

FDRRestTest: Feedback-Driven Logical Testing for REST APIs

Yutong Huo 

China University of Geosciences, Wuhan, China

Dongcheng Li¹ 

Cal Poly Humboldt, Arcata, CA, USA

Abstract

Background. REST APIs have become the primary interaction interface for modern software systems, making automated black-box API testing increasingly important. However, existing testing techniques still face three major limitations: over-reliance on dependencies extracted from static specifications, failure to fully utilise failed requests as feedback, and an overemphasis on crash-oriented verification, resulting in insufficient exploration of non-crash-prone logical defects.

Aims. We design, implement and evaluate a feedback-driven black-box testing framework that turns runtime feedback from a passive outcome into an active control signal, maximising logical-defect yield within practical budgets.

Method. We propose FDRRESTTEST, which integrates OpenAPI-based static semantic bootstrapping, runtime dependency evolution, a 7-category failure diagnosis and repair loop, a multi-dimensional logical oracle engine and a budget-aware utility scheduler in one closed loop. Under a fixed 600-second time budget, we evaluated FDRRESTTEST against five state-of-the-art baselines (RESTler, EvoMaster, Morest, ARAT-RL, AUTORESTTEST) on 12 real-world REST services along five protocols: fixed-time cross-tool comparison, fixed-request cross-tool replication, ablation across three dependency profiles, single-host budget sensitivity, and per-service comparison; significance via two-sided paired Wilcoxon signed-rank tests with Holm-Bonferroni correction, magnitude via standardised paired effect sizes.

Results. FDRRESTTEST improves fault-revealing efficiency (FRE) by up to 6.3% over the strongest baseline, AUTORESTTEST (0.9878 vs. 0.9294; an upper estimate, as AUTORESTTEST's FRE is a post-hoc lower bound), while issuing 6.3–11.8× fewer requests than AUTORESTTEST across the 12 services (8.3× on Features Service). The +12.2% logical-yield advantage holds on every one of the 12 services (+7.0%–+16.5%, geometric mean +12.4%), and the FRE gain is consistent across all 12 in per-service means and statistically significant ($p_{\text{adj}} < .005$, $d_z = 3.73$ on logical bug yield, *LB*Y). An ablation across six subjects spanning three dependency profiles confirms no single component accounts for the advantage, the oracle contributing most – removing it drops FRE below the strongest baseline on all six – while a Pareto-scheduling variant is 4.4% below the weighted utility on the most state-rich host. A manual confirmation study ($N = 120$, stratified over the 236 deduplicated findings of all six tools, 30 per oracle category) confirms 96/120 (80%) as true defects (Cohen's $\kappa = 0.78$), supporting the oracle-finding interpretation of the *LB*Y gain.

Conclusions. Black-box REST API testing heavily benefits from a tightly coupled closed feedback loop. Within this loop, runtime dependency evolution, failure repair, logical-defect-aware validation, and budget-aware scheduling are each necessary; quantifying their interaction would require multi-component removals, which we leave to future work.

2012 ACM Subject Classification Software and its engineering → Software testing and debugging

Keywords and phrases REST API testing, black-box testing, logical defects, feedback-driven testing, oracle-based validation

Digital Object Identifier 10.4230/LIPIcs.ESEM.2026.20

¹ Corresponding author.



© Yutong Huo and Dongcheng Li;
licensed under Creative Commons License CC-BY 4.0

20th International Symposium on Empirical Software Engineering and Measurement (ESEM 2026).

Editors: Robert Feldt, Maria Paasivaara, Daniel Mendez, Stefan Wagner, and Marvin Muñoz Barón;
Article No. 20; pp. 20:1–20:20



Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Category Technical Track Paper

Supplementary Material *Software (Source Code)*: <https://doi.org/10.5281/zenodo.21964240>

1 Introduction

REST APIs serve as the backbone of modern cloud services, microservice platforms, and Internetware systems [10]. Unlike conventional input-output testing it is intrinsically stateful: many operations are meaningful only after prior requests have created or updated the resources they need, so automation must solve dependency inference, sequence orchestration, feedback interpretation, and response validation as well as input generation.

Recent black-box REST API testing has evolved along five paradigms: stateful fuzzing [4], search-based generation [2], dependency-graph-guided exploration [24], learning-based prioritisation [17, 20, 18], and logical-oracle validation [37]; see [11, 38] for surveys.

Three limitations remain. **First**, dependency knowledge is still largely static or only weakly updated at runtime, although many dependencies become visible only after execution (e.g., a returned identifier reusable by another endpoint under a different field name). **Second**, failed requests are treated as penalties or pruning signals, although many carry repairable information: missing dependencies, malformed fields, stale states, inconsistent parameter combinations. **Third**, validation remains crash-centric and misses non-crash logical defects.

We address these by proposing FDRRESTTEST, a feedback-driven black-box framework for logical-defect-aware REST API exploration. The core idea is to elevate runtime feedback from a passive outcome into an active control signal: FDRRESTTEST builds a semantic dependency graph from the OpenAPI specification, updates it incrementally during execution, diagnoses and repairs failed requests, validates responses with multi-dimensional oracles, and allocates budget through a utility-based scheduler.

FDRRESTTEST differs from the closest prior systems [24, 20, 37] along concrete axes rather than generic “integration”: Morest updates a model graph from runtime responses but its oracle layer remains crash-oriented; AutoRestTest couples semantic graphs with large language model (LLM)-driven values but treats failures as negative reinforcement; LogiAgent provides scenario-aware logical testing but lacks runtime dependency evolution and budget-aware scheduling (Section 6). FDRRESTTEST is the only system driving all four—dependency refinement, failure repair, multi-dimensional logical oracles, budget-aware scheduling—from one runtime feedback loop. The ablation is the empirical counterpart of that claim: over six subjects spanning three dependency profiles, removing the oracle drops fault-revealing efficiency below the strongest baseline on all six, the exploration-side removals on three to six (Subsubsection 4.3.2), a pattern that fits mutual reinforcement inside the loop – though single-removal designs establish necessity without quantifying interaction.

The contributions are: (i) a closed-loop black-box framework integrating dynamic dependency evolution, failure repair, logical validation, and budget-aware scheduling; (ii) a runtime-evolving dependency graph beyond static OpenAPI analysis; (iii) a feedback-driven diagnosis-and-repair loop with an explicit 7-category failure taxonomy and prioritised repair actions; (iv) a multi-dimensional oracle framework; and (v) an empirical study across five protocols (fixed-time cross-tool comparison, fixed-request cross-tool replication, ablation, budget sensitivity, per-service analysis) on 12 services, with service-level Wilcoxon tests, Holm-Bonferroni adjustment, standardised paired effect sizes, and a complete artefact for reproducibility.

2 Background and Motivation

Existing dependency-aware approaches [4, 24, 17, 20] infer producer-consumer relations from the specification, but such relations are often incomplete or noisy when behaviour depends on runtime state or undocumented rules [25, 1]: *semantic mismatch* (**name** vs. **productName**), *specification incompleteness* (an optional parameter is in practice required), and *runtime-emergent dependencies*. A static-only strategy therefore misses operation chains or wastes budget on dynamically failing sequences. Validation is likewise crash-oriented, yet a 2xx is not semantic correctness: a delete may report success while the resource remains queryable, or a declared constraint may be silently accepted. LogiAgent [37] highlighted this gap, but logical-defect-aware testing remains under-integrated into closed-loop, budget-aware workflows.

Motivating example. In a feature-management service, the product-creation endpoint returns a generated identifier that a later configuration endpoint requires, but the specification does not document the relation, so a static model cannot connect the two; the resulting 400 or 404 carries diagnostic clues—wrong identifier variant, stale resource, missing prerequisite—that a “dead-end” strategy discards. Effective testing therefore requires runtime-evolving dependency knowledge, failure-aware diagnosis and repair, and logical-defect-aware validation—objectives O1–O3 of Section 3.

