Journal of Innovative Technologies
https://journalinnotech.com/
Vol 2(1) 2026

Fault-Detection Efficiency Under Fixed Compute Budgets: A Comparative Evaluation of Test Case Selection and Prioritization Strategies for Autonomous Driving Simulation Regression Testing

Jianuo Wang[1], Yi Guo[2,*]

[1] Electrical and Computer Engineering, University of Michigan, Ann Arbor, MI, USA
[2] Computer and Information Science, University of Pennsylvania, Philadelphia, PA, USA
* Corresponding Email: guoyi_cs@gmail.com

Abstract

Regression testing of autonomous driving software relies on simulation scenario libraries that grow continuously, while the computational budget available for each verification cycle remains bounded. Executing the full scenario suite is expensive, and uninformed sampling frequently fails to surface defect-revealing cases. This study reports a comparative evaluation of five test case selection and prioritization strategies for simulation-based regression testing under fixed execution budgets: random sampling, additional-coverage greedy selection over the scenario parameter space, similarity-based redundancy reduction, learning to rank over historical execution records, and change-aware selection driven by a module dependency mapping. Scenarios are encoded by actor class, manoeuvre, ego behaviour, and proximity, giving a common input representation across strategies. The evaluation uses 33,530 recorded executable driving scenarios from a public autonomous driving system scenario dataset, drawn from 2,880 test executions and arranged as a regression matrix of 87 reusable scenario configurations across 48 operational contexts, of which 40 are used for evaluation; the cost sensitivity of the learned ranker, against a random baseline, is then examined on two industrial continuous-integration histories that record real per-test execution cost. Learning to rank attains the highest average percentage of faults detected (0.837 against 0.754 for random selection) and reaches the first failing case at position 1.19 on average. In this corpus that advantage cannot be separated from replay density, because a case's verdict is the union of violations over an uneven number of recorded instances. Change-aware selection, driven by an operational-context proxy for code change, falls 0.043 APFD below random over the full ordering, a gap that pairwise contrasts detect but that does not survive the Nemenyi post-hoc procedure. Cost-cognizant scoring reverses the ranking of history-based variants on both industrial histories.

***Keywords*:** regression test prioritization, autonomous driving simulation, compute budget allocation, empirical comparison

This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/)

1. Introduction

1.1. Background and Motivation

Simulation has become the dominant vehicle for validating autonomous driving systems (ADS), since executing a driving stack against synthetic traffic is cheaper and safer than road testing, and open platforms such as CARLA have made large-scale scenario execution widely accessible [1]. The economics remain awkward. A library that begins with a few hundred hand-written cases expands to tens of thousands once parameter sweeps, regulator-derived pre-crash typologies, and search-generated corner cases are folded in, while the window between a commit and the next integration build does not expand at all. Each simulated route occupies graphics hardware for seconds to minutes, so a full library pass can consume a cluster for days. Work on surrogate-assisted testing quantified this pressure directly, replacing simulator invocations with learned approximations precisely because those invocations dominate verification cost [2].

Practitioners respond by curating a "core" subset of scenarios by hand. Such a list is cheap to maintain and easy to defend to auditors, yet carries no quantitative guarantee: it is not updated as the driving stack evolves, and its fault-detection capability relative to the discarded remainder is rarely measured. The testing community has studied the same tension for three decades under the headings of suite minimisation, regression test selection, and test case prioritization [3], producing algorithms that decide which tests to run and in what order. What is missing is evidence on how they behave when the tests are parameterised traffic situations and the binding constraint is graphics compute.

1.2. Problem Statement, Research Questions, and Contributions

The problem addressed here is narrow by construction. The scenario library and the compute budget are both given. The question is only which algorithm should decide what to run, and in what order, so that the defects exposed per unit of compute are maximised. Search-based selection of driving scenarios has been examined for a single method [4]; a controlled comparison across method families under a shared encoding and a shared fault model has not been reported.

A. Research Questions

Three questions organise the study. RQ1 asks how the five strategies rank on fault-detection rate when the whole suite is ordered, and whether that ranking is stable under alternative definitions of what counts as a fault. RQ2 asks how much compute each strategy needs to reach the first failing scenario, and how many faults it recovers at budgets of 5, 10, 20, and 50 percent of the suite. RQ3 asks what the frontier between suite reduction and missed faults looks like, and whether conclusions drawn from execution-count budgets survive when execution cost varies across tests.

B. Contributions and Organisation

The study contributes a reproducible comparison of five selection and prioritization strategies on a regression matrix built from recorded autonomous driving simulation scenarios, a quantification of the reduction–miss frontier for each strategy, and evidence that the ranking of strategies depends on both the fault model and the cost model adopted. Section 2 reviews the literature, Section 3 describes the data and design, Section 4 reports results, Section 5 examines threats to validity, and Section 6 concludes and outlines future work.

2. Related Work

2.1. Regression Test Selection, Minimisation, and Prioritization

A. Coverage-Based and Cost-Cognizant Prioritization

The formal statement of the prioritization problem, together with the average percentage of faults detected (APFD) measure that remains the field's default, was given by Rothermel and colleagues, who showed that coverage-guided orderings recover faults substantially earlier than untreated orderings [5]. A subsequent family of controlled experiments established the comparative methodology still used today, varying subject programs, fault sets, and technique granularity within one design [6]. Both lines assume uniform test cost, an assumption that fails whenever tests differ in runtime; the cost-cognizant reformulation weights each test by its execution cost and each fault by severity, and can reorder techniques that appear equivalent under the uniform assumption [7]. Industrial evaluation of coverage-driven prioritization on production software confirmed that the additional-coverage greedy heuristic, rather than total-coverage greedy, is the variant worth deploying [8].

