COLOMBO SCIENTIFIC PUBLISHING

INTERNATIONAL JOURNAL OF COMPUTER SCIENCE & INFORMATION SYSTEM
E-ISSN: 2536-7919
P-ISSN: 2536-7900

PAGE NO: 85-92

Adaptive Testing of Large Language Model–Enhanced Software Systems: A Comprehensive Framework for Requirements, Test-Case Generation, Prioritization, and Evaluation

Rahul M. Bennett
Department of Computer Science, University of Manchester, United Kingdom

ARTICLE INFO

Article history:
Submission: October 18, 2025
Accepted: November 17, 2025
Published: November 30, 2025
VOLUME: Vol.10 Issue 11 2025

Keywords:
software testing, large language models, test-case generation, requirements testing, test prioritization, configurable systems, focal methods

ABSTRACT

Background: The rise of large language models (LLMs) and their integration into software development toolchains has introduced new dimensions to software testing, from automated test-case generation to system-level validation, while simultaneously complicating requirements testing and configurable-system evaluation (Wang, 2024; dos Santos, 2020). Existing literature on software product lines, configurable systems, and regression test prioritization offers foundational methods that must be reinterpreted when LLMs participate as both test artifact producers and application components (Agh, 2024; Souto, 2017; Elbaum, 2002).

Objective: This article proposes a unified, publication-ready theoretical and methodological framework for testing LLM-enhanced software systems that spans requirements elicitation and testing, automated unit and integration test generation using LLMs, focal-method mapping, test-case selection and prioritization, and empirical evaluation strategies. The framework emphasizes balancing soundness and efficiency in configurable environments while employing machine learning–based prioritization and leveraging recent advances in LLM tool-use and web-agent architectures (Pan, 2022; Tufano, 2022; Schick et al., 2023).

Methods: We synthesize evidence from systematic literature reviews, empirical studies, and recent preprints to build a layered methodology: (1) requirements-level formalization and traceable test intent extraction; (2) LLM-driven test-case generation templates and focal-method mapping; (3) hybrid selection and prioritization using feature-aware ML models and historical regression data; (4) orchestration for end-to-end testing and evaluation in realistic web environments; and (5) continuous monitoring and adaptive re-prioritization. Each component is described with prescriptive guidelines and evaluative metrics. Literature-based rationale and thought experiments ground the proposed choices (dos Santos, 2020; Wang, 2024; Chandra, 2025).

Results: The conceptual framework yields measurable improvements along three axes in thought experiments and empirical analogues discussed herein: coverage of requirement-derived behaviors, fault-revealing power of generated test suites, and regression test efficiency under budget constraints. Drawing on methods from test-case prioritization research and focal-method mapping, we show how LLM-generated tests can be filtered and ranked to achieve higher early-fault detection compared to naïve generation, while addressing configurable-system explosion through sampling and soundness-efficiency trade-offs (Elbaum, 2000; He, 2024; Souto, 2017).

Conclusions: LLMs offer unprecedented capabilities for automating parts of the testing lifecycle, but their effective integration requires principled pipelines that combine requirements engineering, focal-method guidance, adaptive prioritization, and environment realism. The proposed framework provides a roadmap for researchers and practitioners to construct, evaluate, and iterate robust testing systems for contemporary software that integrates LLMs either as tooling or as functional components. Future work should empirically validate the framework across diverse domains, quantify human–LLM collaboration dynamics in testing, and extend the approach to continual learning settings.

INTRODUCTION

The landscape of software testing is undergoing a transformative evolution driven by two parallel forces: the increasing complexity of configurable software systems and the advent of large language models (LLMs) that participate actively in software creation, augmentation, and evaluation. Historically, software testing research has advanced methods for regression prioritization, test-case selection, and automated test generation to manage growing code bases and demanding release cadences (Elbaum, 2002; Elbaum, 2000). Simultaneously, research on requirements testing and software product line testing has articulated the challenges in ensuring consistent behavioral conformance across variants and evolving requirements (dos Santos, 2020; Agh, 2024). More recently, LLMs have emerged as both powerful tools for automating unit- and integration-level test generation and as components embedded within applications, necessitating new testing paradigms that account for non-deterministic behavior, hallucination, and context-dependent responses (Wang, 2024; Lops et al., 2024).

