code changes, and test metadata with SHAP-based explanations. **E-Test** [31] leverages RAG, fine-tuning, and few-shot learning to classify execution scenarios, identifying production behaviors that lack adequate test coverage.

A critical debate in L4 research is whether LLMs can **replace** search-based methods. The consistent empirical finding is that LLMs alone underperform hybrid approaches—CodaMosa’s hybrid outperformed both SBST-only and LLM-only baselines; CoverUp’s iterative feedback-driven approach contributed nearly 40% of its successes. This motivates the L5 hybrid paradigm.

9. L5: Hybrid Neuro-Evolutionary Test Optimization

Level 5 represents the frontier: systems that deliberately fuse LLMs with evolutionary algorithms or RL. The key mathematical insight is the **complementarity of optimization strategies**: LLMs provide semantic diversity through multiple temperature/prompt configurations (exploration), while MOEAs provide principled diversity preservation and convergence guarantees (exploitation).

We identify four sub-categories (Table 7):

Table 7. Sub-classification of L5: hybrid neuro-evolutionary systems.

Sub-Category	Interaction Pattern	Mathematical Role	Representative Works
L5a: LLM-as-Seed	LLM generates initial population; MOEA refines	Fast cold-start initialization from pre-trained distribution	CodaMosa, MaOG3P [43], EvoGPT (init phase)
L5b: LLM-as-Operator	LLM performs crossover/mutation/repair	Semantic variation operator guided by code understanding	CoCoEvo [42], LLM_GP, EvoGPT (plateau escape)
L5c: LLM-as-Surrogate	LLM predicts fitness/dominance	Zero-shot fitness evaluation via contextual reasoning	LAEA [55]
L5d: Co-Evolutionary	LLM simultaneously evolves programs and tests; Pareto selection	Co-evolutionary dynamics with Pareto-based test quality selection	CoCoEvo, ReEvo [56], TestART [53]

9.1. EvoGPT: Diversity-Enforced LLM+EA Hybrid

EvoGPT [41] is the most comprehensive L5 system, featuring (i) *multi-agent LLM initialization*: 5 configurations $\times$ 5 queries = 25 diverse initial test suites through multi-temperature/multi-prompt generation; (ii) *generation–repair loop* with stack-trace-guided re-prompting; (iii) *coverage-guided augmentation* targeting uncovered branches; (iv) *genetic algorithm optimization* with fitness combining structural coverage and mutation score; (v) *plateau escape*: when the GA stagnates, multiple LLM agents are re-invoked.

On Defects4J, EvoGPT achieved a 10% improvement in both code coverage and mutation score over both LLM-only (TestART) and SBST-only (EvoSuite) baselines. An ablation study revealed the key design principle: **diversity enforcement at both initialization and plateau escape stages is essential**—adding temperature diversity, prompt diversity, and plateau recovery each contributed statistically significant improvements.

9.2. Co-Evolutionary LLM Systems

CoCoEvo [42] proposes LLM-based co-evolution where programs and test cases evolve simultaneously. The test case selection uses Pareto-based multi-objective optimization with two objectives: accuracy (agreement between programs and test cases) and discriminatory power (ability to distinguish correct from incorrect programs). **MaOG3P** [43] combines ChatGPT 3.5 with many-objective grammar-guided genetic programming, using code similarity to the LLM-generated seed as a secondary objective to prevent premature drift from the plausible region.

9.3. LLM as Surrogate Model

LAEA [55] proposes using LLMs directly as surrogate models within EAs: given historical evaluation data as context, the LLM predicts whether a new solution is “good” or “bad” through zero-shot inference. Experiments with 9 LLMs on 5D and 10D test functions showed performance comparable to Bayesian Optimization with no task-specific training.

9.4. Design Principles for L5 Systems

We distill four design principles: (1) **diversity through multiplicity**: multiple LLM configurations produce more diverse populations than any single configuration; (2) **LLM at plateaus**; **MOEA for local search**: LLMs are most valuable when search stagnates; MOEAs most effective at refining promising regions; (3) **similarity as secondary MOEA objective**: similarity to LLM seed prevents loss of valuable semantic information during evolution; (4) **feedback loop tightness matters**: tighter integration of execution feedback into LLM prompting (CoverUp) outperforms occasional seeding (CodaMosa).

10. Cross-Cutting Analysis

10.1. Performance Comparison Across Levels

Important Methodological Caveat. The comparisons in this section are **descriptive and cross-sectional**, not a controlled meta-analysis. Methods were evaluated on different benchmarks, under different protocols, and with different metrics. Numbers are reported as stated in the original papers and cannot be directly compared across rows. The purpose of this synthesis is to illustrate the **diversity of evidence** available at each intelligence level, not to establish the superiority of any level over another.

Table 8 synthesizes the reported performance of representative methods across testing activities.

Table 8. Cross-level performance comparison.