B. Diversity-, Learning-, and Change-Based Selection

A parallel line abandons structural coverage in favour of input-space diversity, on the argument that inputs far apart in the specification space are unlikely to fail together; adaptive random testing formalised this intuition and demonstrated gains over uniform sampling [9]. Clustering variants group similar tests and sample within groups, which scales better than pairwise diversity computation while retaining much of its benefit [10]. Learning-based selection instead mines historical execution records, and deployment at industrial scale has shown that a learned selector can halve testing infrastructure cost while still reporting the overwhelming majority of individual test failures [11]. Change-aware selection takes a third route, computing which tests depend on modified code and skipping the remainder, with reported cost models showing that testing less need not sacrifice quality when the dependency relation is accurate [12].

2.2. Simulation-Based Testing of Autonomous Driving Systems

Within the driving domain, most testing research has concentrated on generating critical scenarios rather than selecting among existing ones. Many-objective search has been used to expose feature interaction failures in driving stacks by steering scenario parameters towards safety violations [13], and probabilistic scenario description languages allow libraries to be specified compactly and sampled systematically [14]. Both enlarge the library; neither addresses what to execute once the library exceeds the available compute.

2.3. Positioning of This Study

This work proposes no selection algorithm. It fixes the encoding, the fault model, the budget levels, and the protocol, then measures five established strategy families against each other on recorded driving simulation outcomes. The contribution is decision-relevant evidence about which family to adopt under which conditions, with an explicit account of how sensitive that advice is to modelling choices usually left implicit.

2.4. Extended Evidence Base for Scenario Coverage and Perception Robustness

Simulation-test prioritization depends on both scenario diversity and the failure surface of the perception stack, so recent work on coverage, sensor fusion, environmental corruption, and road-scene understanding is directly relevant. The following additions are included as adjacent evidence rather than as results of the present experiment.

The extended literature includes Guo's work on “A Multi-Dimensional Coverage Metric with Evolutionary Search for Safety-Critical Scenario Generation in…”[24]; Li's work on “Benchmarking Robustness-Efficiency Trade-offs of Camera-LiDAR Fusion Strategies for 3D Object Detection…”[25]; Wu's work on “Caption Generation from Road Images for Traffic Scene Construction”[26]; and Li's work on “Caption generation from road images for traffic scene modeling”[27]. These studies motivate broader external-validity, robustness, and governance checks without changing the design or numerical findings reported below.

The extended literature includes Guo's work on “Performance Evaluation of Lightweight Detection Algorithms on Compact LiDAR-Camera Configurations for…”[28]; Guo's work on “Reliability Assessment and Adaptive Fusion Algorithm for Multi-Sensor Data in Autonomous Driving under…”[29]; Wu's work on “Jointly detecting and retrieving vehicles from road image sequences based on CNN”[30]; and Li's work on “Comparative Analysis of Illumination Normalization Methods for Autonomous Driving Under Challenging…”[31]. These studies motivate broader external-validity, robustness, and governance checks without changing the design or numerical findings reported below.

The extended literature includes Huo's work on “Semantic segmentation and scene reconstruction for traffic simulation using CNN”[32]. These studies motivate broader external-validity, robustness, and governance checks without changing the design or numerical findings reported below.

3. Experimental Design

3.1. Subject Data and the Regression Matrix

The primary subject is DeepScenario, a public dataset of driving scenarios collected by executing an autonomous driving system in simulation, in which every scenario carries recorded rather than predicted outcomes [15]. It contains 33,530 executable scenarios from 2,880 test executions across three generation strategies, three reward settings, four roads, and four weather and lighting conditions, each recording time to collision, distance to obstacles, jerk, a collision flag, the collision partner type, and speed at collision. All published summary statistics were reproduced exactly before use, including the 1,050 collision scenarios split into 917 vehicle, 105 pedestrian, and 28 static-obstacle collisions.

The scenarios are rearranged into a regression matrix. A test case is a distinct recorded scenario configuration, identified by its configuration description string; 87 such configurations recur across the dataset. For the strategies that require it, each configuration is additionally encoded by parsing that description into actor class, manoeuvre, ego behaviour, and proximity, together with the modules it exercises; two pedestrian configurations that differ only in crossing direction coincide under this encoding, so 86 distinct feature vectors and module sets back the 87 cases, and coverage-, similarity-, and change-aware selection treat that pair as interchangeable. A verification cycle is one operational context defined by road, weather, and reward setting, giving 48 cycles. Executing a case in a cycle means replaying the recorded instances of that configuration in that context and taking the union of the violations observed. Four requirements are checked with thresholds fixed before any results were inspected: collision occurrence, time to collision below 0.5 s, distance to obstacles below 2.5 m, and jerk above 15.0 m/s³. On average 28.8 percent of cases fail in a cycle, a high density reflecting the search-driven origin of the library.

Two industrial continuous-integration (CI) histories accompany the analysis because the driving dataset records no execution time. The ABB Paint Control and IOF/ROL histories, released with earlier work on reinforcement-learning prioritization, record real per-test durations alongside real verdicts [16]. Table 1 summarises the three subjects.

Table 1. Characteristics of the three evaluation subjects

Property	DeepScenario	ABB Paint Control	ABB IOF/ROL
Domain	ADS simulation	Industrial CI	Industrial CI
Records	33,530 scenarios	25,594 executions	32,260 executions
Test cases in suite	87 configurations	89 tests	1,941 tests
Cycles available / used	48 / 40	352 / 247	320 / 181
Failure rate	28.8% of case–cycle pairs	19.4%	28.8%

Property	DeepScenario	ABB Paint Control	ABB IOF/ROL
Execution cost recorded	no	yes	yes
Median cost (arbitrary units)	—	25,736	57,046
Licence	GPL-3.0	research release	research release

Sources: computed by the accompanying code from the published dataset files, cross-checked against the original releases. Cycles used are the cycles actually scored; the admission rule for each subject is stated in Section 3.4.