3 Method

3.1 Overview, Design Objectives, and Architecture

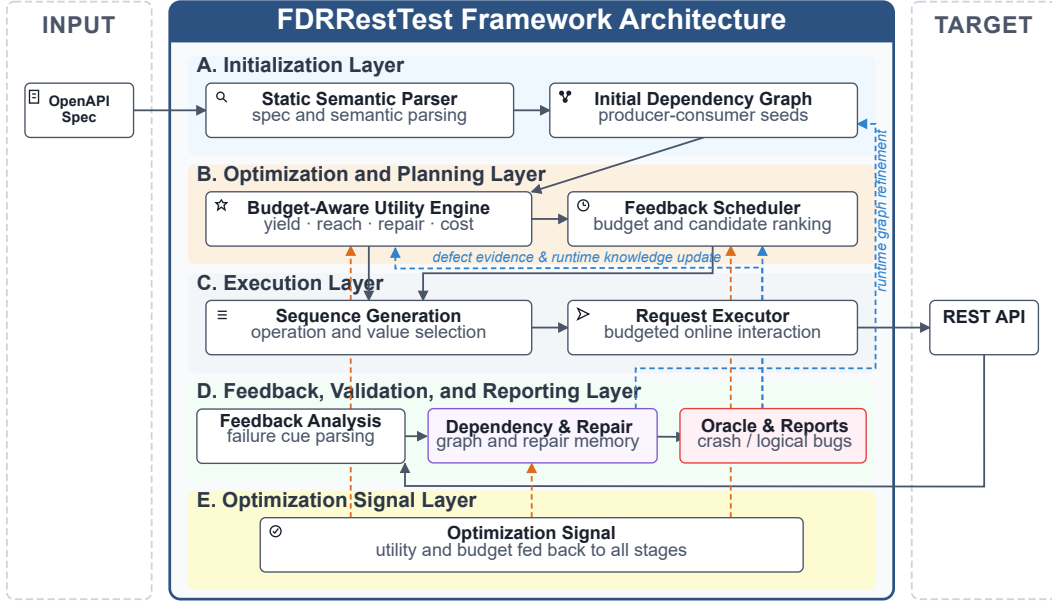
FDRRESTTEST is designed around four objectives from Section 2: **O1 runtime dependency discovery** – find inter-operation relations beyond what OpenAPI specifies; **O2 feedback reutilisation** – convert failed requests into diagnosis and repair signals rather than penalties; **O3 logical-defect-aware validation** – detect non-crash anomalies such as state inconsistency and cross-endpoint contradictions; **O4 budget-efficient exploration** – maximise fault-revealing utility under fixed budgets rather than request volume. Each maps to one term of the scheduler utility (Subsection 3.7): O1→*Reachability* (Subsection 3.4), O2→*Repairability* (Subsection 3.5), O3→*DefectYield* (Subsection 3.6), O4→*Cost*.

Figure 1 shows the five-layer workflow; runtime responses feed three analyses—dependency evolution, repair, oracle validation—whose outputs update both the graph and the scheduler in a closed loop.

3.2 Problem Formulation

Given the operations $\mathcal{O}$ of a specification, a test sequence is an ordered list of executable requests; the goal is to maximise fault-revealing utility under a fixed budget B .

Terminology. Four terms are used consistently: a *failure* is an observable deviation at the API boundary for one request; a *fault* is the cause inside the implementation, inferred only through failures; an *anomaly* is a raw pre-deduplication oracle signal; a *defect finding* is a deduplicated anomaly, and becomes a *confirmed defect* only after the manual validation of Subsubsection 4.3.3. Outcomes are of three types: **crash failure signals** (typically unexpected 5xx), **logical anomaly signals** (structural, state, consistency, or constraint violations), and **repairable failures**. *Logical Bug Yield* and *Fault-Revealing Efficiency* count oracle-triggering signals in their numerators, so they measure signal density per request rather than confirmed faults.



■ **Figure 1** Overall architecture of FDRRESTTEST; the five layers A–E are labelled in the figure.

3.3 Static Semantic Bootstrapping

FDRRESTTEST parses the OpenAPI specification (operations, parameters, response schemas and examples, descriptions, field names) into an initial weighted semantic dependency graph $G_0 = (V, E_0, W_0)$: nodes are operations, edges potential producer-consumer relations, and W_0 holds confidence weights from lexical similarity (threshold τ_{lex}), type compatibility, parameter-response matching and schema-level field overlap. The graph is not ground truth but a prior, continuously corrected by runtime evidence.

3.4 Dynamic Dependency Evolution

A static graph cannot capture runtime semantics, so FDRRESTTEST updates it after each execution; at step t it is $G_t = (V, E_t, W_t)$. Reusable response values are matched against other operations' inputs by field-name and description similarity, type and value-shape compatibility, and the success history of similar reuses; a new relation inserts an edge, and an exercised edge is re-weighted by its downstream outcome:

$$W_{t+1}(e) = \begin{cases} \min(1, W_t(e) + \alpha_r), & \text{if reuse succeeds,} \\ \max(w_{\min}, W_t(e) - \beta_p), & \text{if reuse fails.} \end{cases} \quad (1)$$

Here α_r , β_p , and $w_{\min}$ are the reward, penalty, and confidence-floor factors; together with τ_{lex} , the scheduler weights $\lambda_1 - \lambda_4$, and the repair-memory capacity $|\mathcal{M}|$, all nine hyperparameters are fixed across the 12 subjects after pilot tuning on the held-out *Scout API* (Subsubsection 4.2.4). The graph thus shifts from static speculation toward execution-grounded knowledge.

3.5 Failure Diagnosis and Repair Loop

FDRRESTEST treats failures as diagnosis opportunities. The 7-category taxonomy: (1) *missing dependency*; (2) *malformed parameter*; (3) *unsatisfied constraint*; (4) *stale state*; (5) *auth issue* (typically 401/403); (6) *unsupported method/route* (typically 405/404); (7) *server-side or undiagnosable failure* (5xx, and any failure for which (1)–(6) lack sufficient evidence).

Taxonomy derivation. The taxonomy was not designed a priori: we split the response space by status-code semantics (4xx vs. 5xx), then refined the 4xx side by open-coding failure traces from pilot runs on the held-out *Scout API*, grouping failures that admit the same corrective action and iterating until no pilot failure fell outside the categories. Each category is therefore tied to a distinct repair repertoire rather than to a status code: a 404 may be diagnosed as *missing dependency*, *stale state*, or *unsupported route*, and the three trigger different repairs.

Diagnosis and ambiguity handling. Diagnosis is rule-based first-match over a fixed order (1)→(7) on three evidence sources: the status code, response-body keywords (e.g., "not found", "expired", validator-echoed constraint names), and dependency-graph context (whether a producer edge for the failing parameter exists and was recently exercised). Insufficient evidence conservatively assigns category (7), which triggers no repair, trading repair-loop recall for avoiding wasted retries on undiagnosable failures.

Repair-action selection. For categories 1–6 we apply a prioritised repair repertoire: (a) value substitution from runtime memory (≤ 2 retries); (b) prerequisite insertion (1 retry); (c) sequence reordering (1 retry); (d) resource reconstruction (1 retry); (e) constrained mutation within schema bounds; (f) negative-context avoidance (record failing combinations in a negative-context cache $\mathcal{K}$). Actions are attempted in order, filtered by a category-action compatibility map (e.g., *missing dependency* admits (b)–(d), *malformed parameter* (a) and (e)); the first successful retry ends the episode, and the per-action caps bound repair overhead per failure. A first-in-first-out (FIFO) repair memory $\mathcal{M}$ (Subsubsection 4.2.4) stores recent failures, categories and successful corrections for cross-request reuse; the executor records the diagnosed category, attempted actions, and outcome for every failure.

Taxonomy validation. Over the 12 subjects (324,868 diagnosed failures), category (7), the fallback that triggers no repair, absorbs 71.2%: the rules send every 5xx and quota rejection there unconditionally, so its share is bounded below by their volume, which this measurement meets exactly (231,214 of 324,868) – no 4xx needed the fallback. The six repairable categories share the remaining 28.8%; their within-subset shares, success rates and per-action tallies are in the replication package. Success rates order as the design predicts: the two categories admitting a deterministic correction lead (*auth issue* 69.6%, *missing dependency* 61.9%), those needing a search over values or states follow (55.4% to 44.3%), and *unsupported method/route* succeeds only 14.8%, as most such failures reflect an operation the service does not expose. The taxonomy is thus validated twice: every category is populated at a non-trivial rate, and the categories separate failures by corrective action, not by status code.

3.6 Multi-Dimensional Oracle Engine

A 5xx-only oracle is insufficient for logical testing; FDRRESTTEST uses five oracles (Table 4 breaks down the four non-crash families; the 7-category repair taxonomy of Subsection 3.5 is a different partition of the same evidence: it characterises *failures* triggering repair, not responses): **Crash** – unexpected 5xx; **Structure** – response-body schema conformance (illustrative: GET /products/{id} returning 200 without required createdAt); **Constraint** – response consistent with request constraints (illustrative: "code": "xxxxx" accepted despite maxLength: 4); **State** – lifecycle anomalies, recorded against the write, not the read that exposed it (the one exception is stale_read_after_write, recorded against the read, which names the preceding write); **Consistency** – cross-endpoint mismatch (list returns name="A" for id 42, detail returns name="B").

