

Characterizing Large Language Model Failure Modes in Code Refactoring and Multi-Turn Reasoning: A Controlled Study Design

¹ Stanley Nwakamma, ² Serif Oyindamola Oyesiji, ³ Kingsley Chinazaekpere Ndupu, ⁴ Emmanuel Eniola Ajala.

¹ Independent Researcher, USA; nwakammastanley@gmail.com

² Harrisburg University of Science and Technology, Harrisburg, PA; oyesijiserif@gmail.com

³ Kennesaw State University, Georgia, USA; kingsleynd34@gmail.com

⁴ Independent Researcher, Atlanta, Georgia; emmanuelajala22@gmail.com

Corresponding Author; nwakammastanley@gmail.com

Abstract: Large language models are increasingly used not to write code from scratch but to change code that already works: renaming and restructuring, extracting functions and modules, migrating interfaces, and tightening designs across successive conversational turns. Refactoring is a demanding target for such assistance because its defining contract, behavior preservation under structural change, is exactly the property that generative models do not natively guarantee, and because realistic refactoring is multi-turn, with requirements, corrections, and constraints arriving incrementally in a dialogue whose earlier content the model must keep honoring. Existing code generation benchmarks measure single-shot synthesis against input-output tests and therefore see neither the preservation contract nor the conversational dynamics. This paper develops a research concept for a systematic empirical study of failure modes in LLM-assisted code refactoring under multi-turn interaction. The concept contributes a failure taxonomy spanning behavior-preservation violations, scope violations, constraint forgetting across turns, instruction-following decay, regression reintroduction, sycophantic acceptance of incorrect user assertions, and self-repair pathologies; a task corpus of refactoring scenarios over real codebases with layered verification, regression suites, differential and metamorphic checks, and refactoring-type-specific invariants; a controlled multi-turn protocol with scripted personas that inject corrections, contradictions, distractions, and erroneous claims at defined points; and an annotation and metrics framework that localizes each failure to taxonomy category, turn, and trigger. Prespecified hypotheses target constraint decay with conversation length, the divergence between edit locality instructed and edit locality performed, and the conditions under which iterative repair converges rather than oscillates. Applications to model evaluation, tooling design, and review practice are analyzed alongside limitations.

1. Introduction

The economics of software have always favored modification over creation: most engineering effort is spent evolving systems that exist, an observation formalized in the laws of software evolution (Lehman, 1980) and institutionalized in the practices of modular design and disciplined restructuring (Parnas, 1972; Fowler, 2018). Refactoring, behavior-preserving transformation of code structure, was given its systematic treatment decades ago (Opdyke, 1992; Fowler, 2018), surveyed as a research field (Mens & Tourwé, 2004), and studied empirically at industrial scale, where its practice is frequent, valued, and error-prone even for humans with tool support (Murphy-Hill et al., 2012; Kim et al., 2014), with modern mining infrastructure making refactoring activity detectable in version histories (Tsantalis et al., 2018). The debt metaphor explains its motivation, structure is paid for later if not maintained now (Cunningham, 1992; Kruchten et al., 2012), and code review is the human quality gate through which structural change passes (Bacchelli & Bird, 2013), a gate now mediated by version-controlled pipelines, branch and merge governance, and automated deployment tooling whose configuration determines which structural changes reach production at all (Badmus, Dosunmu, & Ozowara, 2018; Badmus, Jooda, Ozowara, & Anunagba, 2022a; Mbonu, Iwuanyanwu, Aliliele, & Uzoka, 2020b).

Language models entered this workflow rapidly. Code-capable models progressed from synthesis demonstrations (Chen et al., 2021; Li et al., 2022; Nijkamp et al., 2023; Rozière et al., 2023) into general assistants used conversationally inside development environments (Brown et al., 2020; Ouyang et al., 2022; OpenAI, 2023), and are now being wired into enterprise workflows as tool-using agents whose autonomy raises governance, confidentiality, and interface-design questions well beyond the accuracy of any single completion (Nwakamma, Ojukwu, & Oyesiji, 2024b; Ogundairo & Ayivi-Donkor, 2024b; Ominyi, Anichukwueze, & Uzougbo, 2025c), and the evaluation ecosystem followed the synthesis framing: single-shot generation against test suites (Chen et al., 2021; Austin et al., 2021; Hendrycks et al., 2021), hardened by stronger tests (Liu et al., 2023), protected against contamination (Jain et al., 2024), and extended to repository-scale issue resolution (Jimenez et al., 2024), on encoder and encoder-decoder foundations built for code understanding tasks (Feng et al., 2020; Guo et al., 2021; Wang et al., 2021; Lu et al., 2021). Repair-oriented studies established that models can fix bugs competitively with dedicated program repair systems in some regimes (Xia & Zhang, 2022; Fan et al., 2023; Sobania et al., 2023), and iterative techniques, self-refinement, reflective agents, and execution-guided