This article addresses a pressing research gap: while separate literatures have characterized how to test configurable systems, prioritize regression suites, and apply ML for test selection, there is limited consolidation of these methods into a coherent framework that explicitly integrates LLM-driven practices and the realities of modern web-agent and tool-using LLM architectures (Souto, 2017; Pan, 2022; Schick et al., 2023). The integration of LLMs into the testing pipeline creates both opportunities—such as automated test-case creation and natural-language based requirement-to-test traceability—and challenges—such as ensuring generated tests' soundness and managing the vast output of model-based generation (Wang, 2024; Tufano, 2022). Furthermore, the rise of web-focused agent frameworks and tool-using LLM approaches demands test environments that approximate real-world contexts where LLMs act and interact (Nakano et al., 2021; Gur et al., 2023).

Our contribution is an integrative, detailed framework for testing LLM-enhanced software systems. We construct a layered methodology that starts at requirements and traces through to prioritized execution and evaluation. In doing so, we incorporate established strategies from test-case prioritization and focal-method research, apply machine learning to selection and ranking decisions, and emphasize realistic environment orchestration drawing on recent work in web-agent and toolformer paradigms (Elbaum, 2002; He, 2024; Schick et al., 2023). The framework is intended for academic researchers designing empirical studies as well as practitioners seeking to scale testing processes in industry settings where LLMs are becoming commonplace (Chandra, 2025).

The remainder of the article first reviews the relevant literature and situates the proposed framework within it. We then describe the methodology in depth—covering requirements formalization, LLM-guided test generation, focal-method mapping, selection and prioritization models, and evaluation strategies. We present descriptive results and interpretations based on literature-derived evidence and thought experiments, followed by a detailed discussion of limitations, potential counter-arguments, and directions for future research.

METHODOLOGY

This section articulates a prescriptive and theoretically grounded methodology for implementing the proposed testing framework. The methodology is organized into five interlocking layers: Requirements Formalization and Trace Extraction; LLM-Guided Test Generation; Focal-Method Mapping and Test Attribution; Test Case Selection and Prioritization; and Orchestration, Execution, and Evaluation. Each layer is designed to feed information downstream, creating feedback loops for continuous improvement. The design draws on systematic reviews and empirical studies that document strengths and weaknesses of prior approaches (dos Santos, 2020; Pan, 2022; Tufano, 2022).

Requirements Formalization and Trace Extraction

Testing begins with requirements; any test-suite that claims to achieve coverage must be traceable back to requirement-level intents (dos Santos, 2020). We propose a structured, dual-representation approach: (1) a natural-language artifact that preserves stakeholder intent and context; and (2) a semi-formal specification that captures operational preconditions, invariants, and success criteria. The natural-language artifact remains valuable for human validation, while the semi-formal specification (e.g., contracts or pseudo-formal descriptions) supports automated test template generation.

To transform natural-language requirements into test intents suitable for LLM consumption, we advocate a layered extraction pipeline. First, requirements are parsed using hybrid NLP pipelines that combine rule-based extraction with model-based semantic role labeling, producing candidate intents, variable lists, and assertion templates. Second, domain ontologies and configuration models are consulted to enrich the intents with variant-aware constraints (Agh, 2024). Third, a human-in-the-loop validation step ensures that the generated test intents correctly reflect stakeholder requirements; this helps limit propagation of requirement ambiguity into the test generation phase (dos Santos, 2020).

This process also aims to resolve common problems in requirements testing: ambiguity, incompleteness, and conflicts. Ambiguity is handled through intent clarification dialogues initiated by the test pipeline, where an LLM synthesizes clarifying questions to the requirements author; incompleteness is detected by coverage heuristics that flag missing preconditions or assertions; conflicts are surfaced by a constraint-satisfaction analysis comparing overlapping requirements. Each detection mechanism references prior literature emphasizing the necessity of rigorous requirements-to-test traceability (dos Santos, 2020).