Each oracle produces a defect signal with contextual evidence. Signals are deduplicated by (endpoint, oracle type, semantic fingerprint), where the fingerprint is a normalised hash of the response structure skeleton (key names with type tags), the violated constraint identifier (e.g., type@offset; the artefact enumerates the identifiers it uses), and the failing oracle predicate; equivalent failures triggered by different values collapse into one entry, while different failure shapes stay apart. Deduplication is applied uniformly to the logs of all five baselines, which we re-evaluate with the shared harness, while FDRRESTTEST–whose oracle layer is the contribution under study–is reported with its own deduplication (Subsubsection 4.2.3).

3.7 Budget-Aware Utility-Based Test Scheduling

The framework prioritises future exploration with a utility function over several objectives:

$$U(s) = \lambda_1 \cdot \text{DefectYield}(s) + \lambda_2 \cdot \text{Reachability}(s) + \lambda_3 \cdot \text{Repairability}(s) - \lambda_4 \cdot \text{Cost}(s), \quad (2)$$

where $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ are tunable non-negative weights fixed across all experimental settings after pilot tuning on a held-out service; values are reported in Subsubsection 4.2.4. **DefectYield** rewards sequences that previously exposed crash or logical signals; **Reachability** favours yet-unreached or under-exercised operations; **Repairability** prioritises paths whose failures appear recoverable, by repair-memory hit rate. **Cost** penalises high-overhead sequences (length above running median), repeated invalid trials (first request matching the negative-context cache), and low-yield repetitions (zero DefectYield over the last $w = 20$ requests); the three sub-terms are equally weighted, normalised to $[0, 1]$ and scaled by λ_4 .

Why weighted scalarisation rather than Pareto-based scheduling. The scheduler optimises multiple, potentially conflicting factors through a single weighted utility, i.e., a linear scalarisation, not a Pareto-front search. This is a deliberate trade-off: the decision recurs at every loop iteration on a changing candidate set, and maintaining a non-dominated front at that frequency competes for budget with exploration itself. Because a fixed weight vector selects one point on the trade-off surface, we test the decision empirically: a **Pareto** variant replacing the weighted utility with non-dominated sorting over the four factors (crowding distance breaking ties), every other module unchanged, is a sixth ablation variant on the single ablation host, quantified in Table 3; weight sensitivity is in Section 5.

The five components integrate into a single loop: the scheduler selects the next target from the current graph, repair memory and negative-context cache ($G_t, \mathcal{M}, \mathcal{K}$); a dependency-aware sequence is planned, instantiated, and executed; responses update the graph and store, trigger diagnosis and repair, and are evaluated by the oracles with fingerprint deduplication; the utility and remaining budget are then updated.

FDRRESTTEST comprises seven cooperating modules (parser, graph manager, scheduler/-planner, executor, repair engine, oracle engine, dedup/reporting); request- and sequence-level traces capture payloads, responses, diagnosed categories, repair attempts and oracle results, and feed the post-hoc dedup pipeline (all in the artefact).

Implementation basis. FDRRESTTEST is built on the open-source AUTORESTTEST code-base [20], from which it inherits the specification parser, the LLM interface and prompt layer, the multi-agent value-generation machinery, and the execution and reporting infrastructure. The four mechanisms of Subsection 3.4–Subsection 3.7 are new components on top of that basis; the original licence and attribution are preserved in the artefact. This changes how the AUTORESTTEST comparison should be read, in a direction that strengthens it (Subsubsection 4.2.2).

4 Evaluation

4.1 Research Questions

The empirical unit here is a controlled comparison, not a tool description: six tools (five baselines plus FDRRESTTEST) run on 12 heterogeneous REST services under five protocols – the ablation covering six of the 12 services, the budget sweep a single host – with $N=10$ repetitions per cell, service-level paired inference and a manual confirmation study of the findings. The framework of Section 3 is the treatment under study; the questions below concern what can be measured about it and under which conditions those measurements hold.

To systematically evaluate the effectiveness of FDRRESTTEST, we investigate the following research questions: **RQ1** – does FDRRESTTEST improve fault-revealing efficiency under fixed testing budgets compared with representative black-box REST API testing baselines? **RQ2** – how much do dynamic dependency evolution, failure repair, the multi-dimensional oracle, and the scheduler each contribute to the overall effectiveness, and does the contribution pattern hold across dependency profiles? **RQ3** – can FDRRESTTEST detect more non-crash logical defects than crash-oriented baselines, and what is the precision of these findings? **RQ4** – how stable is FDRRESTTEST under different testing budgets? **RQ5** – how does FDRRESTTEST perform across different services, and on which service profiles does the repair loop contribute most?

4.2 Experimental Methodology

4.2.1 Subjects

We evaluate FDRRESTTEST on 12 REST services spanning open-source systems and public-facing APIs (Table 1; sources and pinned revisions in the table note). The subjects differ substantially in operation scale, dependency structure, parameter constraints, and semantic complexity. Requests issued per service are reported in Table 5; we use requests, not test cases, as the volume unit: the tools group requests into sequences under incomparable definitions, so a sequence count is not a common denominator.

4.2.2 Baselines

We compare FDRRESTTEST against representative black-box REST API testing tools: **RESTler** [4] (stateful grammar-based fuzzing), **EvoMaster** [2] (search-based, black-box mode), **Morest** [24] (dependency-aware testing with execution feedback), **ARAT-RL** [17]

■ **Table 1** REST services used in the evaluation. N/A denotes services for which lines of code cannot be reported because the service is a public-facing or closed-source API.

REST Service	LoC	#Ops	REST Service	LoC	#Ops
Features Service	1688	18	Market	7945	13
Language Tool	113170	2	Project Tracking System	3784	67
Rest Countries	1619	22	YouTube-Mock	2422	1
Genome Nexus	22143	23	FDIC	N/A	6
Person Controller	601	12	Spotify	N/A	12
User Management	2805	22	Ohsome API	N/A	122

Sources. Seven subjects (Features Service, Language Tool, Rest Countries, Person Controller, User Management, Market, YouTube-Mock) come from the EvoMaster Benchmark (EMB) <https://github.com/EMResearch/EMB> as deployed in the ARAT-RL and AutoRestTest harnesses [17, 20]; Genome Nexus <https://github.com/genome-nexus/genome-nexus> and Project Tracking System <https://github.com/SelimHorri/Project-Tracking-System> from upstream, all cloned on 2026-02-09 at that day’s HEAD. The public APIs are FDIC <https://banks.data.fdic.gov>, Spotify <https://api.spotify.com> and Ohsome <https://api.ohsome.org>. For Market and Project Tracking System we use the OpenAPI document shipped with the harnesses, not the upstream one; for Spotify we drive a 12-operation subset of the provider’s published Web API, so its *OpR* (Subsubsection 4.2.3) is relative to that subset. Counts are the operations the deployment exposes; User Management’s include the six method variants of Spring Boot’s default */error*.

(reinforcement-learning-based testing), and **AutoRestTest** [20] (LLM-enhanced multi-agent testing), covering four of the five paradigms of Section 1; the logical-oracle paradigm is represented by LogiAgent, excluded from the main comparison for the reason below.

Status of the AutoRestTest comparison. FDRRESTTEST is implemented on the AUTORESTTEST codebase, so the two configurations differ only in the four mechanisms under study; HTTP client, request serialisation, LLM provider and prompting, and reporting are identical. That controlled contrast removes the implementation confounders that ordinarily limit tool-to-tool comparison, and is why AUTORESTTEST is the primary reference. The other four baselines are independent implementations and retain those confounders; *SEY*, identical across all six tools, is the fairness anchor there (Subsubsection 4.2.3).

LogiAgent [37] is excluded from the main comparison because its LLM-as-agent pipeline cannot be fairly matched under a fixed time budget (per-request cost is dominated by external LLM latency). A matched-token-budget pilot on Features Service ($N=1$) reached $FRE = 1.205$, between AUTORESTTEST (1.1742) and FDRRESTTEST (1.2586); having no dispersion it supports no test and we draw no inference from it, and its log is not in the replication package. A full $N=10$ head-to-head is future work.

4.2.3 Metrics

We use five black-box efficiency-oriented metrics, chosen because (i) they are computable from externally observable behaviour alone, putting closed- and open-source subjects on equal terms; (ii) the yield metrics are per-request, decoupling effectiveness from raw throughput when request rates differ by an order of magnitude (Table 5); and (iii) they cover operation coverage (*OpR*), input validity (*RVR*), crash detection (*SEY*) and logical-defect detection (*LBY*), with *FRE* as the primary combined endpoint. The five metrics are: **Server Error Yield** $SEY = N_{\text{use}}/N_{\text{req}} \times 100\%$; **Logical Bug Yield** $LBY = N_{\text{lbs}}/N_{\text{req}} \times 100\%$; **Fault-Revealing Efficiency** $FRE = (N_{\text{use}} + N_{\text{lbs}})/N_{\text{req}}$; **Operation Reachability** $OpR =$

$N_{\text{proc}}/N_{\text{ops}} \times 100\%$; and **Request Validity Rate** $RVR = (N_{2xx} + N_{3xx})/N_{\text{req}} \times 100\%$, where N_{use} , N_{lbs} , N_{req} , N_{proc} , and N_{ops} count server-error responses, logical anomaly signals, issued requests, successfully processed operations, and total specified operations. Following prior work we report *SEY* and *LBY* as percentages and *FRE* as a per-request ratio, so $FRE = (SEY + LBY)/100$ holds row-wise wherever the three appear together; fourth-decimal residuals are rounding of the two component percentages.