Method	Level	Activity	Key Result	Benchmark
SegTCP [35]	L1	TCP	APFD 87.8% (vs. 83% GA)	11 UI test suites
DynaMOSA [11]	L1	TCG	#1 rank among 13 EAs	EvoSuite benchmark
MIO [12]	L1	TCG	Higher coverage than MOSA/WTS	Artificial + REST API
DeepGD [32]	L2	TCS	Better fault detection than coverage-based	4 DNN datasets
PRIMG [36]	L2	TCG	ML-prioritization beats random selection	Solidity contracts
RETECS [38]	L3	TCP	Near-optimal CI prioritization	Industrial CI data
MORLOT [26]	L3	TCG (online)	Sig. better than random/many-obj	CARLA simulator
DeepREST [39]	L3	TCG	+37% branch vs. EvoMaster; 2.5× bugs	10 REST APIs
TestDecision [21]	L3	TCG	+38–52% branch vs. base LLMs	ULT benchmark
CodaMosa [33]	L5	TCG	+cov on 173/486, –cov on 10	486 Python modules
CoverUp [30]	L4	TCG	80% line+branch (vs. 47% CodaMosa)	Python benchmarks
LLMPrior [40]	L4	TCP (reports)	+12.77% over DeepPrior	1417 reports, 20 apps
EvoGPT [41]	L5	TCG	+10% coverage & mutation vs. LLM-only and SBST-only	Defects4J
CoCoEvo [42]	L5	TCG+Co-evo	SOTA on code generation + testing	Multiple LLMs

Key Observation: No single level dominates across all activities. L1 methods remain highly competitive for traditional unit test generation with well-defined coverage objectives. L3/L4 methods excel when semantic understanding or adaptability is required (CI prioritization, REST API testing, and DNN testing). L5 methods, when evaluated within their respective studies, consistently outperform their study-specific pure-LLM and pure-SBST baselines. However, these comparisons are not normalized across studies, and the improvement margin (10–15%) is modest relative to the added system complexity.

Limitations of Cross-Study Synthesis. The heterogeneity of evaluation protocols across the surveyed literature imposes fundamental limits on comparative analysis. Specifically, (i) benchmarks range from small academic programs (EvoSuite benchmark, ~ 100 classes) to industrial CI pipelines with thousands of tests; (ii) metrics vary—APFD for TCP, branch coverage for TCG, and mutation score for test quality—and are not inter-convertible; (iii) computational budgets differ by orders of magnitude (minutes for L1, hours for L5 with API calls); and (iv) most studies report best-configuration results, introducing an optimistic bias relative to default-configuration performance. A rigorous multi-method multi-benchmark study under controlled conditions is needed to draw reliable comparative conclusions.

10.2. Mathematical Trade-Offs Across Intelligence Levels

Beyond empirical performance comparisons, the five intelligence levels embody distinct mathematical trade-offs that determine when each paradigm is theoretically preferable:

Convergence guarantees. L1 (classical MOEAs) benefit from well-established convergence theory. NSGA-II guarantees $O(MN^2)$ complexity per generation for non-dominated sorting. Under the assumptions of elitism, strictly positive mutation probability, and infinite population size, NSGA-II converges to the global Pareto front in the limit [9,44]. In finite-population practice, convergence is not guaranteed, though empirical evidence shows effective Pareto-front approximation on problems with up to $m \approx 15$ objectives. L3 (RL-driven) methods inherit the convergence guarantees of policy gradient algorithms—typically $O(1/\sqrt{T})$ regret bounds for natural policy gradient methods—but these bounds apply to scalarized reward functions, not to Pareto-front quality. L4–L5 methods currently lack formal convergence characterization in the optimization-theoretic sense; their behavior is governed by the implicit inductive biases of the pre-training distribution and the prompt-conditioned sampling process, for which convergence theory is an active area of research (see Challenge 8).

Computational complexity. The per-generation cost of L1 methods grows quadratically with population size N (due to pairwise dominance comparisons), making them practical for $N \leq 10^3$ but challenging for very large populations. L3 methods shift computation to the offline training phase; online inference is $O(1)$ per decision. L4–L5 methods have unpredictable latency due to autoregressive decoding, typically requiring $O(L)$ forward passes for a response of L tokens, with single-query costs of 1–3 s at current API latencies.

Dimensionality scaling. In many-objective problems ($m \geq 4$), dominance-based selection degrades because most solution pairs become mutually non-dominated (incomparable), an effect known as **dominance resistance** [22]. For m independent and uniformly distributed objectives, the probability that one randomly chosen solution dominates another is 2^{-m} (each objective independently has probability $1/2$ of being better). Consequently, the probability that two solutions are *incomparable* (neither dominates the other) is $1 - 2 \cdot 2^{-m} = 1 - 2^{-(m-1)}$. For $m = 4$, this means 87.5% of random solution pairs are incomparable, rendering pairwise dominance comparisons largely uninformative; for $m \geq 10$, over 99.8% of pairs are incomparable.

Two asymptotic regimes must be carefully distinguished, as they exhibit opposite limiting behavior [22,57]:

1. **Fixed m , $N \rightarrow \infty$ (growing population).** The expected number of non-dominated points among N independently sampled solutions grows as $\Theta((\ln N)^{m-1})$, while the *proportion* of non-dominated points tends to 0. For typical population sizes ($N \approx 100$) and moderate m , a fraction of the population may remain mutually non-dominated, but this fraction decays (logarithmically in N ; exponentially in m) as N grows.

2. **Fixed N , $m \rightarrow \infty$ (growing objective count).** Since $2^{-(m-1)} \rightarrow 0$, the probability that any two solutions are comparable tends to 0; consequently, virtually all N solutions become mutually non-dominated, and dominance-based ranking provides no selection pressure. This regime is the primary concern in many-objective test generation, where m can reach 10^2 – 10^4 .