LLM-Guided Test Generation

Once test intents are established, LLMs can be used to generate concrete test cases. Recent work has shown LLMs' capability to produce unit tests, assertions, and test scaffolding when supplied with sufficient context (Wang, 2024; Lops et al., 2024). However, naive generation risks producing syntactically correct but semantically weak tests that fail to reveal faults; therefore, our method prescribes controlled prompting and multi-step generation.

The LLM-guided generation pipeline includes the following elements:

1. Contextual Prompting: Prompts include the target function or API signature, associated focal methods (where available), semi-formal requirement excerpts, and relevant configuration constraints. Prompt templates are standardized to minimize prompt drift and ensure reproducibility (Tufano, 2022).

2. Diverse Example Seeding: To encourage a breadth of test inputs, we seed the model with diverse example pairs of inputs and expected outputs where available, drawn from existing test suites, usage logs, and API contracts. Diversification reduces the likelihood of mode-collapse in generation.

3. Multi-variant Synthesis: For configurable systems, the generator produces test variants that cover orthogonal configuration axes. We leverage combinatorial sampling methods and constraint solvers to avoid infeasible combinations while ensuring coverage across relevant feature combinations (Agh, 2024; Souto, 2017).

4. Assertion Generation and Hardening: For each generated test, the pipeline requires explicit assertions rather than general acceptance checks. We use auxiliary model calls and static analysis to transform soft assertions (e.g., “returns similar structure”) into hardened checks (e.g., shape and value constraints, type assertions, invariants). This reduces flakiness and increases determinism.

5. Meta-evaluation and Filtering: Generated tests are run in sandboxed environments; flaky tests or those that depend on underspecified environmental state are flagged and either rewritten or discarded. A small ensemble of LLMs or test heuristics is used to cross-validate generated assertions, reducing reliance on a single model's idiosyncrasies (Wang, 2024).

Collectively, these measures address the well-documented tension between automation and test quality when using generative models (Lops et al., 2024).

Focal-Method Mapping and Test Attribution

Focal methods—methods that are the central focus of a unit test—play a pivotal role in generating high-quality unit tests and tracing tests to code behaviors (Tufano, 2022; He, 2024). Mapping generated tests to focal methods allows targeted assertion placement, improved fault localization, and clearer prioritization.

Our methodology uses a two-stage approach for focal-method mapping:

1. Static-Focused Identification: Using code parsing and call-graph analysis, candidate focal methods are identified for each requirement or test intent. This includes analyzing method signatures, documentation comments, and usage sites to understand intended functionality.

2. Dynamic Confirmation via Test Execution: Candidate mappings are confirmed by running lightweight dynamic instrumentation during test execution. If a generated test exercises the candidate focal method(s) significantly, the mapping is considered confirmed; otherwise, reclassification or prompt augmentation is performed.

Recent empirical studies emphasize that focal-method-aware tests often exhibit stronger fault localization and yield more meaningful assertions, especially in deep-learning-based assertion generation contexts (He, 2024). By explicitly incorporating focal-method mapping into the pipeline, LLM-generated tests become more actionable and amenable to downstream prioritization.

Test Case Selection and Prioritization

One of the largest practical constraints in real-world testing is limited execution budget. Prioritization seeks to order test execution to maximize early-fault detection, improve feedback cycles for developers, and conserve resources (Elbaum, 2002; Elbaum, 2000). Our framework integrates ML-driven prioritization models with feature-aware heuristics and historical data.

Key components include:

1. Feature Encoding for Tests: Tests are encoded along multiple axes—focal-methods exercised, requirement origin, configuration footprint, input diversity, assertion strength, and historical flaky/failure signals. These features create a representation that an ML model can ingest for ranking purposes (Pan, 2022).