Counting basis of the yield metrics. The numerators of *SEY*, *LBY* and *FRE* count *oracle-triggering signals*: one response may violate several oracles, so *FRE* can exceed 1.0. Neither numerator passes through the cross-request deduplication of Subsection 3.6: both count every triggering response, because a per-request efficiency measure must charge a tool for the budget each signal cost it. *SEY* is consequently dominated by how often a subject answers with a server error, a property of the subject: it separates tools only through how much of the subject’s surface they exercise, and equalises once two tools reach the same surface – as FDRRESTTEST and AUTORESTTEST do (Subsubsection 4.3.1). It therefore serves as a fairness check between comparable tools, not as a discriminator. Table 4 is the complementary view – signals sharing an ⟨endpoint, oracle type, semantic fingerprint⟩ triple collapse into one entry there – and the two cannot be converted into one another. Recurrence therefore raises both yields by design, while Table 4 carries *how many* distinct defects each tool finds.

Fairness of *LBY*. *LBY* is oracle-dependent: because the four non-crash oracles are a contribution of FDRRESTTEST and not a feature of the baselines, Table 2 applies the shared five-oracle harness post-hoc to every baseline’s logs; *SEY*, which depends only on the crash oracle, is the primary cross-tool fairness anchor. The baselines’ raw logs and as-published oracle outputs are not in the artefact, which records the harness output; the re-scoring can therefore be re-counted but not re-derived. The lower-bound direction is not a measured comparison: the harness can score only what each log preserves, and the online oracles of Subsection 3.6 read session history and cross-endpoint context that a replayed log does not carry. Table 6 separately shows the harness families to be a superset of Morest’s and AUTORESTTEST’s, so the recovered counts do not fall below what those two self-report either.

No ground-truth defect list. The 12 subjects carry no curated list of known defects, so no metric here is computed against a ground truth; every defect quantity is an oracle output – raw per-request signals (the yield metrics) or deduplicated findings (Table 4) – whose precision is probed empirically by the manual confirmation study of Subsubsection 4.3.3.

4.2.4 Experimental Setup

All tools share one environment (deployments, access configs, authentication, network, budgets); unless stated the budget is 600s, and RQ4 varies it to rule out single-protocol artefacts. Each cell is repeated $N = 10$; Table 2, Table 3 and Table 5 report means over those repetitions, while Table 4 reports unions over them (Subsubsection 4.2.3); run-to-run dispersion is bounded (over the 24 FDRRESTTEST/AUTORESTTEST cells, interquartile range IQR_{FRE} at Tukey hinges: mean 0.041, max 0.074). Deduplication follows Subsection 3.6; crash failures by ⟨response signature, operation context⟩. Authentication uses pre-provisioned static credentials supplied identically to every tool.

Hyperparameters. All nine are fixed across subjects after grid-search pilot tuning (objective *FRE*) on a held-out service, the *Scout API* of the EMB suite (`cs/rest/scout-api`): an open-source multi-step-state create-read-update-delete (CRUD) service disjoint from the 12 subjects. Values: lexical similarity threshold $\tau_{\text{lex}}=0.70$; edge-reuse reward, penalty and confidence floor $\alpha_r=0.15$, $\beta_p=0.10$, $w_{\text{min}}=0.05$; scheduler weights $\lambda_1-\lambda_4 = 0.40/0.30/0.20/0.10$ for DefectYield, Reachability, Repairability, Cost (Subsection 3.7); repair-memory capacity $|\mathcal{M}|=200$, FIFO (Subsection 3.5).

Statistical analysis. $\alpha=.05$ throughout. All *cross-tool* inferential tests are performed at the *service* level, on the 12 paired per-service means of Table 5, because between-service heterogeneity is the dominant confounder for generalisation claims; every statistic below is recomputable from that table or from the per-service traces in the replication package. We use the two-sided paired Wilcoxon signed-rank test [35] with Holm-Bonferroni adjustment [13] across the family of five cross-tool comparisons; secondary tests (*SEY*, the Pareto variant, the fixed-request caps) are reported unadjusted and are exploratory. We quantify magnitude with the standardised paired difference d_z , computed on the 12 service-level paired means, not on individual runs, and with the distribution of per-service margins, following the effect-size guidance of Arcuri and Briand [3]. Magnitude is reported on *LBY* rather than *FRE*, because the common-mode *SEY* term inflates *FRE* equally for every tool and can only dilute an effect size; significance is unaffected, the sign pattern being identical on both, so the tests stay on *FRE*.

Because FDRRESTTEST’s per-service mean *FRE* exceeds every baseline’s on all 12 subjects, each pairwise signed-rank statistic attains its extreme value ($W^+=78$), so the exact two-sided p is $2/2^{12} = .0005$ per comparison and $p_{\text{adj}} < .005$ for all five after Holm. The exact p is fixed by the sign pattern alone; for magnitude we report, against AUTORESTTEST, a paired mean *LBY* difference of 5.83 points (standard deviation $SD = 1.57$, $d_z = 3.73$) and per-service relative *LBY* margins with median 12.4% and range [7.0%, 16.5%]. A sensitivity analysis for the paired t -test ($N=12$, $\alpha=.05$, power = .80) gives a minimum detectable effect of $\approx 0.89\sigma$; the signed-rank test we use has slightly lower asymptotic efficiency, so this is a lower bound on it.

Scope of effect-size reporting. Service-level tests are reported for all five cross-tool comparisons; d_z and the per-service margin distribution only for RQ1, against the strongest baseline. Within each host, where the $N=10$ repetitions make run-level comparison meaningful, magnitude is Cliff’s δ [5] on *LBY* between the two tools’ independent 10-run distributions. For RQ1, δ against AUTORESTTEST ranges over [0.96, 1.00] across the 12 services (mean 1.00) and equals 1.00 on every service against the other four baselines. For RQ2, every removal yields $\delta = 1.00$ on the host of Table 3 and $\delta \geq 0.42$ across all six ablation subjects, the minimum falling on No-Repair, whose effect is the one the ablation itself reports as host-dependent; per-subject values and the within-host Pareto comparison are in the replication package. For RQ4 every budget step yields $\delta = 1.00$ on the reported host, and $\delta \geq 0.54$ on the three further sweep subjects in the package. Run-level observations are never pooled *across* services, where between-service variance would dominate.

4.3 Results and Analysis

We present the results in the order of the five research questions: overall effectiveness against all baselines (RQ1), component contribution by ablation (RQ2), non-crash logical defect detection (RQ3), robustness across testing budgets (RQ4), and per-service behaviour with repair impact (RQ5).

■ **Table 2** Overall comparison under a fixed 600-second budget, averaged across 12 services. “Req./Sig.” is $1/FRE$, requests per oracle-triggering signal, not per distinct defect (Subsubsection 4.2.3). Last column: Holm-Bonferroni-adjusted p_{adj} from the two-sided paired Wilcoxon test on FRE ; “†” marks $p_{adj} < .05$; effect sizes in Subsubsection 4.2.4. LBY and FRE for the five baselines are recovered post-hoc with the shared harness and are therefore lower bounds, so their Req./Sig. are correspondingly upper bounds. Only FDRRESTTEST’s LBY comes from its own oracle layer; the AUTORESTTEST contrast stays controlled on every other axis (parser, LLM interface, value generation, execution, reporting), which is why it is the primary reference.

Tool	<i>OpR</i> (%)	<i>SEY</i>	<i>LBY</i>	<i>FRE</i>	Req./Sig. ↓	p_{adj}
RESTler	42.5	24.98	4.54	0.2952	3.39	< .005 †
EvoMaster	67.2	35.94	8.85	0.4479	2.23	< .005 †
Morest	75.1	40.87	11.15	0.5202	1.92	< .005 †
ARAT-RL	78.7	44.81	14.55	0.5936	1.68	< .005 †
AUTORESTTEST	82.5	45.26	47.68	0.9294	1.08	< .005 †
FDRRESTTEST	93.3	45.26	53.52	0.9878	1.01	–

4.3.1 RQ1: Overall Effectiveness

Table 2 compares FDRRESTTEST with all baselines under the same fixed 600-second budget, in terms of *OpR*, *SEY*, *LBY*, *FRE*, and requests per oracle-triggering signal, averaged across the 12 services.