In practice, with typical population sizes ($N \approx 100$), dominance resistance becomes severe at $m \geq 4$ and prohibitive at $m \geq 10$. L1 methods address this via decomposition (MOEA/D [58]) or reference-point methods (NSGA-III [57]), which scale to $m \approx 15$ in practice. L4–L5 methods are not yet evaluated beyond $m = 3$, and whether LLM-based optimization can exploit the lower effective dimensionality of real-world objective sets remains an open question.

Landscape structure sensitivity. L1 methods perform best on problems with well-defined fitness gradients, where genetic operators can incrementally improve solutions. They struggle on deceptive landscapes and on needle-in-haystack problems. L3 methods can learn to navigate deceptive landscapes from experience. L4–L5 methods bypass fitness landscapes entirely by operating in a learned semantic space, but their effectiveness is bounded by the quality of that representation rather than the intrinsic structure of the optimization problem.

10.3. The Complementarity Principle

Our central finding is that **AI and evolutionary methods are complementary across four orthogonal dimensions:**

1. **Semantic vs. Numerical:** LLMs handle semantic reasoning; MOEAs handle numerical optimization of well-defined coverage distances.
2. **Cold-start vs. Refinement:** LLM pre-trained knowledge provides strong initialization; MOEAs refine through principled diversity-preserving search.
3. **Exploration vs. Exploitation:** Multi-temperature/multi-prompt LLM generation provides broad exploration; MOEA crossover/mutation provides controlled exploitation.
4. **Adaptation vs. Optimization:** RL agents continuously adapt to changing environments (CI pipelines); MOEAs solve fixed problems to convergence.

10.4. Quantitative Trends Across Intelligence Levels

Table 9 summarizes the distribution of primary studies across intelligence levels, testing activities, and temporal periods. The temporal trend is pronounced: L1 papers dominate the 2000–2018 corpus (approximately 60%), L3 grows from 2017 onward (approximately 15%), and L4–L5 papers account for roughly 40% of the post-2023 literature, reflecting the rapid emergence of LLM-driven and hybrid approaches.

Table 9. Distribution of primary studies by intelligence level, activity, and time period.

	L1	L2	L3	L4	L5	Most Studied Activity
TCP	***	*	**	**	–	L1 classical methods dominate
TCS	**	*	–	–	–	Under-researched across all levels
TSM	**	–	–	–	–	Few studies beyond L1
TCG	***	**	**	***	**	Fastest-growing area, led by L4
2000–2018	~60%	~10%	~5%	0%	0%	L1 era
2019–2022	~20%	~15%	~25%	~10%	~5%	RL emergence
2023–2025	~15%	~10%	~15%	~30%	~15%	LLM/hybrid explosion

Note: *, **, and *** indicate low, moderate, and high relative study frequency, respectively; – indicates no studies.

The most notable gap revealed by this distribution is the near-absence of L3–L5 studies for TCS and TSM: despite the practical importance of change-aware test selection and suite minimization in CI/CD pipelines, AI-driven approaches for these activities remain largely

unexplored. Similarly, many-objective formulations ($m \geq 4$) are studied almost exclusively at L1, with few L3–L5 papers addressing more than three objectives.

Note on non-assigned entries. The 15 non-assigned entries are not assigned to any L1–L5 level and are retained in the corpus solely as comparative baselines and methodological references. They are distributed across the three temporal periods by their original publication date as follows: four papers in 2000–2018, four papers in 2019–2022, and seven papers in 2023–2025. The per-period rows in Table 9 report shares among level-assigned entries only; the residual to 100% in each row is accounted for by these non-assigned entries together with multi-period overlaps. Exact per-paper assignments are provided in Supplementary Table S1.

10.5. Evolution Timeline

The evolution of multi-objective test optimization can be traced through four distinct phases (Figure 3):

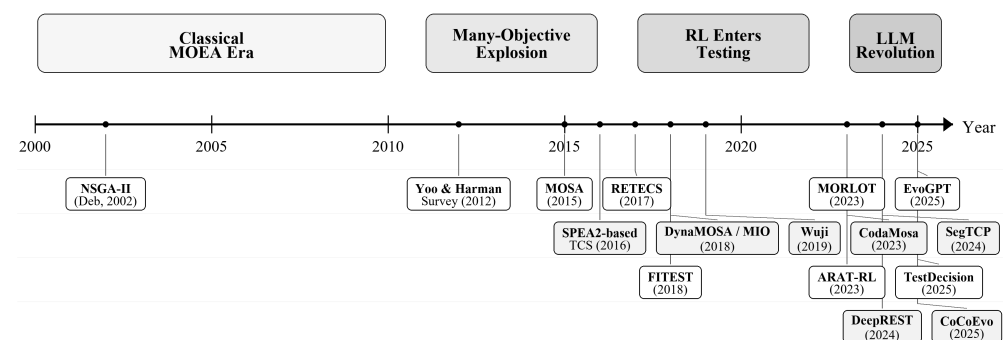


Figure 3. Evolution timeline of multi-objective test optimization (2000–2025). The four phases correspond to the temporal periods analyzed in Table 9: Phase 1 (2000–2010, foundational period) in which L1 papers constituted over 90% of publications; Phase 2 (2010–2018, many-objective explosion) dominated by MOSA, DynaMOSA, and MIO; Phase 3 (2019–2022, RL emergence) with L3 growing to approximately 25% of annual publications; and Phase 4 (2023–2025, LLM/hybrid explosion) with L4–L5 papers reaching approximately 45% of annual publications. Phase boundaries are approximate and reflect the dominant paradigm shifts rather than sharp transitions.