3.2. Scenario Encoding and the Five Strategies

Every strategy consumes the same representation: a categorical feature vector over actor class, manoeuvre, ego behaviour, and proximity, plus the driving stack modules the configuration exercises. The mapping to modules is deterministic, assigning pedestrian scenarios to vulnerable-road-user perception, vehicle scenarios to vehicle perception, crossing and oncoming manoeuvres to prediction, lane changes and overtaking to lateral planning, and lead-vehicle following to longitudinal planning. Table 2 records what each strategy requires.

Table 2. The five strategies and their information requirements

Strategy	Abbrev.	Signal used	Stochastic
Random sampling	RND	none	yes, 30 seeds
Additional-coverage greedy	AGC	parameter-space coverage items	tie-breaking, 30 seeds
Similarity-based redundancy reduction	SIM	pairwise feature distance	start point, 30 seeds
Learning to rank	LTR	historical execution records	model seed, 5 repetitions
Change-aware selection	CAS	context change and module dependency	tie-breaking, 30 seeds

A. Random, Coverage, and Similarity Strategies

Random sampling draws a uniform permutation and supplies the reference for the others. Additional-coverage greedy treats each feature value and each pairwise feature combination as a coverage item, repeatedly selecting the case covering the most items not yet covered and resetting once saturation is reached. Similarity-based reduction orders the suite by farthest-point traversal

under Hamming distance on the one-hot features, so each new case is the one most dissimilar from everything chosen, following the diversity argument developed for model-based test selection [17].

B. Learning-to-Rank and Change-Aware Strategies

The learning strategy scores each case with a gradient-boosted classifier trained only on cycles strictly preceding the one being predicted, which rules out leakage from later cycles in the fixed order. Features are the case's historical and exponentially weighted failure rates, cycles since its last failure, the mean number of signatures it has produced, the failure rate of its modules, and the static feature vector. The first evaluation cycle has no training cycle strictly between the warm-up window and itself, so the learned ranker falls back to a random permutation there; this affects one of the forty evaluation cycles. Change-aware selection mirrors file-level dependency selection as practised in continuous integration [18], substituting an operational-context delta for a source diff: a weather change marks the perception modules affected, a road change marks prediction and planning, and a reward-setting change marks the planning modules. Cases exercising affected modules rank ahead of the remainder, with ties broken at random. Where execution cost is recorded, as on the two industrial histories, a cost-cognizant variant of the learned ranker, reported as cost-cognizant rank in Table 5, divides each predicted failure probability by the recorded execution cost of the test; it is defined only for those subjects, since the driving dataset records no execution time.

3.3. Fault Model and Evaluation Metrics

A. Failure Signatures and Failing Cases

Two fault models are carried through rather than one, because the choice is consequential and rarely justified. Under the primary model a fault is a distinct failure signature, pairing a violated requirement with an exercised module, so many scenarios exposing the same weakness count once; 17.0 signatures appear per evaluation cycle on average. Under the secondary model every failing case counts separately, giving 25.0 faults per cycle. Reporting both shows whether a strategy ranks well by finding varied weaknesses or many instances of one.

B. Budgeted Detection, Time to First Failure, and Cost-Cognizant Scoring

APFD is one minus the mean normalised position of the first case revealing each fault, plus a finite-sample correction. Budgeted detection reports the fraction of a cycle's faults recovered within the first 5, 10, 20, or 50 percent of the suite, which for 87 cases is 4, 9, 17, and 44 executions, and time to first failure is the 1-indexed position of the first failing case in the execution order, so a strategy that fails on its very first execution scores 1. Where cost is recorded, the cost-cognizant variant of APFD, denoted APFDc, replaces test positions by accumulated cost, summing for each fault the cost remaining after the revealing test minus half that test's own cost, normalised by total suite cost and fault count.

3.4. Protocol and Statistical Analysis

Cycles are ordered lexicographically by road, weather, and reward setting, since the dataset records no acquisition time and every recorded execution spans all 48 contexts, with the first eight reserved as history and the remaining 40 used for evaluation, so every learned model is fitted only on cycles earlier in this fixed order. On the two industrial histories the protocol differs: a cycle enters the analysis only if it contains at least twelve executed tests and at least one failing verdict, since APFD is undefined for a failure-free cycle and near-degenerate for very short ones, which retains 252 of 352 cycles on Paint Control and 187 of 320 on IOF/ROL; of these, only the cycles beyond the initial window in which the learned ranker accumulates training history are scored, leaving 247 and 181 cycles respectively. Stochastic strategies are repeated with the seed counts in Table 2 and averaged within each cycle before comparison. The learned ranker is repeated five rather than thirty times because each repetition retrains the model on every cycle prefix; its only stochastic elements are the classifier's random state and the jitter that breaks score ties, so its per-cycle APFD is identical across seeds in 16 of the 40 cycles, the seed-to-seed range stays below 0.01 in 38 of them, and the seed-to-seed standard deviation averages 0.003, an order of magnitude below the 0.074 observed for random sampling at thirty seeds. The resulting seed-induced standard error of its per-cycle mean, 0.001 APFD, is the smallest of the five strategies, so the reduced repetition count does not weaken the comparison. On the two industrial histories the learned rankers are deterministic single runs and random sampling is repeated with twenty seeds. Differences are assessed with the Friedman test over per-cycle scores, followed by the Nemenyi post-hoc procedure, whose critical difference at the 5 percent level is 0.964 mean ranks for five strategies over 40 cycles, and pairwise contrasts report the Vargha–Delaney effect size. Nominally identical tests are known to yield inconsistent verdicts in continuous integration [19]; outcomes here are replayed from recorded executions, so no such variance enters the measurements.