FDRRESTTEST achieves the highest *FRE* (0.9878 vs. 0.9294 for AUTORESTTEST, +6.3% relative, an upper estimate: AUTORESTTEST’s *LBY*– and hence its *FRE*– is a post-hoc lower bound) and correspondingly the fewest requests per oracle-triggering signal (1.01 vs. 1.08); *SEY* is indistinguishable from AUTORESTTEST (45.26 vs. 45.26, discussed below) while *LBY* is substantially higher (53.52 vs. 47.68, +12.2%; because AUTORESTTEST’s *LBY* is a post-hoc lower bound, this margin is an upper estimate of the true gap), and *OpR* is the highest at 93.3% (+10.8 percentage points (pp) above AUTORESTTEST, +14.6 pp above ARAT-RL). Paired Wilcoxon on per-service *FRE* gives $p_{adj} < .005$ for all five pairwise comparisons, with $d_z = 3.73$ on *LBY* against AUTORESTTEST (Subsubsection 4.2.4).

SEY behaves exactly as the fairness anchor it is meant to be. Over the 12 services the two tools are indistinguishable on it: a paired mean difference of +0.004 points, FDRRESTTEST ahead on 6 of 12 subjects, two-sided Wilcoxon $p = .57$. That is what a quantity governed by the crash oracle – identical across all six tools – should look like once it is charged per request, and it holds under equal request volume too (+4.1, +2.1 and +0.3 points at the three caps, Subsubsection 4.3.6). *SEY* is therefore a common-mode term: it measures how often a subject answers with a server error, a property of the subject rather than of the search.

The crash dimension separates the tools only after deduplication: the union of crash triples is 44/54/68/47/53/71 over the six tools in the order of Table 2 – identical *SEY*, yet 34% more distinct crashes than AUTORESTTEST, and Morest isolates 21 more than ARAT-RL despite a lower *SEY*. Hitting server errors and finding different ones are not the same ability. The *FRE* advantage thus rests on *LBY*, which Subsubsection 4.3.2 isolates independently. The one dimension on which FDRRESTTEST is genuinely worse is wall-clock throughput, apportioned in Subsubsection 4.3.6.

■ **Table 3** Ablation on **Features Service**, the most state-rich subject, under the 600-second budget, including the Pareto variant of Subsection 3.7. All rows are runs of this ablation campaign, so Full and AUTORESTTEST differ from the same cell of Table 5 by run-to-run variation. Values are host-specific and not comparable to Table 2, which averages 12 services.

Variant	<i>SEY</i>	<i>LBY</i>	<i>FRE</i>	<i>RVR</i> (%)
Full	50.94	75.52	1.2647	33.85
No-DynDep	44.63	58.42	1.0305	28.45
No-Repair	44.35	51.26	0.9560	24.44
No-Oracle	47.09	9.83	0.5692	31.36
No-Scheduler	41.71	47.94	0.8965	27.38
Pareto	48.50	72.42	1.2092	30.94
AUTORESTTEST	52.03	65.13	1.1716	29.59

4.3.2 RQ2: Component Contribution

We ablate **Full** (complete FDRRESTTEST) against **No-DynDep** (no dynamic dependency evolution), **No-Repair** (no repair loop), **No-Oracle** (crash-only validation), **No-Scheduler** (heuristic scheduling), and **Pareto** (non-dominated sorting in place of the weighted utility, Subsection 3.7), with **AutoRestTest** for reference.

The ablation is run on six subjects spanning three dependency profiles, so that the decomposition is not tied to a single host: *near-stateless* (YouTube-Mock, Language Tool), *medium-scale stateful* (Features Service, User Management), and *large-scale complex* (Ohsome API, Project Tracking System; scale, not write-statefulness, as Ohsome API is read-only and contributes no state findings), $N=10$ per cell. Table 3 reports the per-variant detail on *Features Service*, the most state-rich subject, where component contributions are most visible; the cross-subset picture follows.

On Features Service, Full reaches $FRE=1.2647$, 7.9% above AUTORESTTEST; removing any single component drops FRE below AUTORESTTEST. No-Oracle leaves a qualitatively distinct footprint: it concentrates almost all loss in LBY ($75.52 \rightarrow 9.83$, with SEY dropping only 3.9 and RVR only 2.5 pp). The other three removals degrade all four metrics in the same ordering (LBY hit hardest, SEY and RVR least) and differ chiefly in magnitude – FRE drops by 18.5% (No-DynDep), 24.4% (No-Repair), and 29.1% (No-Scheduler) – with No-Repair also causing the clearest RVR drop (−9.4 pp). The residual $LBY = 9.83$ under No-Oracle comes from a minimal schema check retained in the parser-layer runtime store.

Two observations follow. First, the oracle is not substitutable by the exploration-side components: removing it eliminates 87% of LBY while SEY and RVR barely move. Second, no single-component variant reaches AUTORESTTEST on this host, so the advantage cannot be attributed to any one mechanism alone. The three exploration-side removals share a degradation profile and differ mainly in magnitude, so the ablation ranks their necessity without showing their contributions to be mechanistically distinct. The Pareto variant substitutes a mechanism instead of removing one: non-dominated sorting reaches $FRE = 1.2092$ against 1.2647 (−4.4%; two-sided Mann-Whitney $p < .001$, $N=10$ per variant) while issuing 3.6% fewer requests in the same budget, so the scalarisation is the cheaper of the two at this decision frequency (Subsection 3.7).

The two secondary metrics confirm the mechanism attribution. On RVR , No-Repair causes the largest drop (−9.4 pp), consistent with repair recovering otherwise-invalid requests. On OpR – per variant in the replication package – removing the scheduler costs the most reachability on each of the four subjects with more than two operations, No-DynDep and

No-Repair less, while No-Oracle leaves it almost intact: the expected separation between exploration- and validation-side components, and independent confirmation that the No-Oracle footprint in *LBY* is not an exploration artefact.

Consistency across dependency profiles. Full is highest on all six subjects and the exploration-side losses scale with dependency density. Averaged per subset over the per-subject percentage drops (near-stateless / medium stateful / large complex), No-DynDep costs -2.7% / -15.4% / -22.9% of *FRE* and No-Scheduler -8.7% / -23.6% / -30.7% : both gradients are monotone in dependency density, as the mechanisms predict. Removing the oracle costs -48.5% / -51.5% / -48.1% , essentially profile-independent: the signature of a validation-side component, and the strongest single effect on every subject. No-Repair is the exception: its subset means happen to order (-2.1% / -13.0% / -14.9%), but the within-subset spread reaches 22.8 pp against at most 11.1 pp for any other variant (24.4% on Features Service against 1.6% on User Management, both medium), so we claim no per-subset gradient and report the single-host magnitude instead: repair contributes where inter-operation constraints make intermediate failures frequent, which tracks a service’s constraint density, not its profile.

AUTORESTTEST is closest on the near-stateless subset, where Full leads it by 5.6%, against 6.8% and 6.3% on the other two (these margins are relative to AUTORESTTEST and are computed on the subset-mean *FRE*, unlike the per-subject means above; the component-removal percentages above are relative to Full). Removing the oracle drops *FRE* below AUTORESTTEST on all six subjects; the exploration-side removals do so on four (No-DynDep), six (No-Scheduler) and three (No-Repair) respectively. The ablation therefore establishes the oracle layer as indispensable across all six ablation subjects and all three dependency profiles, while the exploration-side components are necessary but not individually sufficient to clear the strongest baseline on every profile.

4.3.3 RQ3: Logical Defect Detection

Table 4 reports deduplicated counts for every tool and non-crash category. Because the four non-crash oracles are a contribution of FDRRESTTEST rather than a feature of the baselines (Table 6), the per-category entries for every tool other than FDRRESTTEST are produced by applying the shared five-oracle harness post-hoc to its logs; they are therefore lower bounds on what each tool’s own reporting would surface. We draw category-level conclusions only from the FDRRESTTEST–AUTORESTTEST pair, the two systems that share a code base (Subsubsection 4.2.2) and differ exactly in the mechanisms under study.