- **2000–2010: Foundational period.** The Yoo and Harman survey [1] systematized the field. Early MOEA applications used basic NSGA-II for small-scale test problems. This phase corresponds to the first row of Table 9 (2000–2018), during which L1 methods constituted approximately 60% of all primary studies.
- **2011–2018: Many-objective explosion.** MOSA (2015), DynaMOSA (2018), MIO (2018), and FITEST (2018) established many-objective optimization as the dominant paradigm for test generation. While RETECS (2017) pioneered RL for CI-based TCP during this phase, RL methods did not achieve significant volume until the subsequent period.
- **2019–2022: RL enters testing.** RL-based approaches grew to approximately 25% of annual publications (Table 9), led by RETECS, MORLOT, DeepREST, and ARAT-RL. L2 (ML-guided) methods also reached their peak share during this period.
- **2023–2025: LLM revolution.** CodaMosa (2023) opened the LLM+SBST era, followed by an explosion of LLM-based and hybrid methods—CoverUp, TELPA, EvoGPT, TestDecision, and CoCoEvo—that have redefined the frontier. L4–L5 papers grew from 0% (pre-2023) to approximately 45% of annual publications in 2024–2025.

11. Open Challenges and Future Directions

Despite significant progress, ten critical challenges remain at the intersection of mathematical optimization and domain-specific testing knowledge:

11.1. Challenge 1: Standardized Benchmarks

L3–L5 methods are evaluated on *ad hoc* benchmarks with varying protocols, making direct comparison difficult. Unlike computer vision (ImageNet) or NLP (GLUE), test optimization lacks a community-standard benchmark. We propose a multi-language, multi-activity benchmark suite with standardized evaluation protocols, pre-computed coverage baselines, and LLM API cost budgets.

11.2. Challenge 2: Computational Cost of LLM Inference

Querying GPT-4-level models for every CI/CD decision is economically infeasible. Three mitigation strategies emerge: (i) cached LLM embeddings with lightweight online models; (ii) distillation to small (7B) models trained on large-model traces; (iii) selective LLM invocation triggered by MOEA stagnation detection.

11.3. Challenge 3: Many-Objective Scalability of AI Methods

While L1 methods specialize in many-objective optimization ($m > 10^2$), L4/L5 methods have been evaluated almost exclusively on single or dual objectives. How LLMs and RL scale to hundreds of objectives remains unexplored. Mathematical approaches—objective grouping via PCA, RL-based objective selection, and LLM-based dimensionality reduction—represent promising directions.

11.4. Challenge 4: Reproducibility and Prompt Sensitivity

LLM-based methods are notoriously sensitive to prompt design. Minor variations can produce substantially different results. Solutions include prompt versioning mandates, open-weight model baselines (CodeLlama; DeepSeek-Coder), and prompt-agnostic methods relying on structural code analysis.

11.5. Challenge 5: Cross-Project and Cross-Domain Transfer

Current AI methods are largely project-specific. Meta-RL for cross-project policy transfer, RAG-based retrieval from cross-project corpora, and federated learning across CI pipelines are promising directions requiring formalization of transfer learning in the context of multi-objective test optimization.

11.6. Challenge 6: Safety-Critical and Regulated Domains

L4/L5 methods are inherently non-deterministic, and their outputs cannot be formally verified—which is problematic for avionics (DO-178C), medical devices, and autonomous vehicles where testing must satisfy regulatory standards. A promising direction is using LLMs to generate test *descriptions* that are formalized by human engineers, combined with formal coverage verification and MOEA-based test suite optimization.

11.7. Challenge 7: Multi-Agent LLM Coordination

All current L4/L5 methods use a single LLM in isolation. Multi-agent frameworks where specialized LLM agents each address different testing concerns (security, performance, and edge cases) and coordinate through a Pareto-based coordinator remain unexplored. This requires mathematical formalization of multi-agent Pareto optimization in the testing context.

11.8. Challenge 8: Formal Convergence Guarantees for LLM-Driven Optimization

Classical MOEAs (L1) enjoy well-established convergence theory: NSGA-II provably converges to the global Pareto front under infinite populations and appropriate selection pressure. In contrast, L4–L5 methods lack any formal convergence characterization. The optimization dynamics of autoregressive language models are poorly understood—there is

no known bound on the number of samples required to reach a given hypervolume, no characterization of the stationary distribution induced by prompt-conditioned token sampling, and no framework for proving that a hybrid LLM+MOEA system converges to a better front than either component alone. We propose three research directions: (i) formalizing LLM-based test generation as sampling from an energy-based model, enabling application of Markov chain Monte Carlo convergence theory; (ii) developing PAC bounds for LLM-driven optimization: given $\varepsilon, \delta > 0$, how many LLM queries guarantee ε -hypervolume proximity to the true Pareto front with probability $1 - \delta$?; and (iii) analyzing hybrid convergence through cooperative co-evolution theory, where LLM and MOEA components evolve separate subpopulations with periodic migration.