self-debugging, defined the multi-step usage pattern (Madaan et al., 2023; Shinn et al., 2023; Chen et al., 2024), with careful analyses showing its benefits are real but conditional (Olausson et al., 2024).

What this apparatus does not measure is the refactoring contract under conversational dynamics. Behavior preservation is invisible to pass-at-k on synthesis tasks; scope discipline, change only what was asked, has no metric at all; and multi-turn behavior is evaluated mainly through judge-scored dialogue quality (Zheng et al., 2023) rather than through verified code outcomes as instructions accumulate, despite documented mechanisms that predict trouble: positional weaknesses over long contexts (Liu et al., 2024), sensitivity to prompt form and order (Sclar et al., 2024; Lu et al., 2022), sycophantic deference to user assertions (Sharma et al., 2024; Turpin et al., 2023), and unfaithful self-reports about reasoning (Lanham et al., 2023). Human-factors findings raise the stakes, since developers over-accept plausible model output (Vaithilingam et al., 2022; Barke et al., 2023; Dakhel et al., 2023; Perry et al., 2023) and generated code carries measurable defect and vulnerability rates (Pearce et al., 2022), even as applied work reports real gains from AI-assisted testing and automated regression control in settings where the assistance is bounded by an explicit verification regime rather than trusted on its own (Borah, Aliliele, Rakshit, & Vajjhala, 2022; Mbonu, Aliliele, Iwuanyanwu, & Uzoka, 2021b). This paper specifies the study that fills the gap: a controlled, verification-grounded characterization of how LLM refactoring fails across turns. Section 2 reviews background; Section 3 defines the taxonomy; Sections 4 through 7 specify the study design, corpus, protocol, and metrics; Section 8 states hypotheses; Sections 9 and 10 treat implications and limitations; Section 11 concludes.

2. Background and Related Work

Refactoring research fixes the contract and its verification problem. The canonical catalogues define transformation types and their mechanics (Fowler, 2018; Opdyke, 1992), the survey literature organizes supporting techniques from detection to correctness (Mens & Tourwé, 2004), and the empirical strand documents practice: refactoring is interleaved with other change, frequently performed without tools, and a recognized source of regressions, which is why version-history detection matters for studying it at scale (Murphy-Hill et al., 2012; Kim et al., 2014; Tsantalis et al., 2018). Verification of behavior preservation inherits software testing's fundamentals: regression suites as the first line (Yoo & Harman, 2012), their adequacy judged by fault-revealing power rather than coverage alone (Jia & Harman, 2011; Just et al., 2014), the oracle problem as the standing limit on what testing can certify (Barr et al., 2015), and property-based and metamorphic techniques as oracles where examples run out (Claessen & Hughes, 2000; Chen et al., 2018). Applied testing work supplies the engineering constraints under which such oracle stacks are actually assembled and run, from scripting frameworks and instrument control in automated test benches to structured user-acceptance validation in deployment settings (Quainoo, Ogundapo, & Asiedu, 2025b; Eyetsemitan, Ambali, Oyeleye, & Fadayomi, 2023b). This stack defines the study's measurement layer: preservation violations are detectable exactly to the strength of the verification bundle, and the bundle must therefore be layered and its sensitivity audited.