4. Results and Analysis

4.1. RQ1: Fault-Detection Rate Across Strategies

A. Ranking Under the Signature Fault Model

Table 3 reports the headline comparison. Learning to rank achieves the highest APFD over failure signatures at 0.837, ahead of additional-coverage greedy at 0.801, similarity-based reduction at 0.785, random sampling at 0.754, and change-aware selection at 0.711. The Friedman test rejects equality ($\chi^2 = 61.82$, $p = 1.2 \times 10^{-12}$) over 40 cycles. Under the critical difference of 0.964, learning to rank and additional-coverage greedy both separate from random sampling, while the 0.60-rank gap between the two leaders is not distinguishable, and similarity-based reduction sits 0.85 ranks above random without separating despite a significant pairwise contrast. The gains are moderate: the best strategy improves APFD by 0.084 over random, with a Vargha–Delaney effect size of 0.167. The attribution of that gain is less secure than the ordering suggests. Because

a case–cycle verdict is the union of violations over whatever number of instances the generator happened to record for that configuration, and recorded instance counts vary widely across cases, a case's failure probability is partly a function of replay density; no outcome-free baseline ordered by replay count alone was run here, so at the aggregate level the contribution of the learned failure history cannot be separated from that of replay density in this corpus. Section 5 quantifies the confound.

The most useful negative result concerns change-aware selection, which lands 0.043 APFD below random sampling, with a Vargha–Delaney effect size of 0.677 on the pairwise contrast (Holm-adjusted Wilcoxon $p = 9.0 \times 10^{-5}$). The 0.65 mean-rank gap is still smaller than the Nemenyi critical difference of 0.964, so the two are not separated by the omnibus post-hoc procedure, and the pairwise and post-hoc verdicts should be read together. The deficit is also budget-dependent: change-aware selection leads random sampling at the 5, 10, and 20 percent budgets, recovering 28.2 against 20.5, 45.2 against 38.4, and 58.2 against 56.5 percent of signatures, and falls behind from roughly the 22 percent budget onwards, reaching 75.3 against 84.1 percent at half the suite. Ordering by dependency on the changed context front-loads a module group that is often not where violations are, so once the early advantage is exhausted the substitute change signal buys nothing over uninformed ordering.

Table 3. Prioritization quality over 40 evaluation cycles (mean ± standard deviation across cycles)

Strategy	APFD, signatures	APFD, failing cases	Position of first failing test	Mean rank, signatures
Learning to rank	0.837 ± 0.071	0.631 ± 0.056	1.19 ± 0.57	1.82
Additional-coverage greedy	0.801 ± 0.044	0.501 ± 0.016	2.73 ± 0.75	2.42
Similarity-based redundancy reduction	0.785 ± 0.081	0.519 ± 0.039	2.48 ± 0.42	2.80
Random sampling	0.754 ± 0.045	0.501 ± 0.008	3.48 ± 0.87	3.65
Change-aware selection	0.711 ± 0.079	0.537 ± 0.044	2.81 ± 1.10	4.30

Source: per-cycle results file produced by the accompanying code.

B. Sensitivity to the Fault Model

Switching to the failing-case fault model rearranges the middle of the table. Learning to rank keeps first place and widens its margin to 0.631 against 0.501 for random, but additional-coverage greedy collapses to 0.501, indistinguishable from random, while change-aware selection improves

to 0.537 and moves to second on mean rank. Coverage-driven ordering spreads executions across the parameter space and meets many distinct signatures early, yet is indifferent to where failures cluster in volume; dependency-driven ordering does the reverse. No family dominates under both accounting rules except the history-based ranker, which holds first place under either, although its lead over additional-coverage greedy is not distinguishable at the critical difference under the signature model. Among the remaining strategies the ordering is an artefact of the fault definition rather than a property of the algorithms, since coverage-driven selection leads the non-learned group over signatures at 0.801 yet drops to the level of random ordering over failing cases, while dependency-driven selection does the reverse, sitting 0.043 below random over signatures and at the top of the non-learned group over failing cases. Cautions have been raised about substituting artificial faults for real ones [20]; the accounting rule applied to real failures matters in the same way.

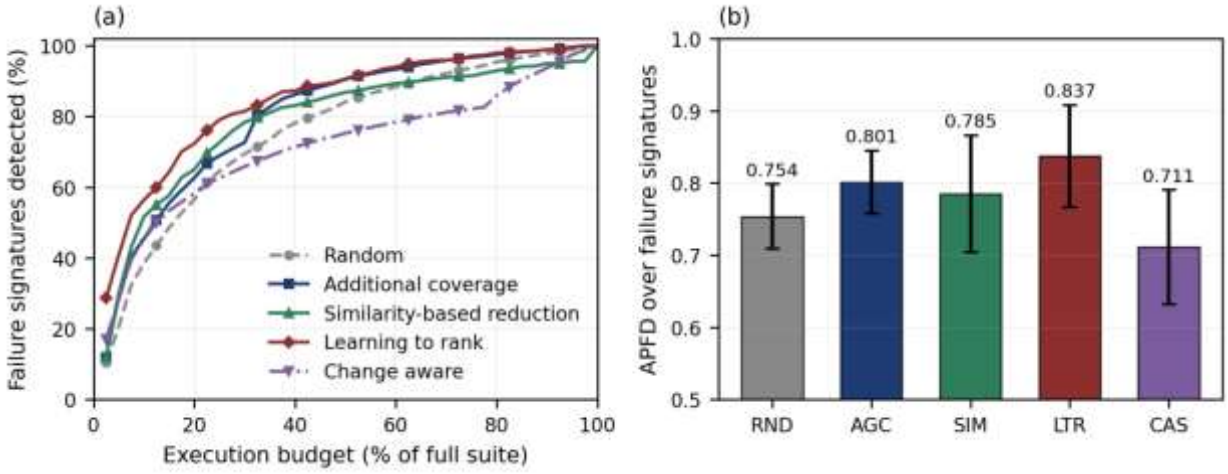


Figure 1. Signature Detection Against Execution Budget and Prioritization Quality