2. Supervised Learning from Historical Runs: Where historical test execution data exists, supervised ranking models (e.g., gradient-boosted trees or neural ranking architectures) are trained to predict the likelihood a test will reveal faults early. Training targets can be binary (fault-detecting vs. not) or ordinal (time-to-failure). This mirrors practices in test-case selection and prioritization literature that have demonstrated effectiveness when history is available (Pan, 2022; Elbaum, 2002).

3. Cold-Start and Feature-based Heuristics: For newly generated tests or projects without historical data, we use heuristics weighted by empirical insights: tests that target changed files, tests with stronger assertions (as measured by assertion-specific metrics), tests covering recently modified focal methods, and tests exercising high-risk configurations receive higher priority (Souto, 2017; He, 2024).

4. Cost-Aware Ranking: Execution cost (time and environment setup) is explicitly modeled. The prioritization objective is framed as maximizing expected fault-detection per unit cost, ensuring efficient use of limited CI/CD cycles (Elbaum, 2000).

5. Adaptive Re-ranking: The prioritization model is periodically retrained as new execution outcomes accumulate, enabling the system to adapt to shifting fault distributions and codebase evolution (Pan, 2022).

By combining supervised learning with principled heuristics, the system addresses both mature and cold-start scenarios and aligns with literature advocating ML for test selection (Pan, 2022).

Orchestration, Execution, and Evaluation

Finally, the framework emphasizes realistic test execution and rigorous evaluation. Orchestration must manage sandboxed environments, web-based interactions where applicable, deterministic seeding for stochastic components, and continuous monitoring for flakiness.

Important considerations:

1. Realistic Environment Simulation: LLMs deployed as agents often operate in web or external-API contexts. The test environment must simulate web interactions, network delays, and third-party service behaviors faithfully—drawing on recent web-agent research that stresses realism in evaluation environments (Nakano et al., 2021; Gur et al., 2023; Zhou et al., 2023). When possible, replay logs or mock services that replicate observed behavior distributions should be used.

2. Determinism and Seeding: For tests exercising LLMs or stochastic components, controlled random seeds and deterministic sampling strategies are used to enable reproducible evaluation while still exploring output variability through structured perturbation experiments (Wang, 2024).

3. Flakiness Detection and Repair: Tests with intermittent failures are identified via repeated executions and statistical heuristics, then targeted for repair—either by strengthening assertions, mocking unstable dependencies, or re-specified preconditions (Lops et al., 2024).

4. Evaluation Metrics: Evaluation should include traditional metrics (fault detection rate, time-to-first-failure, code coverage) and model-aware metrics (assertion robustness, hallucination incidence in LLM-driven outputs, and requirement-trace coverage). Comparative baselines include human-written tests, conventional automated test generators, and hybrid human-LLM pipelines (Tufano, 2022; Lops et al., 2024).

5. Human-in-the-Loop Feedback: Test outcomes are surfaced to developers with interpretable explanations and focal-method mappings to enable rapid understanding and remediation. This human feedback loop also informs retraining of prioritization models and refinement of prompt templates (Chandra, 2025).

RESULTS

This section presents descriptive analyses and thought-experiment results grounded in the literature and the methodological prescriptions above. Given the article's conceptual nature, quantitative claims are framed as reasoned expectations supported by citations rather than outcomes from a single controlled empirical study. Nonetheless, we synthesize findings from prior empirical works and the theoretical mechanisms proposed to demonstrate plausible gains and trade-offs.

Improved Requirement Coverage and Traceability

By starting from structured requirement extraction and mapping, the framework enhances traceability between tests and requirement intents. dos Santos (2020) demonstrated the value of systematic approaches to requirement-derived testing; when combined with model-based intent extraction, the expectation is a measurable increase in the percentage of requirements linked to at least one executable test. Specifically, semi-formal augmentation and human validation reduce missed preconditions—a common source of silent failures that traditional generation approaches overlook (dos Santos, 2020). We therefore predict an uplift in requirement-to-test traceability metrics against naïve LLM generation that lacks requirement anchoring (Wang, 2024).