LLM-for-code research supplies the treatment side. Synthesis benchmarks and their hardening establish measurement conventions the study extends rather than repeats (Chen et al., 2021; Liu et al., 2023; Jain et al., 2024), repository-scale benchmarks demonstrate evaluation over real codebases with real test suites, the setting refactoring requires (Jimenez et al., 2024), and repair studies establish both capability and characteristic error patterns when models modify existing code under test feedback (Xia & Zhang, 2022; Fan et al., 2023; Sobania et al., 2023), a framing that adjacent applied work extends into security engineering and defensive automation, where models reason over existing artifacts rather than generate them fresh, and into natural language processing pipelines whose reliability rests on the same evaluation discipline (Odejebi, Okonkwo, Ahiaekwe Patrick, Okeke, & Mayo, 2025; Hussain & Adebayo, 2026; Atakpa, Abolaji, & Akeju, 2023). The iterative-usage literature defines the loop structure the protocol instruments, refine, reflect, debug against execution signals (Madaan et al., 2023; Shinn et al., 2023; Chen et al., 2024), and its critical strand identifies the pathologies the taxonomy names: repair that plateaus, oscillates, or trades one fault for another, with net benefit contingent on feedback quality (Olausson et al., 2024). A parallel governance literature treats model-generated evaluation as itself an object of scrutiny, cataloguing the biases and release-gating practices that arise when models are used to judge model output, which is precisely the dependency a study of conversational failure must avoid inheriting (Nwakamma, Ojukwu, & Oyesiji, 2024a).

The behavioral mechanisms literature motivates the multi-turn design. Long-context studies show mid-context information is systematically underused (Liu et al., 2024), predicting decay of early-turn constraints; prompt-sensitivity studies show that semantically irrelevant variation moves outcomes (Sclar et al., 2024; Lu et al., 2022), predicting instability across conversational phrasings; sycophancy studies show models revise correct positions under user pushback and adopt user-asserted errors (Sharma et al., 2024), predicting acceptance of false claims about code behavior; and faithfulness studies caution that model explanations of their own edits cannot be taken as mechanisms (Turpin et al., 2023; Lanham et al., 2023), which the annotation framework respects by grounding every failure in verified code state rather than in model narration. The same caution motivates treating human judgment as a measured rather than assumed input, following work on human-in-the-loop pipelines, human-centered annotation under generative assistance, and the interpretability requirements that make automated decisions auditable at all (Ladapo, Dosunmu, Jooda, & Abolaji, 2022; Ladapo, Jooda, Dosunmu, & Abolaji, 2024; Sanni, Iwuanyanwu, & Essien, 2026; Komi & Adamolekun, 2021). Multi-turn evaluation practice to date scores conversational quality with model judges (Zheng et al., 2023), a practice whose

adaptation, curation, and deployment constraints the applied language-model literature has begun to map across model scales and serving environments (Oyesiji, Nwakamma, & Ojukwu, 2023, 2024, 2025; Nwakamma, Ojukwu, & Oyesiji, 2025); the study's departure is to make the conversation the independent variable and verified program behavior the dependent one.

3. Failure Mode Taxonomy

The taxonomy is organized by the contract each failure violates, and it is the study's coding scheme as well as its conceptual contribution. Behavior-preservation violations: the refactored code fails regression, differential, or metamorphic checks that the pre-change code passed, subdivided by surface, functional regressions, interface breaks, and semantic drift in edge behavior that only property-level oracles catch (Barr et al., 2015; Chen et al., 2018). Interface breaks matter disproportionately in integrated systems, where a published contract is consumed by parties the refactoring author cannot see, which is why API governance and integration-pattern discipline treat the published surface as a controlled asset rather than an implementation detail (Aliliele, Mbonu, & Iwuanyanwu, 2023b; Badmus, Jooda, Ozowara, & Anunagba, 2022b). Scope violations: edits exceed the instructed change locality, gratuitous rewrites, reformatting beyond the target, drive-by modifications to untargeted units, measurable as the difference between instructed and realized edit sets, a category with no analogue in synthesis evaluation and central to refactoring discipline (Fowler, 2018; Murphy-Hill et al., 2012). Constraint forgetting: a requirement established in an earlier turn, preserve this public signature, do not touch that module, keep compatibility with this caller, is violated in a later turn, the conversational decay mechanism predicted by long-context findings (Liu et al., 2024). Instruction-following decay: progressive drift in adherence to standing stylistic or procedural instructions across turns even without contradiction. Regression reintroduction: a defect fixed in an earlier turn reappears after later edits, the code-level signature of lost conversational state. Sycophantic incorporation: the model accepts a user assertion that is false, about current behavior, about test status, about a language or library semantics claim, and edits accordingly, the code consequence of documented sycophancy (Sharma et al., 2024). Analogous incorporation risks arise wherever a semi-autonomous system acts on unverified assertions, which is why guardrail and orchestration designs place validation between assertion and action rather than after it (Ogundairo, 2025; Nwakamma, Ojukwu, & Oyesiji, 2024b). Self-repair pathologies: under iterative feedback, oscillation between states, plateau below correctness with confident completion claims, and repair that fixes the reported symptom while violating preservation elsewhere (Olausson et al., 2024; Chen et al., 2024). Categories are not exclusive per episode; the annotation unit is the failure event, localized to turn, triggered condition, and verified evidence. This organization follows the general pattern of accountable decision frameworks, in which an automated decision is auditable only when its trigger, its evidence, and its control point are recorded as separate fields rather than collapsed into an outcome label (Shah, Oziri, & Seyi-Lande, 2025; Odejide, 2024; Aliliele, Mbonu, Uzoka, & Iwuanyanwu, 2025b).