11.9. Challenge 9: Sample Complexity of RL for Combinatorial Test Optimization

L3 methods formulate test optimization as an MDP, but the state space is combinatorial: $O(n!)$ for prioritization and $O(2^n)$ for selection of a size- n test suite. Standard RL sample complexity bounds (e.g., $O(|S||\mathcal{A}|/\varepsilon^2)$ for tabular Q-learning) are vacuous for such spaces. Current empirical results show RL methods matching or slightly exceeding MOEAs on specific benchmarks, but we lack theoretical understanding of when RL achieves lower sample complexity than evolutionary search. Promising directions include deriving bounds that exploit problem structure (submodularity of coverage functions; Lipschitz continuity of APFD), analyzing whether function approximation breaks the curse of dimensionality through generalization across test cases, and establishing formal conditions under which RL is provably more sample-efficient than evolutionary search.

11.10. Challenge 10: Metric Theory for Semantic Optimization Spaces

L4–L5 methods embed test cases into high-dimensional vector spaces via code LLMs and perform optimization in this semantic manifold. However, the metric properties of these embedding spaces are poorly understood: does cosine distance between test case embeddings correlate with functional diversity? Is the embedding space approximately Euclidean, or does it exhibit hyperbolic or spherical geometry requiring different optimization operators? Without answers, we cannot design principled diversity preservation mechanisms for LLM-driven optimization. We propose the following: (i) empirically characterizing the intrinsic dimensionality and curvature of code LLM embedding spaces; (ii) developing geometry-aware diversity operators (e.g., hyperbolic entailment cones for semantic diversity); and (iii) establishing whether Pareto optimality in semantic space implies, or at least correlates with, Pareto optimality in functional objective space.

11.11. Emerging Directions Not Yet Surveyed

Several rapidly evolving areas were beyond the scope of this survey at the time of writing but merit attention in future updates. First, **agentic testing frameworks** where multiple LLM agents collaborate with specialized roles (test designer, test executor, oracle judge, and coverage analyst) are beginning to emerge, though their optimization dynamics remain uncharacterized. Second, the use of **code-specific foundation models** (DeepSeek-Coder-V2, StarCoder2, and CodeLlama) as drop-in replacements for general-purpose LLMs in L4–L5 methods may substantially alter the cost–quality trade-off. Third, **LLM-guided neural architecture search** for evolving test generation models represents a frontier where automated machine learning intersects with multi-objective optimization. These directions are expected to feature prominently in the next update of this mapping study.

We note that, with the exception of three late additions incorporated during revision and explicitly identified as such (CoverUp [30], TestDecision [21], E-Test [31]; see Section 3 for details), papers published after May 2025 are not included in the main corpus. A living version of this mapping study will incorporate new developments on an ongoing basis.

11.12. Research Roadmap

We propose the following timeline:

1. **Immediate (2025–2026):** standardized benchmarks; cost-aware LLM invocation; reproducibility mandates.
2. **Medium-term (2026–2028):** many-objective LLM/RL methods; cross-project transfer; industrial CI/CD integration.
3. **Long-term (2028+):** multi-agent LLM optimization; safety-certified AI-driven testing; fully autonomous self-improving test suites.

12. Conclusions

This survey has traced the mathematical and computational evolution of multi-objective test optimization—from classical Pareto-based evolutionary algorithms to the emerging frontier of LLM-driven and hybrid neuro-evolutionary methods. We proposed a novel five-level taxonomy that captures the increasing role of mathematical intelligence, formalized the core optimization problems using rigorous notation, and systematically analyzed 120+ approaches.

Our analysis reveals a clear trajectory “From Pareto to Neural”—and, more precisely, toward hybrid Pareto-neural optimization—but also a more nuanced finding: the levels are **complementary** rather than strictly progressive. Classical MOEAs (L1) remain the method of choice for well-defined coverage optimization with hundreds of objectives. ML-guided methods (L2) reduce evaluation costs. RL-driven approaches (L3) excel in adaptive, online settings. LLM-driven methods (L4) bring unprecedented semantic understanding. Hybrid neuro-evolutionary systems (L5) consistently achieve the best results by combining complementary strengths.

The field stands at an inflection point where the mathematical foundations of multi-objective optimization intersect with the semantic capabilities of modern AI. Realizing the vision of optimization systems that are simultaneously *semantically aware*, *computationally efficient*, and *continuously adaptive* requires addressing the ten critical challenges we have identified—challenges that demand both mathematical rigor and domain-specific engineering knowledge, spanning formal convergence theory, sample complexity analysis, and the metric geometry of semantic optimization spaces.

Supplementary Materials: The following supporting information can be downloaded at <https://www.mdpi.com/article/10.3390/math14152694/s1>: Table S1: Complete corpus of 126 primary studies with taxonomy classification and extracted metadata; Table S2: Inter-rater agreement data for inclusion screening and taxonomy classification (30-paper sample, Cohen’s κ analysis); Table S3: PRISMA flow data with counts and exclusion reasons at each screening stage.

