10.48047/jocaaa.2024.33.08.253

Cognitive AI-Driven Decision Support for Sustainable and Resource-Efficient Software Testing in Small and Medium Enterprises

Meghana R,
Research Scholar (1NT22PCS03, Visvesvaraya Technological University,Belagavi)
Intel Corporation,
meghanar95@gmail.com

Dr.Piyush Kumar Pareek
Research Supervisor ,Nitte Meenakshi Institute of Technology , Visvesvaraya Technological University ,Belagavi-590018,India
Piyush.kumar@nmit.ac.in

Abstract

Software testing is indispensable for quality assurance, yet in small and medium enterprises (SMEs) it frequently becomes the most time- and compute-intensive phase of delivery, inflating costs, energy use, and time-to-market. This paper proposes CORA-TEST (Cognitive Resource-Aware Testing), an AI-driven decision support framework that helps SMEs conduct sustainable and resource-efficient software testing without compromising fault detection. The framework senses telemetry from CI/CD pipelines, code changes, and flakiness logs; reasons with cognitive AI models to assess risk, yield, and environmental impact; recommends prioritized execution plans and remediation actions; and governs adoption through policy-as-code and experiment-driven validation. We synthesize recent advances in Green-AI, test case selection/prioritization, just-in-time defect prediction, flakiness mitigation, and energy-aware CI, and adapt them to the constraints and opportunities of SME settings. The proposed approach is incremental, explainable, and auditable, enabling teams of 10–75 engineers to realize measurable reductions in test time, cloud spend, and estimated carbon while maintaining or improving reliability.

Keywords: sustainable software testing; small and medium enterprises; cognitive AI; test case prioritization; flakiness mitigation; energy-aware CI/CD; policy-as-code; decision support

Introduction

SMEs increasingly ship software through rapid CI/CD loops, yet many still run large regression suites indiscriminately across multiple environments and configurations. The result is expensive, energy-hungry pipelines, slow feedback for developers, and avoidable contention for limited compute. Traditional remedies—such as manually pruning tests or adding hardware—do not scale in lean organizations and risk hidden quality trade-offs.

Cognitive, data-driven decision support offers a pragmatic alternative. By learning from change metadata, historical failures, coverage maps, execution costs, and environmental signals, AI can rank tests by expected failure yield per unit cost, detect and quarantine flaky cases, recommend right-sized runners and images, and shift non-urgent executions to lower-carbon windows or regions. Crucially, these models need not be heavyweight; compact tree ensembles

10.48047/jocaaa.2024.33.08.253

or small neural networks trained on local logs often suffice when wrapped in guardrails and explanations.

Sustainability must be explicit. Beyond cutting minutes in CI, SMEs are increasingly asked to demonstrate environmental stewardship. Exposing estimated kWh and $kgCO_2e$ alongside time and cost reframes testing as a multi-objective problem rather than a single-metric race. Embedding that perspective at the pipeline level normalizes sustainability as a quality attribute.

We contribute CORA-TEST, a framework that inserts cognitive AI into everyday testing decisions while preserving developer trust and velocity. We lay out a modular architecture, adoption playbook, and evaluation methods suitable for SMEs, and we discuss expected impacts, risks, and governance patterns that make improvements durable.

Literature Survey

Green-AI and reporting discipline. Recent position and survey works advocate measuring and reporting energy and carbon for AI workloads together with accuracy and latency. This motivates adding energy/CO_2 fields to test and build "cards," prioritizing efficient architectures, and using carbon-aware scheduling for non-urgent jobs—practices that map directly to testing pipelines in SMEs.

Machine-learning test selection/prioritization. Systematic reviews from 2021–2022 show that ML-based test case selection and prioritization (TSP/TCP) often outperform heuristic baselines under CI constraints. High-signal features include test history, failure yield, coverage overlap with modified files, dependency reachability, and change metrics (diff size, churn, ownership).

Metrics and evaluation for TCP. Complementary surveys emphasize APFD/APFDc and history-aware strategies, arguing for rolling-window retraining to handle test aging. They recommend combining static risk cues (e.g., touched components) with dynamic runtime signals (recent flake, execution time) to stabilize gains and avoid brittle prioritizers.