4. Study Design Overview

The study is a controlled multi-factor experiment. Units are refactoring episodes: a codebase state, a refactoring objective, a scripted multi-turn interaction, and a model configuration. Factors are refactoring type, drawn from the catalogue families, rename and move, extract and inline, interface migration, structural decomposition (Fowler, 2018); conversation script class, cooperative, corrective, contradictory, distracting, and adversarially misinformed, defined in Section 6; conversation length and constraint density; codebase scale and test strength strata; and model and decoding configuration, with the execution environment held fixed so that infrastructure variability does not confound model effects, a control standard familiar from predictive modeling and capacity-planning studies in which platform elasticity silently alters measured outcomes (Ahmed & Odejobi, 2018a; Ahmed, Odejobi, & Oshoba, 2020; Edivri & Oteri, 2022). Outcomes are verified code states after every turn, scored by the verification harness of Section 5 and annotated under the taxonomy of Section 3. The design is fully scripted rather than user-observational: personas inject the same turns at the same points across models and repetitions, trading naturalism for the attribution the research questions require, with observational replication over real developer sessions named as follow-on work rather than smuggled into this design. Repetition across sampling seeds quantifies stochastic variance, reported as an uncertainty band rather than a point estimate in line with forecasting practice for stochastic systems (Wedraogo & Sanni, 2024; Mayo, Ogbole, Okoruwa, & Babatope, 2021), and all prompts, scripts, harnesses, and annotations are versioned for exact reproduction, the reporting posture repository-scale evaluation has normalized (Jimenez et al., 2024; Jain et al., 2024).

5. Task Corpus and Verification Harness

The corpus is built from real codebases rather than synthetic snippets, sampled across the scale strata from single-module libraries to multi-package projects, because refactoring difficulty lives in coupling and context that toy programs lack (Jimenez et al., 2024; Kim et al., 2014). Each episode's starting state is a pinned commit with a passing test suite; the refactoring objective is specified as a developer would state it, in natural language with the target units named; and the ground-truth change locality, the file and symbol set a faithful execution would touch, is authored per episode, giving the scope metrics of Section 7 their reference. Refactoring-type coverage is enforced across the catalogue families, and, where history supports it, episodes are seeded from

refactorings mined out of the project's own past commits (Tsantalis et al., 2018), anchoring objectives in changes the codebase actually underwent. Corpus construction is itself an extraction, transformation, and loading problem, with provenance, schema stability, and reproducible builds as first-order concerns rather than incidental ones (Badmus, Dosunmu, & Ozowara, 2019a; Badmus, Dosunmu, Ozowara, & Anunagba, 2020).

The verification harness layers oracles in the pattern the testing literature prescribes. The project's regression suite is the base layer, with suite strength measured by mutation score so that preservation claims carry their sensitivity bound rather than an assumption (Jia & Harman, 2011; Just et al., 2014; Yoo & Harman, 2012). A differential layer executes pre- and post-change code on generated inputs, property-based where input structure permits (Claessen & Hughes, 2000), asserting output equivalence for behavior-preserving objectives. A metamorphic layer covers surfaces examples miss, equivalence under input transformations and invariants specific to the refactoring type, extracted functions must compose to the original, moved units must preserve call semantics, interface migrations must satisfy adapter equivalence on the legacy surface (Chen et al., 2018; Barr et al., 2015). Static layers complete the bundle: compilation and type checks, public-interface diffing against declared compatibility constraints, and edit-set extraction for scope measurement. Compatibility constraints are expressed against a single documented standard per episode, since divergence between nominally equivalent standards is a well-documented source of false conformance findings in regulated assurance settings (Aneke, Adesemoye, & Sydney, 2024). Every layer runs after every turn, so the study observes trajectories of verified state, not just endpoints. Instrumentation follows the pattern established in operational telemetry and automated response systems, where continuous capture of state transitions, rather than periodic sampling, is what makes post hoc attribution possible (Mbonu, Iwuanyanwu, Uzoka, & Oluoha, 2019; Dosunmu & Ogundele, 2025c). Execution isolation and portability across language runtimes are handled at the harness boundary so that episodes drawn from different stacks remain comparable (Ojukwu, Oyesiji, & Nwakamma, 2025), with the network and deployment surface of each sandbox constrained by the same reasoning applied to distributed enterprise environments and to workload classification in containerized systems (Ogunwola, 2026; Idika, Salami, Ijiga, & Enyejo, 2021).