Author Contributions: Conceptualization, methodology, investigation, writing—original draft preparation, and writing—review and editing were all performed by X.Z. and W.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding. The APC was funded by the authors.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Acknowledgments: The author gratefully acknowledges the researchers whose work is surveyed in this paper.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

APFD	Average Percentage of Faults Detected
CI	Continuous Integration
DNN	Deep Neural Network
EA	Evolutionary Algorithm
LLM	Large Language Model
MDP	Markov Decision Process
MOEA	Multi-Objective Evolutionary Algorithm
MOP	Multi-Objective Optimization Problem
MORL	Multi-Objective Reinforcement Learning
NSGA-II	Non-dominated Sorting Genetic Algorithm II
PPO	Proximal Policy Optimization
RAG	Retrieval-Augmented Generation
RL	Reinforcement Learning
SAEA	Surrogate-Assisted Evolutionary Algorithm
SBST	Search-Based Software Testing
TCG	Test Case Generation
TCP	Test Case Prioritization
TCS	Test Case Selection
TSM	Test Suite Minimization

References

1. Yoo, S.; Harman, M. Regression testing minimization, selection and prioritization: A survey. *Softw. Test. Verif. Reliab.* **2012**, *22*, 67–120. [\[CrossRef\]](#)
2. Nouwou Mindom, P.S.; Nikanjam, A.; Khomh, F. A comparison of reinforcement learning frameworks for software testing tasks. *Empir. Softw. Eng.* **2023**, *28*, 111. [\[CrossRef\]](#)
3. Samad, A.; Mahdin, H.B.; Kazmi, R.; Ibrahim, R.; Baharum, Z. Multiobjective test case prioritization using test case effectiveness: Multicriteria scoring method. *Sci. Program.* **2021**, *2021*, 9988987. [\[CrossRef\]](#)
4. Bertolino, A.; Guerriero, A.; Miranda, B.; Pietrantuono, R.; Russo, S. Learning-to-rank vs ranking-to-learn: Strategies for regression testing in continuous integration. In Proceedings of the 42nd IEEE/ACM International Conference on Software Engineering (ICSE), Seoul, Republic of Korea, 27 June–19 July 2020; pp. 1–12.
5. Mohd-Shafie, M.L.; Kadir, W.M.N.W.; Lichter, H.; Khatibsyarbini, M.; Isa, M.A. Model-based test case generation and prioritization: A systematic literature review. *Softw. Syst. Model.* **2022**, *21*, 717–753.
6. Pradhan, D.; Wang, S.; Ali, S.; Yue, T. Search-based cost-effective test case selection within a time budget: An empirical study. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), Denver, CO, USA, 20–24 July 2016.
7. Fraser, G.; Arcuri, A. EvoSuite: Automatic test suite generation for object-oriented software. In Proceedings of the 19th ACM SIGSOFT Symposium on Foundations of Software Engineering (ESEC/FSE), Szeged, Hungary, 5–9 September 2011; pp. 416–419.
8. Xue, Y.; Li, Y.F. Multi-objective integer programming approaches for solving the multi-criteria test-suite minimization problem. *ACM Trans. Softw. Eng. Methodol.* **2020**, *29*, 1–50. [\[CrossRef\]](#)
9. Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **2002**, *6*, 182–197. [\[CrossRef\]](#)
10. Panichella, A.; Kifetew, F.M.; Tonella, P. Reformulating branch coverage as a many-objective optimization problem. In Proceedings of the 8th IEEE International Conference on Software Testing, Verification and Validation (ICST), Graz, Austria, 13–17 April 2015.
11. Panichella, A.; Kifetew, F.M.; Tonella, P. Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets. *IEEE Trans. Softw. Eng.* **2018**, *44*, 122–158. [\[CrossRef\]](#)
12. Arcuri, A. Test suite generation with the Many Independent Objective (MIO) algorithm. *Inf. Softw. Technol.* **2018**, *104*, 195–206. [\[CrossRef\]](#)
13. Catal, C.; Mishra, D. Test case prioritization: A systematic mapping study. *Softw. Qual. J.* **2013**, *21*, 445–478.
14. Pan, R.; Bagherzadeh, M.; Ghaleb, T.A.; Briand, L. Test case selection and prioritization using machine learning: A systematic literature review. *arXiv* **2021**, arXiv:2106.13891.
15. Khatibsyarbini, M.; Isa, M.A.; Jawawi, D.N.; Tumeng, R. Test case prioritization approaches in regression testing: A systematic literature review. *Inf. Softw. Technol.* **2018**, *93*, 74–93. [\[CrossRef\]](#)