Just-in-time (JIT) defect prediction. Modern surveys catalog change-level features—entropy of edits, developer history, hot-spot files—and learners suitable for CI integration. Practical guidance is to route high-risk commits to fuller suites or additional configurations, while low-risk paths receive prioritized subsets, with explainable rationales to preserve reviewer trust.

Deep learning for defect prediction. Evidence indicates compact deep nets can outperform classic learners on multiple corpora, though generalization and explainability remain concerns. Hybrid pipelines—deep scorers paired with interpretable meta-rules and conservative gating—improve adoption in small teams.

Effort estimation in agile teams. Reviews highlight that simple, explainable models calibrated on local data reduce planning noise, stabilize sprint cadence, and indirectly lower compute waste by avoiding thrash in CI/testing schedules.

Quality for AI-enabled systems. Systematic mappings of non-functional attributes—robustness, fairness, security, reliability—warn that speed and energy optimizations must not

10.48047/jocaaa.2024.33.08.253

degrade product qualities. Embedding bias/drift/robustness checks as CI gates keeps product integrity aligned with efficiency drives.

Energy-aware CI/CD. Research prototypes show that modest instrumentation (runtime, CPU/GPU utilization, cache hit rates, container size) can expose hotspots and support energy-aware recommendations. Treating energy regressions like any other quality regression normalizes sustainability in day-to-day engineering.

Unified regression test selection (code + configuration). Newer approaches elevate configuration changes (YAML, Helm, Terraform) to first-class status, shrinking the candidate set safely before any ML ranking. This is pivotal in modern monorepos and infra-heavy stacks common in SMEs.

Community consolidation around AI engineering. The emergence of venues, datasets, and governance patterns at the SE↔AI boundary provides reusable templates—model cards, dataset versioning, drift monitors, policy-as-code—that SMEs can adopt rather than build from scratch, accelerating responsible AI-in-the-pipeline practices.

Research Methods: The CORA-TEST Framework

Scope and principles

Target context: SMEs with 10–75 engineers, multi-service architectures, and standard CI/CD. Principles: reuse existing rails; keep AI thin and explainable; favor incremental rollout; encode guardrails as code; ensure auditability and human oversight.

Architecture

Sensing (observe). Collect per-run telemetry: test durations, pass/fail and flake signals, coverage deltas, queue times, container size/layer reuse, cache hits, and resource usage (CPU/memory/IO). Enrich with change metadata (files, churn, ownership), component risk tags, and optional energy/CO_2 estimates via consistent proxies.

Reasoning (learn).

- *Change-risk model:* scores commits using change and history features to determine suite breadth.
- *Test-yield model:* estimates each test's probability of revealing a failure given the incoming change and cost (time/compute), enabling yield per unit cost ranking.
- *Flakiness model:* classifies tests as stable/unstable using intermittency, environment sensitivity, and dependency signals.
- *Env/right-sizing model:* recommends runner class, parallelism, and container profile appropriate to the plan.

All models are compact (tree ensembles or small neural nets), retrained on rolling windows to handle drift, and expose feature importances or exemplar-based explanations.

10.48047/jocaaa.2024.33.08.253

Recommendation (decide). Produce an execution plan: prioritized subset vs. full run; configuration matrix; runner type/region; schedule (immediate vs. off-peak low-carbon); and remediation actions (quarantine flake, deflake task, cache fix, container slimming). Each plan includes expected impact on time, cost, and estimated kWh/kgCO_2e, along with a confidence label and explicit "won't-do" constraints (never skip security/compliance suites).

Governance (assure).
Policy-as-code enforces coverage floors, flake budgets, maximum container size, latency/SLO bounds, and red-line suites. The framework auto-generates test cards (inputs, plan, outcomes, impacts) for auditing. Rollouts follow canary patterns; exceptions exist for hotfixes and critical incidents.

Operating model (phased adoption)

Phase 1 — Observe. Enable telemetry and publish baseline scorecards for 2–3 weeks; identify top offenders (longest tests, flakiest suites, heaviest images, queue hotspots).
Phase 2 — Recommend. Run cognitive models in advisory mode only; surface plans in PR comments or dashboards; fix obvious hygiene issues (deflake queue, caching, container layers).
Phase 3 — Gate. Turn on gentle policy gates (warn→block) for egregious regressions; allow high-risk changes to escalate automatically to fuller suites/configurations.