Counts increase monotonically across the paradigms, but the per-tool sets are largely disjoint (201 of the 236 union triples come from one tool; the only substantial overlap is the 35 shared by the two systems with a common code base), so the ordering reflects how much of each tool’s traffic the shared harness can score, not containment between paradigms. Every column other than FDRRESTTEST’s is a lower bound, so the $+41\%$ – $+69\%$ steps beneath compare two lower bounds; only the FDRRESTTEST-vs-AUTORESTTEST step ($+26$ findings, $+37\%$ on a base of 70) contrasts two systems that share a code base and differ exactly in the mechanisms under study. We read the ordering as a trend across paradigms, not as evidence about any single step. FDRRESTTEST leads AUTORESTTEST in all four categories, with the largest gains in constraint findings ($+10$) and state findings ($+7$), consistent with the roles of dependency evolution and the constraint oracle.

■ **Table 4** Logical defect findings across 12 services as *absolute deduplicated counts*: the distinct-defect view, not the per-request signals underlying *LBY*, and not aggregable with them (Subsubsection 4.2.3). FDRRESTTEST’s counts come from its own oracle layer; the other five are recovered post-hoc with the shared harness and are lower bounds. Deduplication applies the (endpoint, oracle type, semantic fingerprint) triple uniformly; counts are the union over the $N=10$ repetitions.

Finding Category / Tool	RESTler	EvoMaster	Morest	ARAT-RL	AutoRestTest	FDRRestTest
Structural findings	3	5	7	9	16	19
State findings	4	6	8	12	19	26
Consistency findings	3	5	7	11	17	23
Constraint findings	3	6	9	12	18	28
Total deduplicated findings	13	22	31	44	70	96

Representative defects. One representative deduplicated defect per non-crash category, each cited by the `predicate` under which the artefact records it: *state*: `idempotent_put_changed_state` on `PUT /api/person` (Person Controller, MC-107), a repeated identical PUT leaving a different resource state; *constraint*: `type_coercion_accepted` on `GET /history` (FDIC, MC-062), constraint `type@offset`; *structural*: `response_type_mismatch` on `POST /annotation` (Genome Nexus, MC-087), constraint `type@most_severe_consequence`; *consistency*: `cross_endpoint_divergence` on `GET /elements/area/groupBy/key` (Ohsome API, MC-082). The identifiers are the `sample_id` column of the annotation matrix.

Manual confirmation study. This study estimates the precision of the *shared oracle layer* behind every entry of Table 4: native for FDRRESTTEST, post-hoc for the five baselines. The frame is the union of all six tools’ deduplicated findings under the same triple: 236 distinct findings (51/59/56/70 structural/state/consistency/constraint), smaller than the column sums because the sets overlap – FDRRESTTEST and AUTORESTTEST, sharing a code base, in 35 triples. We draw 30 per category ($N = 120$, a 51% sampling fraction), allocated per service in proportion to its share of that category’s frame, so read-only subjects contribute nothing to the state stratum. Two annotators labelled each finding independently against source code or documented behaviour, resolving disagreements by discussion. The confirmation rate is $96/120 = 80\%$ (95% Wilson confidence interval, CI, [72.0%, 86.2%]), with inter-annotator agreement Cohen’s $\kappa = 0.78$ (95% CI [0.63, 0.93], substantial), computed on the pre-discussion labels. Table 4 therefore reads as an inventory of *oracle findings* of which a substantial majority are genuine defects, for every tool; the interval width still precludes per-category claims. The matrix and guidelines are in the replication package.

4.3.4 RQ4: Budget Sensitivity

We vary the time budget on *Features Service*, the most state-rich subject, $N=10$ per cell. Each tier is a separate execution inside this sweep. At 60/180/300/600s: *SEY* = 44.80, 48.30, 49.70, 51.44; *LBY* = 59.62, 66.43, 70.53, 75.32; *FRE* = 1.0442, 1.1474, 1.2022, 1.2676. At 600s it reaches *FRE* = 1.2676, within run-to-run variation of the 1.2586 measured for

■ **Table 5** Per-service comparison under the 600-second budget; each cell reads *AutoRest-Test*/*FDRRestTest*. Two effects drive *FDRRestTest*'s lower request count – budget-aware scheduling (algorithmic) and per-request diagnosis/repair/oracle overhead (implementation) – apportioned in Section 4.3.6. The higher *OpR* shows the reduced volume still reaches more operations, and Subsubsection 4.3.6 shows the per-request advantage survives equal volume.

Service	Requests	<i>OpR</i> (%)	<i>SEY</i>	<i>LBY</i>	<i>FRE</i>
Features Service	52 374 / 6 315	100.0 / 100.0	52.12 / 51.01	65.30 / 74.85	1.1742 / 1.2586
Language Tool	48 513 / 4 102	100.0 / 100.0	33.57 / 33.87	36.27 / 41.62	0.6984 / 0.7549
Rest Countries	54 261 / 6 138	90.9 / 100.0	46.21 / 45.72	51.33 / 56.56	0.9754 / 1.0228
Genome Nexus	28 518 / 4 232	82.6 / 95.7	45.71 / 43.65	48.82 / 54.82	0.9453 / 0.9847
Person Controller	36 414 / 4 831	83.3 / 100.0	40.09 / 45.21	45.20 / 50.67	0.8529 / 0.9589
User Management	32 137 / 4 572	81.8 / 95.5	47.41 / 47.59	52.14 / 57.59	0.9955 / 1.0518
Market	25 668 / 3 681	76.9 / 92.3	44.46 / 44.56	42.30 / 49.28	0.8676 / 0.9384
Project Tracking System	18 504 / 2 914	68.7 / 82.1	43.41 / 40.87	41.91 / 48.49	0.8532 / 0.8936
YouTube-Mock	45 076 / 5 284	100.0 / 100.0	49.34 / 48.46	55.12 / 59.29	1.0445 / 1.0776
FDIC	22 483 / 3 195	66.7 / 83.3	45.58 / 47.29	47.43 / 50.72	0.9301 / 0.9801
Spotify	19 674 / 2 836	75.0 / 91.7	51.06 / 51.15	41.85 / 47.09	0.9292 / 0.9823
Ohsome API	15 435 / 2 457	63.9 / 78.7	44.14 / 43.77	44.53 / 51.23	0.8867 / 0.9500

the same cell under the main protocol (Table 5). *FRE* increases monotonically across the four budgets – by +21.4% from 60s to 600s – with all three adjacent steps giving Cliff's $\delta = 1.00$ on *LBY*. *OpR* rises 72.2/88.9/94.4/100.0% over the same budgets, so the extra time reaches new operations instead of re-walking reached ones. The replication package additionally sweeps Language Tool, Ohsome API and Project Tracking System; there the final 300→600s step gives $\delta = 0.54, 0.84$ and 0.84 , so strict per-step separation is a property of this host rather than a general one.

4.3.5 RQ5: Per-Service Behaviour and Repair Impact

Table 5 compares the two tools per service: *FDRRESTTEST* wins on *FRE* in all 12, so the gain in Table 2 is not outlier-driven; the largest request-volume reduction is on Language Tool (11.8×). The structural pattern is in *OpR*: gains are large (+9.1–+16.7 pp) on every service not already fully reachable, and do not order by service scale. Relative *FRE* gains do *not* order by state richness: near-stateless Language Tool (+8.1%) is above the 12-service average of +6.3%, YouTube-Mock (+3.2%) below. We therefore support the design hypothesis on reachability, where the mechanism-to-metric link is direct, and claim no monotone relationship between state richness and per-request efficiency. The per-service rows are full configurations and do not isolate the repair loop; that separation comes from Subsubsection 4.3.2: on the state-rich host, removing repair costs 24.4% *FRE* and 9.4 pp *RVR*, the largest validity drop of any single removal. The per-subset decomposition of Subsubsection 4.3.2 finds no ordering with dependency density, so it does not support repair contributing proportionally more on dependency-dense services.

4.3.6 RQ5 complement: Robustness to the budget denomination

A fixed-time protocol favours tools differently according to their per-request overhead, so we re-ran the full cross-tool comparison under a *fixed-request* budget: every tool is capped at the same number of issued requests per service and runs to completion, with no time limit. The three caps – 1000, 3000, 6000 requests per service – span the volumes *FDRRESTTEST* issues within 600s (Table 5); 12 services × 6 tools × $N=10$ per cap.

The ranking is unchanged and the margin widens as the cap tightens: at 3000 requests FDRRESTTest reaches $FRE = 0.9670$ against 0.8307 (+16.4%), winning on all 12 services (exact two-sided $p = 2/2^{12}$, $d_z = 5.74$); the margin is +24.0% at 1000 and +9.6% at 6000, with the four weaker baselines trailing by +66% or more at every cap on their post-hoc recovered FRE , again an upper estimate. The shrinking advantage is expected: the baselines converge towards their own saturation while FDRRESTTest is already near its ceiling at 3000 requests.