6. Multi-Turn Interaction Protocol

Conversation scripts operationalize the study's behavioral probes as persona classes with defined injection points. The cooperative script issues the objective, then plausible follow-on requests within scope, measuring baseline multi-turn execution and instruction persistence. The corrective script reports genuine harness-detected faults back to the model in developer language, instantiating the iterative repair loop under controlled feedback quality (Chen et al., 2024; Olausson et al., 2024). The contradictory script introduces a mid-conversation requirement change that legitimately conflicts with an earlier constraint, measuring whether the model surfaces the conflict or silently violates one side. The distracting script interleaves in-scope work with unrelated questions and tangents, probing constraint retention across intervening content, the conversational analogue of the long-context and irrelevant-context findings (Liu et al., 2024; Sclar et al., 2024). Scripted progression through material of controlled difficulty mirrors the sequencing logic of adaptive instructional systems, in which what is presented next is a function of what has already been established and retained (Onwubuya et al., 2026; Seun et al., 2024; Joseph & Uzodu, 2024a). The adversarially misinformed script asserts falsehoods with confidence, that a test already passes, that an API behaves otherwise than it does, that a change was already made, measuring sycophantic incorporation against verifiable ground truth (Sharma et al., 2024). This script is adversarial in the threat-modeling sense rather than the security sense: the input is crafted to move behavior away from ground truth, and its structure borrows from adversarial evaluation frameworks that fix attacker capability, injection point, and success criterion in advance (Adebayo, Adegbite, & Ahmed, 2022; Nwakamma, Ojukwu, & Oyesiji, 2024b). Script wording is held constant across models and, where episodes run in more than one natural language, rendered under a controlled translation protocol, since phrasing and accessibility choices measurably change how instructions are received (Odejide, 2023).

Scripts are parameterized by length and constraint density, with constraints registered at scripted turns and their observance mechanically checkable by the harness's interface and edit-set layers, so that constraint forgetting is measured by verification, not judgment. Injection points are fixed relative to conversation structure, early, middle, late, enabling positional analysis of decay. Model-side conditions cross the scripts with decoding regimes and with context-management variants, full history versus summarized history, because context handling is itself a deployed design choice whose effect on constraint retention the study can estimate directly.

7. Annotation and Metrics

Measurement is two-layered: mechanical metrics computed by the harness, and taxonomy annotation grounded in them. Mechanical metrics per turn include preservation status per oracle layer with the suite's mutation-score bound attached; scope divergence, precision and recall of the realized edit set against the authored locality reference; constraint compliance vectors over all registered constraints; interface compatibility status; and, across turns, trajectory statistics, turns-to-success, oscillation counts between verified states, regression reintroduction events, and final-state quality. Conversation-level metrics aggregate by script class and injection position, yielding the decay curves, compliance as a function of turns since constraint registration, that the hypotheses

target. Because every metric is computed from a timestamped harness record rather than from a summary, the metric set supports process-level reconstruction of how a given failure arose, the auditability property that process mining and control-oriented reporting frameworks are built to deliver (Sanni, Iwuanyanwu, & Essien, 2025; Edivri & Oteri, 2024; Mbonu, Aliliele, Iwuanyanwu, & Uzoka, 2022; Fobellah, 2025b).

Taxonomy annotation assigns failure events to categories with turn, trigger, and evidence fields, performed by two annotators per episode from the harness record and transcript, with disagreement adjudicated and inter-annotator agreement reported before any substantive analysis, treating annotator disagreement as a measurable property of the coding scheme rather than as noise to be averaged away (Ladapo, Jooda, Dosunmu, & Abolaji, 2026), since the taxonomy's reliability bounds every claim built on it. Model self-explanations are recorded but never used as evidence of mechanism, only as an object of study in their own right, scored for accuracy against the verified record in line with the faithfulness caution (Lanham et al., 2023; Turpin et al., 2023). Analysis prespecifies mixed-effects structure, episodes and codebases as random effects, factors as fixed effects, with stochastic repetition supplying variance estimates, and all comparisons across models are paired on identical episode-script combinations.