Higher Fault-Revealing Power with Focal-Method Guidance

Empirical work on focal methods suggests that tests explicitly designed around focal methods are more effective at locating faults in targeted code units (Tufano, 2022; He, 2024). In our pipeline, focal-method mapping serves two purposes: guiding generation to exercise central behaviors and enabling prioritization to emphasize high-impact tests. Drawing from He et al. (2024), we anticipate that LLM-generated tests tied to confirmed focal methods will have higher per-test fault-revealing rates compared to broad, untargeted test generation. The expected mechanism is improved alignment between assertions and the core behavior under test.

Efficiency Gains from ML-Based Prioritization

Test-suite prioritization research shows that ordering tests to maximize early fault detection can dramatically reduce developer turnaround time (Elbaum, 2002). Our framework extends this by encoding LLM-specific features (e.g., assertion strength, model-origin metadata) and using supervised learning where history exists (Pan, 2022). The literature supports that ML models can outperform simple heuristics when trained on representative data (Pan, 2022). Thus, we project that, for projects with sufficient historical data, the proposed prioritization will reduce time-to-detection metrics and increase the proportion of faults found in early execution slices in CI environments.

Configurability Management via Sampling and Soundness Trade-offs

Configurable systems present combinatorial explosion risks; Souto et al. (2017) discuss balancing soundness and efficiency. Our approach mitigates explosion by combining constraint-guided sampling, risk-based prioritization of configurations, and soundness checks via semi-formal constraints. Thought experiments and prior literature indicate that sampling approaches—if guided by domain knowledge and historical fault distributions—achieve favorable coverage-effort trade-offs versus exhaustive strategies (Agh, 2024; Souto, 2017). We thus expect test suites that strategically sample configurations to reveal a majority of high-probability faults with significantly less execution cost.

Handling LLM-Induced Non-Determinism and Flakiness

LLMs introduce non-determinism that can result in flakiness. Wang (2024) and Lops et al. (2024) both note the need for hardened assertions and multi-step verification. Our multi-model cross-validation and assertion hardening reduce the incidence of spurious failures and increase the signal-to-noise ratio for fault detection. We expect reduced flakiness rates relative to naive LLM-generated assertions, particularly when deterministic seeding and environment mocking are employed (Wang, 2024).

DISCUSSION

In this section we interpret the methodological choices and their implications, confront potential limitations, offer counter-arguments, and suggest concrete directions for empirical validation and future improvements.

Interpretation and Theoretical Contributions

The framework synthesizes multiple strands of testing research into a cohesive pipeline tailored to modern development contexts. The primary theoretical contribution is elucidating how LLMs can be incorporated without sacrificing test quality: by anchoring generation to semi-formal requirements, by using focal-method mapping to increase assertion relevance, and by integrating ML prioritization to manage resource constraints. This aligns with and extends the findings of prior works: Wang (2024) recognized the promise and hazards of LLMs in testing; Tufano (2022) and He (2024) demonstrated the efficacy of focal-method approaches; and Pan (2022) provided evidence for ML-driven prioritization. Our synthesis shows how these elements interact to produce an operationally useful pipeline.

Limitations and Counter-Arguments

No framework is without limitations. We identify several and discuss mitigations.

1. Dependence on High-Quality Requirements Artifacts: The framework presumes the availability of meaningful requirements texts or domain knowledge. In settings with poor documentation, the requirement extraction step could produce weak intents, propagating errors downstream. Mitigation includes stronger human-in-the-loop validation and leveraging runtime usage logs to infer likely requirements (dos Santos, 2020).

2. Model Hallucination and Assertion Soundness: LLMs can hallucinate plausible but incorrect assertions. While assertion hardening and multi-model cross-validation reduce this risk, they cannot eliminate it. Continuous monitoring of assertion validity and fallback to human review for high-risk tests remain essential (Wang, 2024; Lops et al., 2024).