Evaluation and validation

Process experiments. Randomize merges into *baseline* vs. *CORA-TEST plan* and measure CI time, compute costs, pass/fail yield, escaped defects, and estimated energy/CO_2e.
Reliability metrics. Track coverage, APFD-like early fault detection, and flake trend lines.
Fairness/safety. Audit whether certain teams or components see excessive pruning; lock down mandatory suites; monitor blast radius on change failure rate and MTTR.
Learning loop. Retrain models on fresh data, log accepted/rejected recommendations to reduce noise, and publish monthly "green wins" (minutes saved, kWh avoided).

Discussion

Expected benefits. Predictive test selection combined with flake control commonly yields 10–30% reductions in CI time, with proportional decreases in compute spend and estimated emissions. Right-sizing runners and slimming containers further reduce wall-clock and cold-start latencies. Routing high-risk changes to fuller suites improves detection stability, decreasing escaped defects and rework.

Trade-offs and safeguards. Over-pruning risks missed regressions; aggressive image minimization can hinder local development; carbon-aware scheduling may slightly delay non-urgent feedback. Multi-objective recommendations, conservative defaults, and policy guardrails (coverage floors, never-skip suites, latency/SLO caps) mitigate these risks. Confidence labels and explanations help reviewers exercise judgment on when to accept or override.

10.48047/jocaaa.2024.33.08.253

Fit for SMEs. The framework's thin AI and reuse of existing CI, artifact registries, and trackers minimize operational burden. Most benefits accrue where SMEs already feel pain—tests, containers, and queues—so early wins are visible within a sprint or two, which is essential for sustaining buy-in.

Limitations and future directions. Energy estimates based on proxies are noisy; use them comparatively and calibrate over time. Cross-repo generalization of yield models can be limited; pretraining on multi-repo corpora followed by local fine-tuning helps. Future enhancements include causal A/B attribution, integration with vendor carbon-intensity APIs for finer scheduling, and extending governance to dataset/model provenance for ML-heavy products.

Conclusion

CORA-TEST demonstrates how cognitive AI can make software testing in SMEs simultaneously leaner and greener. By sensing real pipeline signals, reasoning about change risk and test yield, recommending resource-aware execution plans, and enforcing guardrails through policy-as-code, SMEs can reduce CI time and compute, cut estimated carbon, and improve reliability. The framework is practical, incremental, and auditable, turning sustainability from an aspiration into an operational capability embedded in everyday engineering.

References

1. Schwartz, R., Dodge, J., Smith, N. A., Etzioni, O., "Green AI," *Communications of the ACM*, 63(12), 2020.
2. Pan, R., Bagherzadeh, M., Ghaleb, T. A., Briand, L., "Test Case Selection and Prioritization Using Machine Learning: A Systematic Literature Review," *Empirical Software Engineering*, 2022.
3. Barbosa, G. A. de S., et al., "A Systematic Literature Review on Prioritizing Software Test Cases," *Information and Software Technology*, 2022.
4. Zhao, Y., Xia, X., Lo, D., Pan, J. Z., "A Systematic Survey of Just-in-Time Software Defect Prediction," *ACM Computing Surveys*, 2023.
5. Giray, G., "On the Use of Deep Learning in Software Defect Prediction: A Systematic Literature Review," *Information and Software Technology*, 2023.
6. Fernández-Diego, M., Méndez, E., González-Ladrón-de-Guevara, F., Abrahão, S., Insfran, E., "An Update on Effort Estimation in Agile Software Development: A Systematic Literature Review," *IEEE Access*, 2020.
7. Gezici, B., Tarhan, A. K., "Software Quality for AI-Based Systems: A Systematic Literature Review," *Empirical Software Engineering*, 2022.

10.48047/jocaaa.2024.33.08.253

8. Kruglov, A., et al., “Incorporating Energy-Efficiency Measurement into CI/CD Pipelines,” in *Proc. ACM/ICPS*, 2021.

9. Wang, S., et al., “Unified Regression Test Selection for Code and Configuration,” Technical Report/Preprint, 2022.

10. IEEE/ACM, “International Conference on AI Engineering—Software Engineering for AI (CAIN),” Community Proceedings, 2022–2023.