8. Hypotheses

Six hypotheses are prespecified. First, constraint compliance decays with turns since registration, steepest when intervening turns are distracting, the conversational instantiation of positional underuse (Liu et al., 2024). Second, scope divergence is nonzero even in cooperative scripts and grows with codebase scale, with realized edit sets exceeding instructed locality more often by addition than by omission, quantifying the gratuitous-rewrite tendency practitioners report (Barke et al., 2023; Murphy-Hill et al., 2012); the practitioner-side counterpart is an adoption gap in which capability is assumed and verification is not, a pattern documented across settings where organizations deploy AI faster than they build the readiness to supervise it (Ogundairo & Ayivi-Donkor, 2024a; Afrihyia, Akinse, & Ojukwu, 2025; Akintola, Fawehinmi, Aiyenitaju, Chinemerem, & Emmanuella, 2025). Third, sycophantic incorporation rates under adversarially misinformed scripts are substantial and rise when the false assertion is delivered with correction framing, following documented sycophancy dynamics (Sharma et al., 2024). Fourth, corrective-loop convergence is conditional on feedback specificity: fault reports naming failing properties converge faster and oscillate less than generic failure reports, consistent with the feedback-quality dependence of self-repair (Olausson et al., 2024; Chen et al., 2024). Fifth, preservation violations detected only by the metamorphic and differential layers, invisible to the project suite, constitute a material fraction of all violations, quantifying the oracle gap for refactoring claims (Chen et al., 2018; Jia & Harman, 2011). Sixth, summarized-history context management trades reduced late-conversation constraint forgetting against increased early-detail loss, yielding measurably different failure profiles rather than uniform improvement. Each hypothesis names its metric, contrast, and direction; disconfirmations are reportable results.

9. Anticipated Implications

For evaluation practice, the study supplies what synthesis benchmarks omit: preservation, scope, and conversational persistence as first-class measured properties, with a released corpus, scripts, and harness constituting a reusable benchmark for refactoring-under-dialogue, complementing repository-scale issue benchmarks on the modification side of the capability space (Jimenez et al., 2024; Liu et al., 2023). For tooling design, the failure profile localizes where guardrails pay: scope divergence motivates edit-set contracts enforced by the environment rather than the model; constraint decay motivates externalized constraint registries checked mechanically each turn rather than entrusted to context; sycophancy rates motivate verification-before-incorporation for user assertions about program state, all of which are implementable in current assistant architectures. Each of these guardrails instantiates the general result that automation delivers reliably when the surrounding process supplies explicit control points, standardized procedure, and defined exception handling, and not when the automated component is merely made individually stronger (Okonkwo, Agbabiaka, Mayo, & Okeke, 2024f; Akinleye & Adeyoyin, 2021; Eyetsemitan, Ambali, Oyeleye, & Fadayomi, 2025; Akanbi & Sunday, 2025). For review practice, category base rates give code reviewers of model-assisted changes an evidence-based checklist, sharpening the human gate that modern review already constitutes (Bacchelli & Bird, 2013; Perry et al., 2023). Whether such a checklist changes behavior is an adoption question as much as an evidence question, since review practice shifts only when readiness, training, and incentives shift with it (Akhigbe, Falemi, & Akin-Oluyomi, 2024; Eyetsemitan, Ambali, Oyeleye, & Fadayomi, 2023a). And for model development, decay curves and pathology rates across context-management variants provide optimization targets that conversational judge scores do not expose (Zheng et al., 2023).

10. Limitations and Threats to Validity

Construct limits are inherited and stated: preservation is certified only to the layered oracles' sensitivity, bounded by mutation score and by the chosen properties, so undetected semantic drift remains possible, and all preservation results carry their bounds (Barr et al., 2015; Jia & Harman, 2011). Scripted conversations trade ecological validity for attribution: real developers adapt to model behavior, and interaction effects between human strategy and model failure are outside this design's reach, marked for