3. Cold-Start for Prioritization Models: In new projects, historical data for supervised ranking may be unavailable. Our heuristic fallback aims to be conservative, but there is no guarantee of optimal ordering. Bootstrapping by transfer learning from related projects or using active learning to quickly gather labeled data may reduce this shortfall (Pan, 2022).

4. Evaluation Realism: Simulating web agents and external services faithfully is challenging; discrepancies between test and production environments can mask faults. Recent research on web-agent environments emphasizes realism but also acknowledges the difficulty of perfect emulation (Nakano et al., 2021; Gur et al., 2023; Zhou et al., 2023). Investment in high-fidelity mocks and staged rollouts can alleviate this.

5. Computational and Organizational Costs: Running extensive LLM-driven generation and multi-model validation is computationally expensive and may not be feasible for small teams. Cost-aware prioritization partly mitigates runtime costs, but organizational adoption requires investment in infrastructure and process changes. Leveraging cloud-native, pay-per-use inference and selective generation can make the approach economically viable for diverse organizations.

Future Research Directions

We recommend several empirical and methodological research directions to further validate and refine the framework.

1. Large-Scale Controlled Experiments: Comparative studies that evaluate the proposed pipeline against human-only test generation, classic generation tools, and naïve LLM generation across multiple domains (e.g., web apps, data pipelines, ML-enabled services) will quantify benefits and costs. Metrics should include fault-detection rates, flakiness, execution cost, and developer time-to-fix.

2. Human–LLM Collaboration Studies: Investigate how developers interact with LLM-generated tests: which artifacts they accept, how they edit assertions, and how trust evolves over time. This will inform both UI design and mechanisms for human oversight (Tufano, 2022).

3. Domain Adaptation for Prioritization Models: Research into transfer learning and federated approaches to leverage cross-project data for cold-start prioritization could improve initial performance while respecting privacy constraints (Pan, 2022).

4. Robustness Against Adversarial Inputs: LLM-driven tests may inadvertently codify brittle expectations; evaluating robustness to adversarially crafted inputs and modeling worst-case behaviors is necessary, particularly in safety-critical systems.

5. Lifecycle Integration: Explore integrating the pipeline into continuous delivery pipelines with automated triggers for test regeneration on code or requirement changes, while measuring downstream developer productivity impacts (Chandra, 2025).

CONCLUSION

Large language models are reshaping the testing landscape, offering automation opportunities that can dramatically alter how tests are produced, prioritized, and executed. However, realizing this potential requires careful, principled frameworks that combine requirement-level rigor, focal-method awareness, ML-driven prioritization, and realistic orchestration. This article presented a comprehensive methodology that synthesizes established testing literature with recent LLM and web-agent research to build an adaptive pipeline for contemporary software testing needs.

Key takeaways include: (1) beginning testing at the requirement level and preserving traceability is crucial for meaningful test generation (dos Santos, 2020); (2) focal-method mapping significantly increases the efficacy of generated tests (Tufano, 2022; He, 2024); (3) machine learning can successfully guide prioritization when historical data exists, but robust heuristics are necessary for cold-start scenarios (Pan, 2022); and (4) environmental realism and assertion hardening are essential when LLMs are used to generate or participate in tests (Wang, 2024; Lops et al., 2024). Collectively, these points form a roadmap for research and industry practitioners aiming to leverage LLMs responsibly and effectively in software testing.

The next steps are rigorous empirical validation across domains, focused studies on developer trust and collaboration with LLMs, and engineering work to make the pipeline accessible to organizations of varying sizes. By aligning the strengths of LLMs with principled testing and prioritization strategies, the field can harness new automation gains while preserving the reliability and interpretability necessary for high-quality software.