Two observations matter for interpreting the fixed-time results. First, the SEY parity of Subsubsection 4.3.1 survives equalised volume: at 3000 requests FDRRESTTest's SEY is 35.84 against AUTORESTTest's 33.73, converging to 35.67 vs. 35.40 at 6000. Per-request crash detection is thus a property of the shared oracle, not of either search strategy. Second, the closed loop costs wall-clock time: reaching the same request count takes FDRRESTTest about $7.3\times$ longer. Under a request-denominated budget FDRRESTTest is the more effective tool per request; under a wall-clock budget that advantage must be earned back against a much lower request rate. Neither protocol subsumes the other.

Apportioning the throughput gap. Two mechanisms are confounded in the $7.7\times$ request-volume gap: the scheduler *chooses* fewer, better sequences (algorithmic), and diagnosis, repair and oracle evaluation sit inside the request loop (implementation). A same-throughput control separates them – an *async-feedback* variant moving those three stages out of the loop, 12 subjects, $N=10$. Throughput rises $3.00\times$ ($4\,213 \rightarrow 12\,646$ requests per 600s), leaving $2.56\times$ attributable to the scheduler's restraint; the two shares are comparable, so neither alone explains the gap. The extra volume costs per-request efficiency: FRE falls to 0.8385 (−15.1%), entirely through LBY ($53.52 \rightarrow 38.64$) while SEY barely moves ($45.26 \rightarrow 45.21$), since the added requests are the ones the scheduler had de-prioritised and delayed feedback degrades its choices. Under a quota-bound budget the synchronous configuration is the right operating point; both run sets are in the artefact.

4.4 Discussion

The aggregate FRE gain over AUTORESTTest is +6.3%, an upper estimate (Subsubsection 4.3.1) and a moderate margin whose interpretation rests on three orthogonal properties: *consistency* (12/12 subjects, $d_z = 3.73$ on LBY), *budget efficiency* ($6.3\text{--}11.8\times$ fewer requests), and *statistical reliability* ($p_{\text{adj}} < .005$). Consistency is the strongest of the three: a uniform sign across heterogeneous subjects is harder to produce by chance than a given aggregate margin.

Practical implications. The three views of the LBY uplift – +12.2% per request, +37% deduplicated count, 26 more distinct findings – translate for a continuous-integration pipeline into a larger share of the API surface within the window (OpR 93.3% vs. 82.5%). Each report carries the operation sequence, request, response and predicate, mapping to developer action (*state*→persistence/cache, *constraint*→input validation, *consistency*→cross-endpoint sync, *structural*→schema contract).

5 Threats to Validity

Internal validity / conclusion. All nine hyperparameters (Subsubsection 4.2.4) are fixed after pilot tuning on a held-out *Scout API* and not re-tuned per subject. Perturbing each by $\pm 20\%$ and re-running all 12 subjects ($N=5$ per configuration): per-service mean FRE

varies by at most 7.9% on any subject, the largest absolute shift being 0.082 at $\tau_{\text{lex}} - 20\%$ on Features Service; two of the eighteen perturbations improve *FRE* slightly, so the fixed values are a good operating point, not a sharp optimum. The four scheduler weights $\lambda_1 - \lambda_4$, the parameters most directly implicated by the scalarisation of Subsection 3.7, stay within that same band. The sweep is one-factor-at-a-time and does not probe parameter interactions, a threat to conclusion validity; the sweep run-level records are in the artefact.

Construct validity / external validity. LBY is oracle-dependent, mitigated by the shared harness. Table 4 counts deduplicated oracle findings, not confirmed bugs; the study of Subsubsection 4.3.3 gave 80% true positives (95% Wilson CI [72.0%, 86.2%], $\kappa = 0.78$), so oracle precision is bounded in aggregate but not per category, and the two annotators shared guidelines, which may correlate their errors. The benchmark excludes enterprise-level APIs, and OpenAPI dialect may affect transferability. FDIC, Spotify and Ohsome API enforce quotas and our oracles do not separate a business-logic fault from an infrastructure-level rejection under load, so some error signals may come from throttling; 429 responses are 0.7–0.9% of requests there and differ by at most 0.02 pp between the two tools, so the inflation is common-mode. RQ2 covers six of the 12 subjects and its single-removal design limits it to necessity claims; RQ4 and the Pareto comparison are single-host, so budget-monotonicity is supporting evidence on one state-rich subject.

Implementation efficiency. FDRRestTest’s request rate is markedly lower than some baselines (Table 5). *FRE* is per-request, but a lower rate under a fixed time budget might undercount defects requiring many *different* requests. The fixed-request replication of Subsubsection 4.3.6 is the direct test: at the same request count the ranking holds at all three caps and the margin over AUTORESTTEST widens to +16.4% at 3000, so the advantage is not an artefact of the fixed-time protocol. The residual threat runs the other way: reaching a given request count costs FDRRESTTEST about $7.3\times$ more wall-clock time, so where the binding constraint is time rather than quota, that overhead limits achievable yield. The same-throughput control of Section 4.3.6 apportions the two shares ($3.00\times$ implementation, $2.56\times$ algorithmic) but does not remove the overhead itself. Finally, the exclusion of LogiAgent is a limitation; we treat it as an orthogonal direction requiring special comparison protocols (matched token budgets, per-defect costs).

6 Related Work

Building on REST and specification foundations [29, 31, 27, 32], prior black-box REST testing spans stateful fuzzing, search-based, graph-guided, reinforcement-learning, and LLM-assisted paradigms [4, 24, 2, 17, 20, 37, 18], with adjacent tools [15, 26, 33, 23, 36, 7, 12, 6, 1, 25, 19, 16, 8, 30, 14, 9] and surveys [11, 38, 21, 34, 22, 28].

Table 6 makes the differentiation concrete: no row is novel on its own – runtime dependency evolution exists in Morest, a five-family logical oracle in LogiAgent – but no prior system carries all six dimensions, four of which are evaluated here; the ablation of Subsubsection 4.3.2 drops *FRE* below the strongest baseline on every removal.

Applicability boundaries. Effectiveness is predicated on a reasonably complete OpenAPI specification and stateful, multi-step interactions; the benefit is narrowest in *OpR* – where the mechanism-to-metric link is most direct – on subjects already reachable without dependency reasoning (YouTube-Mock, Language Tool). Relative *FRE* gains, by contrast, do not order by

■ **Table 6** Differentiation from the closest prior systems. ✓ present; ~ partial or auxiliary; × absent.

Design dimension	Morest	AutoRestTest	LogiAgent	FDRRestTest
Runtime dependency-graph evolution	✓	~	×	✓
Failure repair: taxonomy + memory	×	×	×	✓
Logical oracle, five families	×	×	✓	✓
Non-crash state/consistency oracles	×	×	✓	✓
Budget-aware scheduling	~	~	×	✓
Fingerprint dedup + statistical test	×	~	~	✓

state richness (Subsubsection 4.3.5). The implementation assumes no dynamic-interaction authentication (e.g., OAuth2.0 grants) and no budget-exhausting rate limit; our FDIC, Spotify and Ohsome API runs were quota-constrained, and OpenAPI 3.1 features remain unexplored.

7 Conclusion and Future Work

We presented FDRRestTest, integrating dependency-graph evolution, failure repair, a multi-dimensional logical oracle and budget-aware scheduling in one closed loop; on 12 services against five baselines it delivers a consistent *FRE* improvement (up to +6.3% over the strongest baseline on all 12 subjects, $p_{\text{adj}} < .005$, $d_z = 3.73$ on *LBV*). A stratified manual confirmation study ($N = 120$, 80% confirmed, $\kappa = 0.78$) supports the oracle-finding reading of the headline *LBV* gain, and an ablation over six subjects in three dependency profiles shows the oracle layer costs the most on removal. Future work: a matched-token-budget LogiAgent comparison, multi-component removals to quantify interaction effects, and validation on enterprise-scale APIs.

Data Availability Statement

The replication package is archived at <https://doi.org/10.5281/zenodo.21964240>: the implementation, the shared-oracle and deduplication harness, the statistical scripts, run-level records for every reported cell with the repair tallies of Subsection 3.5, and – for the 720 fixed-time runs – per-request operation and status-code traces (200 sampled each), per-run histograms, the triggering request and response of every finding, and the $N = 120$ annotation matrix with guidelines and sampling record. Attribution is in NOTICE.