observational follow-on rather than claimed here (Vaithilingam et al., 2022; Barke et al., 2023). Corpus external validity is bounded by language, scale strata, and project selection, and mined-refactoring seeding, while anchoring realism, biases objectives toward historically performed change types (Tsantalis et al., 2018). Annotation subjectivity is controlled but not eliminated by dual coding and agreement reporting. Contamination is a live threat for public codebases, mitigated by commit pinning, symbol renaming variants, and preference for post-cutoff project states, following contamination-aware practice (Jain et al., 2024). Disclosure limits bound this further, since proprietary training-data and configuration details that would sharpen contamination analysis are routinely withheld, and the transparency obligations now emerging in AI regulation will determine how much of that gap future replications can close (Annan, 2024; Ominyi, Anichukwueze, & Uzougbo, 2024b). Rapid model turnover dates any single campaign's numbers, which is why the durable contributions are the taxonomy, corpus, and harness, with model results versioned as snapshots against them.

11. Conclusion

This paper has developed a research concept for the systematic study of LLM failure modes in code refactoring under multi-turn interaction. Its contributions are a contract-organized failure taxonomy, from behavior-preservation and scope violations through constraint forgetting, sycophantic incorporation, and self-repair pathologies; a real-codebase task corpus with layered verification whose sensitivity is itself measured; a scripted persona protocol that makes conversational dynamics an experimental factor with fixed injection points; and a metrics and annotation framework that grounds every failure event in verified program state, with prespecified hypotheses linking the design to documented behavioral mechanisms.

The concept's premise is that modification, not synthesis, is where language models will do most of their work in software, and that the properties modification prices, preserving behavior, respecting scope, honoring accumulated constraints, resisting confident misinformation, are precisely the ones current evaluation does not measure. Executed, the study converts anecdote into base rates, base rates into guardrail requirements, and guardrail requirements into evaluable tooling claims, moving assisted refactoring from a capability demonstrated to a practice characterized, bounded, and improvable.

References

- Adebayo, A., Adegbite, M. P., & Ahmed, M. O. (2022). Adversarial machine learning in critical infrastructure: A conceptual framework for threat modeling AI enabled OT systems. *World Journal of Innovation and Modern Technology*, 6(1), 184–234. <https://doi.org/10.56201/wjimt.v6.no1.2022.pg184.234>
- Afrihyia, E., Akinse, S. G., & Ojukwu, P. U. (2025). Organizational readiness for generative AI integration in healthcare operations: Comparative management capabilities between the U.S. and low- and middle-income countries. *Iconic Research and Engineering Journals*, 8(10), 1673–1697. <https://doi.org/10.64388/IREV8I10-1714670>
- Ahmed, K. S., & Odejobi, O. D. (2018a). Conceptual framework for scalable and secure cloud architectures for enterprise messaging. *IRE Journals*, 2(1), 1–15.
- Ahmed, K. S., Odejobi, O. D., & Oshoba, T. O. (2020). Predictive model for cloud resource scaling using machine learning techniques. *Journal of Frontiers in Multidisciplinary Research*, 1(1), 173–183.
- Akanbi, O., & Sunday, E. A. (2025). A systematic review of AI-driven autonomous mobile robots (AMR) in scalable micro-fulfillment centers. *International Journal of Advanced Multidisciplinary Research and Studies*, 5(6), 2363–2379. <https://doi.org/10.62225/2583049X.2025.5.6.6182>
- Akhigbe, R., Falemi, A., & Akin-Oluyomi, O. T. (2024). An advanced framework for technology adoption that supports employee readiness in modern organizations. *International Journal of Advanced Multidisciplinary Research and Studies*, 4(1), 1637–1659. <https://doi.org/10.62225/2583049X.2024.4.1.5344>
- Akinleye, O. K., & Adeyoyin, O. (2021). Process automation framework for enhancing procurement efficiency and transparency. *Shodhsharyam, International Scientific Refereed Research Journal*, 4(4), 356–387.
- Akintola, A. S., Fawehinmi, Y. O., Aiyenitaju, O., Chinemerem, B., & Emmanuella, O. (2025). Digital transformation in telehealth: A systematic review of user trust, privacy, and regulatory governance in AI-powered remote monitoring systems. *Journal of Scientific Research and Reports*, 31(11), 163–182. <https://doi.org/10.9734/jsrr/2025/v31i113658>
- Aliliele, C., Mbonu, I. S., & Iwuanyanwu, U. (2023b). A review of API governance and risk prioritization frameworks in modern financial institutions. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 9(10), 395–433. <https://doi.org/10.32628/CSEIT2361072>