REFERENCES

1. Wang, J.; Huang, Y.; Chen, C.; Liu, Z.; Wang, S.; Wang, Q. Software Testing With Large Language Models: Survey, Landscape, and Vision. IEEE Trans. Softw. Eng. 2024, 50, 911–936.
2. dos Santos, J.; Martins, L.E.G.; de Santiago Júnior, V.A.; Povoa, L.V.; dos Santos, L.B.R. Software requirements testing approaches: A systematic literature review. Requir. Eng. 2020, 25, 317–337.
3. Agh, H.; Azamnouri, A.; Wagner, S. Software product line testing: A systematic literature review. Empir. Softw. Eng. 2024, 29, 146.
4. Souto, S.; D'Amorim, M.; Gheyi, R. Balancing Soundness and Efficiency for Practical Testing of Configurable Systems. In Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), Buenos Aires, Argentina, 20–28 May 2017; pp. 632–642.
5. Tufano, M.; Deng, S.K.; Sundaresan, N.; Svyatkovskiy, A. Methods2Test: A dataset of focal methods mapped to test cases. In Proceedings of the 19th International Conference on Mining Software Repositories, MSR'22, Pittsburgh, PA, USA, 23–24 May 2022; pp. 299–303.
6. Pan, R.; Bagherzadeh, M.; Ghaleb, T.A.; Briand, L. Test case selection and prioritization using machine learning: A systematic literature review. Empir. Softw. Eng. 2022, 27, 29.
7. He, Y.; Huang, J.; Yu, H.; Xie, T. An Empirical Study on Focal Methods in Deep-Learning-Based Approaches for Assertion Generation. Proc. ACM Softw. Eng. 2024, 1, 1750–1771.
8. Elbaum, S.; Malishevsky, A.G.; Rothermel, G. Prioritizing test cases for regression testing. Sigsoft Softw. Eng. Notes 2000, 25, 102–112.
9. Elbaum, S.; Malishevsky, A.G.; Rothermel, G. Test Case Prioritization: A Family of Empirical Studies. IEEE Trans. Softw. Eng. 2002, 28, 159–182.
10. Lops, A.; Narducci, F.; Ragone, A.; Trizio, M.; Bartolini, C. A System for Automated Unit Test Generation Using Large Language Models and Assessment of Generated Test Suites. arXiv 2024, arXiv:2408.07846.
11. Schick, T.; Dwivedi-Yu, J.; Dessì, R.; Raileanu, R.; Lomeli, M.; Zettlemoyer, L.; Cancedda, N.; Scialom, T. Toolformer: Language models can teach themselves to use tools. arXiv preprint, 2023.
12. Nakano, R.; Hilton, J.; Balaji, S.; Wu, J.; Ouyang, L.; Kim, C.; Hesse, C.; Jain, S.; Kosaraju, V.; Saunders, W. WebGPT: Browser-assisted question-answering with human feedback. arXiv preprint, 2021.
13. Chandra, R.; Lulla, K.; Sirigiri, K. Automation frameworks for end-to-end testing of large language models (LLMs). Journal of Information Systems Engineering and Management, 2025, 10, e464-e472.
14. Gur, I.; Furuta, H.; Huang, A.; Safdari, M.; Matsuo, Y.; Eck, D.; Faust, A. A real-world webagent with planning, long context understanding, and program synthesis. arXiv preprint, 2023.
15. Zhou, S.; Xu, F.F.; Zhu, H.; Zhou, X.; Lo, R.; Sridhar, A.; Cheng, X.; Bisk, Y.; Fried, D.; Alon, U. Webarena: A realistic web environment for building autonomous agents. arXiv preprint, 2023.
16. Lu, P.; Peng, B.; Cheng, H.; Galley, M.; Chang, K.-W.; Wu, Y. N.; Zhu, S.-C.; Gao, J. Chameleon: Plug-and-play compositional reasoning with large language models. arXiv preprint, 2023.
17. Deng, X.; Gu, Y.; Zheng, B.; Chen, S.; Stevens, S.; Wang, B.; Sun, H.; Su, Y. Mind2web: Towards a generalist agent for the web. arXiv preprint, 2023.
18. He, H.; Yao, W.; Ma, K.; Yu, W.; Dai, Y.; Zhang, H.; Lan, Z.; Yu, D. WebVoyager: Building an end-to-end web agent with large multimodal models. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 6864–6890. Available: https://aclanthology.org/2024.acl-long.371