(a) Percentage of a cycle's failure signatures recovered as the execution budget grows from 2.5 to 100 percent of the 87-case suite, averaged over 40 cycles. Learning to rank leads across the low-budget range, recovering 40.8 percent at a 5 percent budget against 20.5 percent for random sampling, and converges with additional-coverage greedy above the 50 percent budget at 90.6 and 90.5 percent. Change-aware selection falls below random beyond the 20 percent budget, reaching 75.3 percent at half the suite. (b) Mean APFD over failure signatures with standard deviation across cycles, showing the 0.084 separation between learning to rank at 0.837 and random sampling at 0.754. Strategies are labelled by the abbreviations defined in Table 2 (RND, AGC, SIM, LTR, CAS).

4.2. RQ2: Efficiency Under Small Budgets

What matters in practice is not the quality of a complete ordering but how much is recovered before the budget runs out. At a 5 percent budget, four executions out of 87, learning to rank recovers 40.8 percent of signatures against 20.5 percent for random sampling. The advantage compresses monotonically as the budget grows, narrowing to 90.6 against 84.1 percent at half the

suite. Half the suite buys roughly nine tenths of the signatures under the better strategies, and the marginal value of a good ordering nearly vanishes once most of the suite runs anyway.

Time to first failure separates the strategies most cleanly. The first failing case appears at position 1.19 on average under learning to rank against 3.48 under random sampling, with the other three strategies between 2.48 and 2.81, so a developer waiting for the first red signal sees it almost three times sooner. The finding that history-based ordering pays off when resources are scarce echoes early results in resource-constrained environments [21]. Table 4 collects the budgeted detection rates.

Table 4. Fault detection at fixed execution budgets, mean over 40 cycles

Strategy	5% budget (4 tests)	10% (9)	20% (17)	50% (44)	Effect size vs. random, APFD
Learning to rank	40.8%	56.4%	72.3%	90.6%	0.167, large
Additional-coverage greedy	29.0%	45.5%	62.6%	90.5%	0.208, large
Similarity-based redundancy reduction	30.0%	51.7%	65.0%	86.8%	0.362, small
Random sampling	20.5%	38.4%	56.5%	84.1%	—
Change-aware selection	28.2%	45.2%	58.2%	75.3%	0.677, medium, favours random

Percentages are the share of a cycle's failure signatures recovered. Effect sizes are Vargha–Delaney statistics on per-cycle APFD, with Holm-adjusted Wilcoxon p-values below 0.014 throughout.

4.3. RQ3: Reduction–Miss Trade-off and Cost-Cognizant Evaluation

A. The Reduction–Miss Frontier

Figure 2 recasts the measurements as the quantity a verification manager negotiates: how many faults are missed for how much compute saved. Discarding half the suite costs 9.4 percentage points of signature coverage under learning to rank and 15.9 under random sampling; discarding four fifths costs 27.7 and 43.5 points. No strategy reaches large reduction with negligible loss, which tempers the expectation that intelligent selection makes aggressive reduction free. Multi-

objective treatments have argued that such frontiers, rather than scalar scores, are the right object of comparison [22], and these curves agree, since strategies close in APFD separate visibly at the low-budget end.

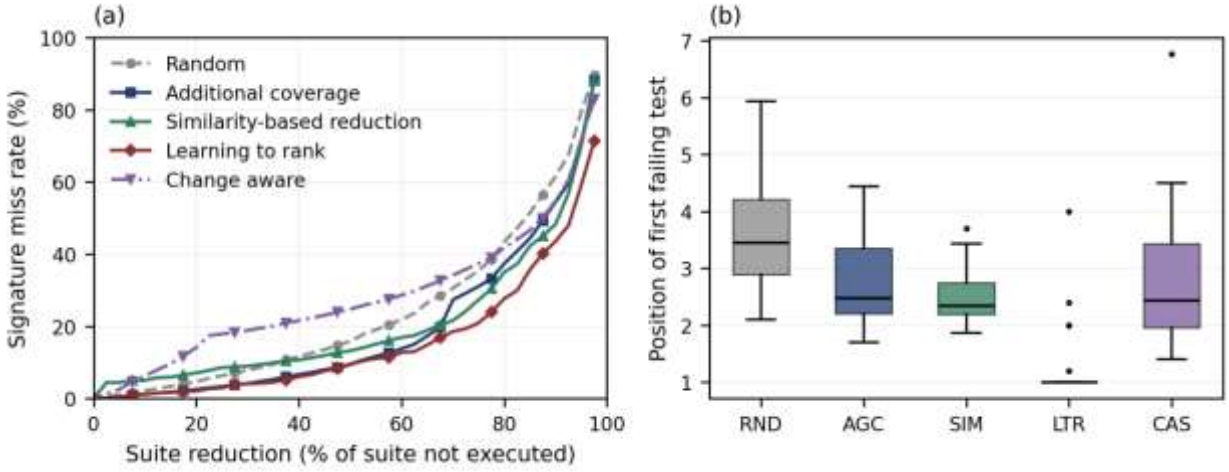


Figure 2. Suite Reduction Against Missed Failure Signatures, and Time to First Failure

(a) Signature miss rate against the share of the suite left unexecuted, averaged over 40 cycles. At 50 percent reduction the miss rate is 9.4 percent for learning to rank and 15.9 percent for random sampling; at 80 percent reduction the gap widens to 27.7 against 43.5 percent. (b) Distribution across cycles of the position of the first failing case, with learning to rank at a mean of 1.19 against 3.48 for random sampling, and change-aware selection showing the widest dispersion, with a standard deviation of 1.10 executions.