References

- 1 J. C. Alonso, A. Martin-Lopez, S. Segura, J. Garcia, and A. Ruiz-Cortes. ARTE: Automated generation of realistic test inputs for web APIs. *IEEE Transactions on Software Engineering*, 49(1):348–363, 2023. doi:10.1109/TSE.2022.3150618.
- 2 Andrea Arcuri. RESTful API automated test case generation with EvoMaster. *ACM Transactions on Software Engineering and Methodology*, 28(1):3:1–3:37, 2019. doi:10.1145/3293455.
- 3 Andrea Arcuri and Lionel Briand. A hitchhiker’s guide to statistical tests for assessing randomized algorithms in software engineering. *Software Testing, Verification and Reliability*, 24(3):219–250, 2014. doi:10.1002/stvr.1486.
- 4 Vaggelis Atlidakis, Patrice Godefroid, and Marina Polishchuk. RESTler: Stateful REST API fuzzing. In *Proceedings of the 41st International Conference on Software Engineering*, pages 748–758, 2019. doi:10.1109/ICSE.2019.00083.
- 5 Norman Cliff. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin*, 114(3):494–509, 1993. doi:10.1037/0033-2909.114.3.494.

- 6 Davide Corradini, Zeno Montolli, Michele Pasqua, and Mariano Ceccato. 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*, pages 1383–1394, 2024. doi:10.1145/3691620.3695511.
- 7 Davide Corradini, Amedeo Zampieri, Michele Pasqua, Emanuele Viglianisi, Michael Dallago, and Mariano Ceccato. Automated black-box testing of nominal and error scenarios in RESTful APIs. *Software Testing, Verification and Reliability*, 32(5):e1808, 2022. doi:10.1002/stvr.1808.
- 8 Alix Decrop, Gilles Perrouin, Mike Papadakis, Xavier Devroey, and Pierre-Yves Schobbens. You can REST now: Automated specification inference and black-box testing of RESTful APIs with large language models. *arXiv preprint arXiv:2402.05102*, 2024. doi:10.48550/arXiv.2402.05102.
- 9 Sida Deng, Rubing Huang, Man Zhang, Chenhui Cui, Dave Towey, and Rongcun Wang. LRAS-Gen: LLM-based RESTful API specification generation. *arXiv preprint arXiv:2504.16833*, 2025. Manuscript submitted to ACM. doi:10.48550/arXiv.2504.16833.
- 10 Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Ph.D. dissertation, University of California, Irvine, 2000. URL: https://roy.gbiv.com/pubs/dissertation/fielding_dissertation.pdf.
- 11 Amid Golmohammadi, Man Zhang, and Andrea Arcuri. Testing RESTful APIs: A survey. *ACM Transactions on Software Engineering and Methodology*, 33(1):27:1–27:41, 2023. doi:10.1145/3617175.
- 12 Zac Hatfield-Dodds and Dmitry Dygalo. Deriving semantics-aware fuzzers from web API schemas. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, pages 345–346, 2022. doi:10.1145/3510454.3528637.
- 13 Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979. URL: <https://www.jstor.org/stable/4615733>.
- 14 Minh-Hieu Huynh, Quoc-Tri Le, Tien N. Nguyen, and Vu Nguyen. Using LLM for mining and testing constraints in API testing. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pages 2486–2487, 2024. doi:10.1145/3691620.3695341.
- 15 Stefan Karlsson, Adnan Čaušević, and Daniel Sundmark. QuickREST: Property-based test generation of OpenAPI-described RESTful APIs. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification*, pages 131–141, 2020. doi:10.1109/ICST46399.2020.00023.
- 16 Myeongsoo Kim, Davide Corradini, Saurabh Sinha, Alessandro Orso, Michele Pasqua, Rachel Tzoref-Brill, and Mariano Ceccato. Enhancing REST API testing with NLP techniques. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 1232–1243, 2023. doi:10.1145/3597926.3598131.
- 17 Myeongsoo Kim, Saurabh Sinha, and Alessandro Orso. Adaptive REST API testing with reinforcement learning. In *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering*, pages 446–458, 2023. doi:10.1109/ASE56229.2023.00218.
- 18 Myeongsoo Kim, Saurabh Sinha, and Alessandro Orso. LlamaRestTest: Effective REST API testing with small language models. *Proceedings of the ACM on Software Engineering*, 2(FSE):FSE022:1–FSE022:24, 2025. doi:10.1145/3715737.
- 19 Myeongsoo Kim, Tyler Stennett, Dhruv Shah, Saurabh Sinha, and Alessandro Orso. Leveraging large language models to improve REST API testing. In *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, pages 37–41, 2024. doi:10.1145/3639476.3639769.
- 20 Myeongsoo Kim, Tyler Stennett, Saurabh Sinha, and Alessandro Orso. A multi-agent approach for REST API testing with semantic graphs and LLM-driven inputs. In *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering, ICSE '25*, pages 1409–1421. IEEE Press, 2025. doi:10.1109/ICSE55347.2025.00179.

- 21 Myeongsoo Kim, Qi Xin, Saurabh Sinha, and Alessandro Orso. Automated test generation for REST APIs: No time to REST yet. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 289–301, 2022. doi:10.1145/3533767.3534401.
- 22 Caroline Lemieux, Jeevana Priya Inala, Shuvendu K. Lahiri, and Siddhartha Sen. CODAMOSA: Escaping coverage plateaus in test generation with pre-trained large language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering*, pages 919–931, 2023. doi:10.1109/ICSE48619.2023.00085.
- 23 Jiaxian Lin, Tianyu Li, Yang Chen, Guangsheng Wei, Jiadong Lin, Sen Zhang, and Hui Xu. foREST: A tree-based black-box fuzzing approach for RESTful APIs. In *2023 IEEE 34th International Symposium on Software Reliability Engineering*, pages 695–705, 2023. doi:10.1109/ISSRE59848.2023.00023.
- 24 Yi Liu, Yuekang Li, Gelei Deng, Yang Liu, Ruiyuan Wan, Runchao Wu, Dandan Ji, Shiheng Xu, and Minli Bao. Morest: Model-based RESTful API testing with execution feedback. In *Proceedings of the 44th International Conference on Software Engineering*, pages 1406–1417, 2022. doi:10.1145/3510003.3510133.
- 25 Alberto Martin-Lopez, Sergio Segura, and Antonio Ruiz-Cortés. A catalogue of inter-parameter dependencies in RESTful web APIs. In *Service-Oriented Computing*, pages 399–414, 2019. doi:10.1007/978-3-030-33702-5_31.
- 26 Alberto Martin-Lopez, Sergio Segura, and Antonio Ruiz-Cortés. RESTest: Automated black-box testing of RESTful web APIs. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 682–685, 2021. doi:10.1145/3460319.3469082.
- 27 OpenAPI. OpenAPI standard. <https://www.openapis.org>, 2023.
- 28 Rangeet Pan, Myeongsoo Kim, Rahul Krishna, Raju Pavuluri, and Saurabh Sinha. ASTER: Natural and multi-language unit test generation with LLMs. In *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '25, pages 413–424. IEEE Press, 2025. doi:10.1109/ICSE-SEIP66354.2025.00042.
- 29 Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. RESTful web services vs. “big” web services: Making the right architectural decision. In *Proceedings of the 17th International Conference on World Wide Web*, pages 805–814, 2008. doi:10.1145/1367497.1367606.
- 30 André Pereira, Bruno Lima, and João Pascoal Faria. ApiTestGenie: Automated API test generation through generative AI. *arXiv preprint arXiv:2409.03838*, 2024. doi:10.48550/arXiv.2409.03838.
- 31 Leonard Richardson, Mike Amundsen, and Sam Ruby. *RESTful Web APIs: Services for a Changing World*. O'Reilly Media, 2013.
- 32 SmartBear Software. Swagger specification. <https://swagger.io/specification/v2/>, 2023.
- 33 Emanuele Viglianisi, Michael Dallago, and Mariano Ceccato. RestTestGen: Automated black-box testing of RESTful APIs. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification*, pages 142–152, 2020. doi:10.1109/ICST46399.2020.00024.
- 34 Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. Software testing with large language models: Survey, landscape, and vision. *IEEE Transactions on Software Engineering*, 50(4):911–936, 2024. doi:10.1109/TSE.2024.3368208.
- 35 Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945. doi:10.2307/3001968.
- 36 Huayao Wu, Lixin Xu, Xintao Niu, and Changhai Nie. Combinatorial testing of RESTful APIs. In *Proceedings of the 44th International Conference on Software Engineering*, pages 426–437, 2022. doi:10.1145/3510003.3510151.
- 37 Ke Zhang, Chenxi Zhang, Chong Wang, Chi Zhang, YaChen Wu, Zhenchang Xing, Yang Liu, Qingshan Li, and Xin Peng. LogiAgent: Automated logical testing for REST systems with LLM-based multi-agents. *arXiv preprint arXiv:2503.15079*, 2025. doi:10.48550/arXiv.2503.15079.
- 38 Man Zhang and Andrea Arcuri. Open problems in fuzzing RESTful APIs: A comparison of tools. *ACM Transactions on Software Engineering and Methodology*, 32(6):144:1–144:45, 2023. doi:10.1145/3597205.