16. Mehmood, A.; Ilyas, Q.M.; Ahmad, M.; Shi, Z. Test suite optimization using machine learning techniques: A comprehensive study. *IEEE Access* **2024**, *12*, 168645–168670. [\[CrossRef\]](#)
17. Hu, Q.; Guo, Y.; Xie, X.; Cordy, M.; Ma, W.; Papadakis, M.; Le Traon, Y. Test optimization in DNN testing: A survey. *ACM Trans. Softw. Eng. Methodol.* **2024**, *33*, 111:1–111:42. [\[CrossRef\]](#)
18. Augusto, C.; Bertolino, A.; De Angelis, G.; Lonetti, F.; Morán, J. Large language models for software testing: A research roadmap. *arXiv* **2025**, arXiv:2509.25043.
19. Stapleton, F.; Galván, E. Initial steps towards tackling high-dimensional surrogate models for neuroevolution. In Proceedings of the International Conference on Optimization and Learning (OLA), Dubrovnik, Croatia, 13–15 May 2024.
20. Stapleton, F.; Cody-Kenny, B.; Galván, E. NeuroLGP-SM: A surrogate-assisted neuroevolution approach using linear genetic programming. In Proceedings of the International Conference on Optimization and Learning (OLA), Dubrovnik, Croatia, 13–15 May 2024.
21. Wang, G.; Yang, C.; Zhou, X.; Sun, Z.; Wang, B.; Lo D.; Hao, D. TestDecision: Sequential test suite generation via greedy optimization and reinforcement learning. *arXiv* **2025**, arXiv:2604.01799.
22. Chand, S.; Wagner, M. Evolutionary many-objective optimization: A quick-start guide. *Surv. Oper. Res. Manag. Sci.* **2015**, *20*, 35–42. [\[CrossRef\]](#)
23. Rothmel, G.; Untch, R.H.; Chu, C.; Harrold, M.J. Test case prioritization: An empirical study. In Proceedings of the IEEE International Conference on Software Maintenance (ICSM), Oxford, UK, 30 August–3 September 1999; pp. 179–188.
24. Das, S.; Gary, K. An AI driven decision system for value aware regression testing. In *Agile Processes in Software Engineering and Extreme Programming*; Springer Nature Switzerland: Cham, Switzerland, 2026; Volume 578, pp. 175–190.
25. Panichella, A.; Oliveto, R.; Di Penta, M.; De Lucia, A. Improving multi-objective test case selection by injecting diversity in genetic algorithms. *IEEE Trans. Softw. Eng.* **2015**, *41*, 358–383. [\[CrossRef\]](#)
26. Haq, F.U.; Shin, D.; Briand, L. Many-objective reinforcement learning for online testing of DNN-enabled systems. In Proceedings of the 45th IEEE/ACM International Conference on Software Engineering (ICSE), Melbourne, Australia, 14–20 May 2023; pp. 1814–1826.
27. Budgen, D.; Turner, M.; Brereton, P.; Kitchenham, B. Using mapping studies in software engineering. In Proceedings of the 20th Annual Workshop of the Psychology of Programming Interest Group (PPIG), Lancaster, UK, 10–12 September 2008; pp. 195–204.
28. Torres-Carrión, P.V.; González-González, C.S.; Aciar, S.; Rodríguez-Morales, G. Methodology for systematic literature review applied to engineering and education. In Proceedings of the IEEE Global Engineering Education Conference (EDUCON), Santa Cruz de Tenerife, Spain, 17–20 April 2018; pp. 1364–1373.
29. Kitchenham, B.A.; Budgen, D.; Brereton, P. *Evidence-Based Software Engineering and Systematic Reviews*; CRC Press: Boca Raton, FL, USA, 2023.
30. Altmayer Pizzorno, J.; Berger, E.D. Coverup: Effective high coverage test generation for python. *Proc. Acm Softw. Eng.* **2025**, *2*, 2897–2919. [\[CrossRef\]](#)
31. Lu, Z.; Zhang, P.; Nie, Y.; Yang, Y.; Tang, Y.; Chong, C.Y.; Zhou, Y. Beyond Coverage: Automatic Test Suite Augmentation for Enhanced Effectiveness using Large Language Models. In *Proceedings of the ACM on Programming Languages*, 10 (OOPSLA1); Association for Computing Machinery: New York, NY, USA, 2026; pp. 1403–1431.
32. Aghababaeian, Z.; Abdellatif, M.; Dadkhah, M.; Briand, L. DeepGD: A multi-objective black-box test selection approach for deep neural networks. *arXiv* **2023**, arXiv:2303.04878.
33. Lemieux, C.; Inala, J.P.; Lahiri, S.K.; Sen, S. CodaMosa: Escaping coverage plateaus in test generation with pre-trained large language models. In Proceedings of the 45th IEEE/ACM International Conference on Software Engineering (ICSE), Melbourne, Australia, 14–20 May 2023; pp. 919–931.
34. Yang, C.; Chen, J.; Lin, B.; Wang, Z.; Zhou, J. Advancing code coverage: Incorporating program analysis with large language models. *ACM Trans. Softw. Eng. Methodol.* **2026**, *35*, 1–31. [\[CrossRef\]](#)
35. Huynh, H.; Pham, N.; Nguyen, T.N.; Nguyen, V. Segment-based test case prioritization: A multi-objective approach. In Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), Vienna, Austria, 16–20 September 2024.
36. Bouafif, M.S.; Hamdaqa, M.; Zulkoski, E. Primg: Efficient llm-driven test generation using mutant prioritization. In Proceedings of the 29th International Conference on Evaluation and Assessment in Software Engineering, Istanbul, Turkey, 17–20 June 2025; pp. 1107–1116.
37. Yu, X.; Wang, Y.; Zhu, L.; Filev, D.; Yao, X. Experience-based evolutionary algorithms for expensive optimization. *IEEE Trans. Evol. Comput.* **2023**, *27*, 1390–1404.
38. Spieker, H.; Gotlieb, A.; Marijan, D.; Mossige, M. Reinforcement learning for automatic test case prioritization and selection in continuous integration. In Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), Santa Barbara, CA, USA, 10–14 July 2017; pp. 12–22.