B. Rank Reversal Under Cost-Cognizant Scoring

The driving dataset records no execution time, so the cost question is answered on the two industrial histories, where durations vary by more than an order of magnitude. Only random sampling and the two learned variants are transferred to these subjects: the coverage, similarity, and change-aware strategies require the scenario feature encoding and module mapping that the industrial histories do not provide, and faults there are recorded per-test verdicts rather than failure signatures, so the industrial analysis answers the cost question alone and is not a replication of the five-strategy comparison. On ABB Paint Control the uniform-cost learned ranker attains APFD 0.733 while its cost-cognizant variant, which divides predicted failure probability by execution cost, attains only 0.672. Scored by APFDc the ordering flips to 0.771 against 0.724. The same reversal holds on IOF/ROL, where the cost-cognizant variant loses 0.035 APFD and gains 0.050 APFDc, and detection under real cost budgets follows it rather than the APFD leader, reaching 64.2 against 56.9 percent at a 20 percent cost budget on Paint Control.

The reading is narrow but firm. Where a cycle is bounded by compute rather than test count, a strategy chosen on APFD alone can be the wrong choice, and the two metrics disagree on both industrial subjects. Whether the reversal appears on driving scenarios depends on how widely

simulation cost varies across a library, which no public dataset reports; simulator behaviour under repeated execution suggests such runtime characteristics are themselves unstable [23].

Table 5. Cost-cognizant evaluation on two industrial histories with recorded execution durations

Subject	Strategy	APFD	$APFD_c$	Detection at 10% cost	at 20% cost	at 50% cost
Paint Control	Random	0.503	0.503	9.3%	19.2%	49.3%
Paint Control	Learning to rank	0.732	0.724	45.7%	56.9%	75.8%
Paint Control	Cost-cognizant rank	0.672	0.771	51.0%	64.2%	81.6%
IOF/ROL	Random	0.501	0.501	9.6%	19.3%	48.9%
IOF/ROL	Learning to rank	0.635	0.621	24.6%	37.9%	64.3%
IOF/ROL	Cost-cognizant rank	0.600	0.671	31.7%	45.4%	69.6%

Budgets are fractions of total suite execution cost. Wilcoxon p-values for the two learned variants are below 10^{-3} on both metrics and subjects.

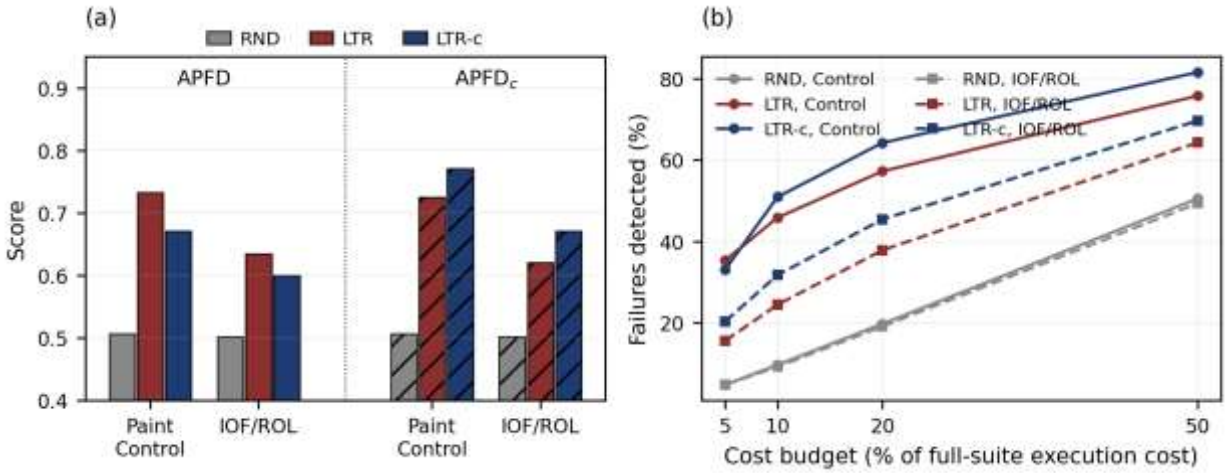


Figure 3. Divergence Between Uniform-Cost and Cost-Cognizant Evaluation on Two Industrial Histories

(a) Mean APFD and $APFD_c$ for random sampling, the learned ranker, and its cost-cognizant variant. Under APFD the learned ranker leads on both subjects, by 0.061 on Paint Control and 0.035 on IOF/ROL; under $APFD_c$ the ordering reverses, with the cost-cognizant variant ahead by

0.047 and 0.050. (b) Share of failures detected as the cost budget grows from 5 to 50 percent of total suite cost, where the cost-cognizant variant leads at every budget beyond 5 percent on both subjects and random sampling tracks the budget fraction.

5. Threats to Validity

The most consequential construct threat is the fault model. Faults here are requirement violations recorded during scenario execution, not defects localised in source code, so a single underlying defect may generate several signatures and several defects may collapse into one. Section 4.1 shows the strategy ranking moves in its middle ranks under a change of accounting rule, and readers should treat the ordering as conditional on the signature definition rather than absolute. The thresholds were fixed before results were inspected, yet remain choices, and a collision-only definition would reduce the fault count per cycle and widen the intervals. A second construct threat follows from the same rule. Because a case–cycle verdict is the union of violations over whatever number of instances the generator happened to record for that configuration, and recorded instance counts range from 1 to 35 with a median of 7, a case's failure probability is partly a function of replay density rather than of the configuration's intrinsic criticality: the per-case correlation between mean recorded instances and failure rate is 0.65, and failing case–cycle pairs average 10.8 recorded instances against 7.0 for passing ones. Any strategy whose ordering correlates with replay count inherits part of this advantage, and because no outcome-free ordering by replay count was evaluated, we cannot rule out that a share of the margin reported for learning to rank reflects replay density rather than the learned failure history itself.

Internal validity rests on a strict prefix split over a fixed cycle order: genuinely chronological on the two industrial CI histories, and a fixed lexicographic order over operational contexts on the driving dataset, which records no acquisition time. Every learned model sees only cycles earlier in that order. It rests further on deterministic seeding, which reproduces all reported figures exactly within a fixed software environment. Because the driving-dataset order is lexicographic rather than chronological, the warm-up window falls entirely on the first road, and the reported margins are conditional on that ordering. The four non-learned strategies are implemented directly in NumPy and reproduce bit for bit, whereas the learned ranker inherits the numerical behaviour of the gradient-boosting implementation and can shift slightly across library versions, without affecting the ordering of strategies or the significance verdicts. The change-aware strategy is the weakest link: no public dataset links source-code diffs to driving scenario outcomes, so the change signal is an operational-context delta and the dependency mapping is a documented heuristic rather than a computed program dependency. Its poor showing is evidence about that substitute signal, not a verdict on change-aware selection practised with real dependency analysis.

External validity is bounded in three ways. The library was generated by search procedures aimed at critical situations, giving a 28.8 percent failure density far above a curated regression suite, which likely compresses the differences between strategies. The suite holds 87

configurations, so a 5 percent budget is four executions and low-budget estimates are coarse. The encoding also omits attributes present in the recorded description, such as crossing direction, so a small number of configurations are indistinguishable to the parameter-space strategies. Cost-cognizant conclusions rest entirely on the two industrial histories; transferring them to graphics-bound simulation needs a corpus that publishes per-scenario runtime.

6. Conclusion

Ordering a driving scenario suite by learned failure history recovers roughly twice as many distinct failure signatures as random sampling at a 5 percent budget and reaches the first failing case at position 1.19 rather than 3.48, while additional-coverage greedy selection is statistically indistinguishable from it once half the suite runs. How much of that gain belongs to the learned history signal itself is not settled by this design, since the union rule that defines a case–cycle verdict makes a case's failure probability partly a function of how many instances were replayed, and no outcome-free replay-count baseline was evaluated to separate the two. The advantage is real and moderate rather than transformative: at half the suite, every strategy except the change-aware variant recovers between 84 and 91 percent of signatures (84.1 percent for random sampling and 90.6 percent for learning to rank), and no strategy makes aggressive reduction free. Two modelling choices that are usually left implicit turn out to govern the answer, since counting failing cases instead of failure signatures demotes coverage-driven selection to the level of random ordering, and scoring with execution cost instead of test position reverses the ranking of the two learned variants on both industrial subjects.

Three extensions follow. A corpus publishing per-scenario simulation runtime would let the cost-cognizant analysis be repeated where compute is actually the constraint. Linking driving stack commits to scenario outcomes would replace the context-delta substitute with genuine dependency information and give change-aware selection a fair test. Repeating the comparison on a curated suite with realistic failure density would establish whether the moderate separations reported here widen when failures are rare.

Declarations

Data and Code Availability. The DeepScenario dataset [15], distributed under GPL-3.0, and the ABB Paint Control and IOF/ROL continuous-integration histories [16] are publicly available under their original licences and are retrieved by the fetch script supplied with the replication package rather than redistributed with this work. The regression-matrix construction, the five strategies, the evaluation harness, and the per-cycle results files from which every table and figure in this paper is produced are provided in that package, which will be deposited in a public archive with a persistent identifier on acceptance. All stochastic components are seeded, so rerunning the harness in a fixed software environment reproduces the reported values.

References

[1] Dosovitskiy, A., Ros, G., Codevilla, F., López, A., & Koltun, V. (2017). CARLA: An open urban driving simulator. Proceedings of the 1st Annual Conference on Robot Learning (CoRL), 1–16.

[2] Haq, F. U., Shin, D., & Briand, L. (2022). Efficient online testing for DNN-enabled systems using surrogate-assisted and many-objective optimization. *Proceedings of the 44th International Conference on Software Engineering (ICSE)*, 811–822.

[3] Yoo, S., & Harman, M. (2012). Regression testing minimization, selection and prioritization: A survey. *Software Testing, Verification and Reliability*, 22(2), 67–120.

[4] Lu, C., Zhang, H., Yue, T., & Ali, S. (2021). Search-based selection and prioritization of test scenarios for autonomous driving systems. Proceedings of the 13th International Symposium on Search-Based Software Engineering (SSBSE), 41–55.

[5] Rothermel, G., Untch, R. H., Chu, C., & Harrold, M. J. (2001). Prioritizing test cases for regression testing. *IEEE Transactions on Software Engineering*, 27(10), 929–948.

[6] Elbaum, S., Malishevsky, A. G., & Rothermel, G. (2002). Test case prioritization: A family of empirical studies. *IEEE Transactions on Software Engineering*, 28(2), 159–182.

[7] Elbaum, S., Malishevsky, A., & Rothermel, G. (2001). Incorporating varying test costs and fault severities into test case prioritization. *Proceedings of the 23rd International Conference on Software Engineering (ICSE)*, 329–338.