39. Corradini, D.; Montolli, Z.; Pasqua, M.; Ceccato, M. DeepREST: Automated test case generation for REST APIs exploiting deep reinforcement learning. In Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE), Sacramento, CA, USA, 27 October–1 November 2024; pp. 1383–1394.
40. Ling, Y.; Yu, S.; Fang, C.; Pan, G.; Wang, J.; Liu, J. Redefining crowdsourced test report prioritization: An innovative approach with large language model. *Inf. Softw. Technol.* **2025**, *179*, 107629. [\[CrossRef\]](#)
41. Broide, L.; Stern, R.; Mordoch, A. EvoGPT: Leveraging LLM-driven seed diversity to improve search-based test suite generation. *arXiv* **2025**, arXiv:2505.12424.
42. Li, K.; Yuan, Y.; Yu, H.; Guo, T.; Cao, S. CoCoEvo: Co-evolution of programs and test cases to enhance code generation. *arXiv* **2025**, arXiv:2502.10802.
43. Saber, T. Enhancing program synthesis with large language models using many-objective grammar-guided genetic programming. *Algorithms* **2024**, *17*, 287. [\[CrossRef\]](#)
44. Rudolph, G. On a multi-objective evolutionary algorithm and its convergence to the Pareto set. In Proceedings of the IEEE International Conference on Evolutionary Computation (CEC), Anchorage, AK, USA, 4–9 May 1998; pp. 511–516.
45. Campos, J.; Ge, Y.; Albulian, N.; Fraser, G.; Eler, M.; Arcuri, A. An empirical evaluation of evolutionary algorithms for unit test suite generation. *Inf. Softw. Technol.* **2018**, *104*, 207–235. [\[CrossRef\]](#)
46. Li, D.; Wong, W.E.; Pan, S.; Koh, L.-S.; Li, S.; Chau, M. Automatic test case generation using many-objective search and principal component analysis. *IEEE Access* **2022**, *10*, 85518–85529. [\[CrossRef\]](#)
47. Marchetto, A.; Islam, M.M.; Asghar, W.; Susi, A.; Scanniello, G. A multi-objective technique to prioritize test cases. *IEEE Trans. Softw. Eng.* **2016**, *42*, 918–940. [\[CrossRef\]](#)
48. Abdessalem, R.B.; Panichella, A.; Nejati, S.; Briand, L.C.; Stifter, T. Testing autonomous cars for feature interaction failures using many-objective search. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE), Montpellier, France, 3–7 September 2018; pp. 143–154.
49. Kim, M.; Sinha, S.; Orso, A. Adaptive REST API testing with reinforcement learning. In Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), Echternach, Luxembourg, 11–15 September 2023; pp. 446–458.
50. Giamattei, L.; Biagiola, M.; Pietrantuono, R.; Russo, S.; Tonella, P. Reinforcement learning for online testing of autonomous driving systems: A replication and extension study. *Empir. Softw. Eng.* **2025**, *30*, 19. [\[PubMed\]](#)
51. Zheng, Y.; Xie, X.; Su, T.; Ma, L.; Hao, J.; Meng, Z.; Liu, Y.; Shen, R.; Chen, Y.; Fan, C. Wuji: Automatic online combat game testing using evolutionary deep reinforcement learning. In Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), San Diego, CA, USA, 11–15 November 2019; pp. 772–784.
52. Kalyani, P.; Rao, C.P.; Goparaju, B.; Babu, K.K.; Kandimalla, P.C.R. BugPrioritizeAI for multimodal test case prioritisation using bug reports, code changes, and test metadata. *Sci. Rep.* **2025**, *16*, 1539. [\[CrossRef\]](#) [\[PubMed\]](#)
53. Gu, S.; Zhang, Q.; Li, K.; Fang, C.; Tian, F.; Zhu, L.; Zhou, J.; Chen, Z. TestART: Improving LLM-based unit testing via co-evolution of automated generation and repair iteration. *arXiv* **2025**, arXiv:2408.03095.
54. Tufano, M.; Drain, D.; Svyatkovskiy, A.; Deng, S.K.; Sundaresan, N. Unit test case generation with transformers and focal context. *arXiv* **2020**, arXiv:2009.05617.
55. Hao, H.; Zhang, X.; Zhou, A. Large language models as surrogate models in evolutionary algorithms: A preliminary study. *Swarm Evol. Comput.* **2024**, *91*, 101741. [\[CrossRef\]](#)
56. Ye, H.; Wang, J.; Cao, Z.; Berto, F.; Hua, C.; Kim, H.; Park, J.; Song, G. ReEvo: Large language models as hyper-heuristics with reflective evolution. *arXiv* **2024**, arXiv:2402.01145.
57. Deb, K.; Jain, H. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, Part I: Solving problems with box constraints. *IEEE Trans. Evol. Comput.* **2014**, *18*, 577–601. [\[CrossRef\]](#)
58. Zhang, Q.; Li, H. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Trans. Evol. Comput.* **2007**, *11*, 712–731. [\[CrossRef\]](#)

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.