[8] Di Nardo, D., Alshahwan, N., Briand, L., & Labiche, Y. (2013). Coverage-based test case prioritisation: An industrial case study. *Proceedings of the 6th IEEE International Conference on Software Testing, Verification and Validation (ICST)*, 302–311.

[9] Chen, T. Y., Kuo, F.-C., Merkel, R. G., & Tse, T. H. (2010). Adaptive random testing: The ART of test case diversity. *Journal of Systems and Software*, 83(1), 60–66.

[10] Yoo, S., Harman, M., Tonella, P., & Susi, A. (2009). Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge. *Proceedings of the 18th International Symposium on Software Testing and Analysis (ISSTA)*, 201–212.

[11] Machalica, M., Samylkin, A., Porth, M., & Chandra, S. (2019). Predictive test selection. *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 91–100.

[12] Herzig, K., Greiler, M., Czerwonka, J., & Murphy, B. (2015). The art of testing less without sacrificing quality. *Proceedings of the 37th International Conference on Software Engineering (ICSE)*, 483–493.

[13] Ben Abdessalem, R., Panichella, A., Nejati, S., Briand, L. C., & Stifter, T. (2018). Testing autonomous cars for feature interaction failures using many-objective search. *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE)*, 143–154.

[14] Fremont, D. J., Dreossi, T., Ghosh, S., Yue, X., Sangiovanni-Vincentelli, A. L., & Seshia, S. A. (2019). Scenic: A language for scenario specification and scene generation. *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 63–78.

[15] Lu, C., Yue, T., & Ali, S. (2023). DeepScenario: An open driving scenario dataset for autonomous driving system testing. *Proceedings of the 20th IEEE/ACM International Conference on Mining Software Repositories (MSR)*, 52–56.

[16] Spieker, H., Gotlieb, A., Marijan, D., & Mossige, M. (2017). Reinforcement learning for automatic test case prioritization and selection in continuous integration. *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 12–22.

[17] Hemmati, H., Arcuri, A., & Briand, L. (2013). Achieving scalable model-based testing through test case diversity. *ACM Transactions on Software Engineering and Methodology*, 22(1), Article 6.

[18] Gligoric, M., Eloussi, L., & Marinov, D. (2015). Practical regression test selection with dynamic file dependencies. *Proceedings of the 2015 International Symposium on Software Testing and Analysis (ISSTA)*, 211–222.

[19] Luo, Q., Hariri, F., Eloussi, L., & Marinov, D. (2014). An empirical analysis of flaky tests. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, 643–653.

[20] Just, R., Jalali, D., Inozemtseva, L., Ernst, M. D., Holmes, R., & Fraser, G. (2014). Are mutants a valid substitute for real faults in software testing? *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, 654–665.

[21] Kim, J.-M., & Porter, A. (2002). A history-based test prioritization technique for regression testing in resource constrained environments. *Proceedings of the 24th International Conference on Software Engineering (ICSE)*, 119–129.

[22] Epitropakis, M. G., Yoo, S., Harman, M., & Burke, E. K. (2015). Empirical evaluation of Pareto efficient multi-objective regression test case prioritisation. *Proceedings of the 2015 International Symposium on Software Testing and Analysis (ISSTA)*, 234–245.

[23] Amini, M. H., Naseri, S., & Nejati, S. (2024). Evaluating the impact of flaky simulators on testing autonomous driving systems. *Empirical Software Engineering*, 29(2), Article 47.

[24] Guo, Y. (2026). A Multi-Dimensional Coverage Metric with Evolutionary Search for Safety-Critical Scenario Generation in Autonomous Driving Testing. Journal of Science, Innovation & Social Impact, 2(2), 82-93.

[25] Li, Y. (2026). Benchmarking Robustness-Efficiency Trade-offs of Camera-LiDAR Fusion Strategies for 3D Object Detection Under Environmental Corruptions. Journal of Science, Innovation & Social Impact, 2(2), 104-116.

[26] Wu, C., Li, Y., Li, L., Wang, L., & Liu, Y. (2020). Caption Generation from Road Images for Traffic Scene Construction. In 2020 IEEE Intelligent Vehicles Symposium (IV) (pp. 1271-1276). IEEE.

[27] Li, Y., Wu, C., Li, L., Liu, Y., & Zhu, J. (2021). Caption generation from road images for traffic scene modeling. IEEE Transactions on intelligent transportation systems, 23(7), 7805-7816.

[28] Guo, Y. (2025). Performance Evaluation of Lightweight Detection Algorithms on Compact LiDAR-Camera Configurations for Freight Transportation. Journal of Science, Innovation & Social Impact, 1(1), 398-409.

[29] Guo, Y. (2025). Reliability Assessment and Adaptive Fusion Algorithm for Multi-Sensor Data in Autonomous Driving under Adverse Weather Conditions. Journal of Sustainability, Policy, and Practice, 1(4), 143-155.

[30] Wu, X., Li, Y., Liu, Y., Pang, S., Wang, L., Wu, C., & Huo, H. (2019). Jointly detecting and retrieving vehicles from road image sequences based on CNN. In 2019 IEEE Intelligent Vehicles Symposium (IV) (pp. 2530-2535). IEEE.

[31] Li, Y. (2025, December). Comparative Analysis of Illumination Normalization Methods for Autonomous Driving Under Challenging Lighting Conditions. In Proceedings of the 2025 6th International Conference on Computer Science and Management Technology (pp. 633-639).

[32] Huo, H., Li, Y., Wu, C., Wu, X., Tian, Z., & Liu, Y. (2019). Semantic segmentation and scene reconstruction for traffic simulation using CNN. In 2019 2nd China Symposium on Cognitive Computing and Hybrid Intelligence (CCHI) (pp. 117-122). IEEE.