- Aliliele, C., Mbonu, I. S., Uzoka, E., & Iwuanyanwu, U. (2025b). Advances in data lakehouse governance architectures for enterprise data loss prevention and compliance assurance. *Shodhshauryam, International Scientific Refereed Research Journal*, 8(4), 171–213. <https://doi.org/10.32628/SHISRRJ258474>
- Aneke, O. B., Adesemoye, A. C., & Sydney, E. (2024). Regulatory harmonization and convergence in global medicine quality assurance: A review of frameworks, pathways, and implementation barriers. *International Journal of Medical Evaluation and Physical Report*, 8(6), 213–252.
- Annan, A. O. (2024). Algorithmic accountability and trade secret protection in artificial intelligence. *Shodhshauryam, International Scientific Refereed Research Journal*, 7(5), 315–347.
- Atakpa, M. I., Abolaji, T. O., & Akeju, B. (2023). A review of natural language processing for social media-based public health surveillance and analytics. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 9(6), 1030–1060. <https://doi.org/10.32628/CSEIT23906783>
- Austin, J., Odena, A., Nye, M., et al. (2021). Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.
- Bacchelli, A., & Bird, C. (2013). Expectations, outcomes, and challenges of modern code review. *Proceedings of the International Conference on Software Engineering*, 712–721.
- Badmus, O., Dosunmu, A. A., & Ozowara, D. E. (2018). A systematic review of CI/CD pipeline strategies in Salesforce DevOps: Tools, practices, and deployment outcomes. *Iconic Research and Engineering Journals*, 2(6).
- Badmus, O., Dosunmu, A. A., & Ozowara, D. E. (2019a). A conceptual model for ETL design and data integration in Salesforce-centric enterprise architectures. *Iconic Research and Engineering Journals*, 3(5).
- Badmus, O., Dosunmu, A. A., Ozowara, D. E., & Anunagba, C. O. (2020). A comparative framework for Salesforce DevOps tooling: Evaluating Copado, Flosom, and Salesforce DX across enterprise deployment contexts. *International Journal of Multidisciplinary Research and Growth Evaluation*, 1(5), 1021–1031. <https://doi.org/10.54660/IJMRGE.2020.1.5.1021-1031>
- Badmus, O., Jooda, D., Ozowara, D. E., & Anunagba, C. O. (2022a). A framework for Git-based version control and code review governance in Salesforce development teams. *Shodhshauryam, International Scientific Refereed Research Journal*, 5(2), 473–493.
- Badmus, O., Jooda, D., Ozowara, D. E., & Anunagba, C. O. (2022b). A systematic review of MuleSoft ESB integration patterns in multi-cloud Salesforce architectures. *Gyanshauryam, International Scientific Refereed Research Journal*, 5(3), 462–481.
- Barke, S., James, M. B., & Polikarpova, N. (2023). Grounded Copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages*, 7(OOPSLA1), 85–111.
- Barr, E. T., Harman, M., McMin, P., Shahbaz, M., & Yoo, S. (2015). The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering*, 41(5), 507–525.
- Borah, S., Aliliele, K. C., Rakshit, S., & Vajjhala, N. R. (2022). Applications of artificial intelligence in software testing. In P. K. Mallick et al. (Eds.), *Cognitive informatics and soft computing (Lecture Notes in Networks and Systems, Vol. 375, pp. 727–736)*. Springer. https://doi.org/10.1007/978-981-16-8763-1_60
- Brown, T., Mann, B., Ryder, N., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33, 1877–1901.
- Chen, M., Tworek, J., Jun, H., et al. (2021). Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.
- Chen, T. Y., Kuo, F.-C., Liu, H., et al. (2018). Metamorphic testing: A review of challenges and opportunities. *ACM Computing Surveys*, 51(1), 4.
- Chen, X., Lin, M., Schärli, N., & Zhou, D. (2024). Teaching large language models to self-debug. *Proceedings of the International Conference on Learning Representations*.
- Claessen, K., & Hughes, J. (2000). QuickCheck: A lightweight tool for random testing of Haskell programs. *Proceedings of the ACM SIGPLAN International Conference on Functional Programming*, 268–279.
- Cunningham, W. (1992). The WyCash portfolio management system. *Addendum to the Proceedings of OOPSLA*, 29–30.
- Dakhel, A. M., Majdinasab, V., Nikanjam, A., Khomh, F., Desmarais, M. C., & Jiang, Z. M. (2023). GitHub Copilot AI pair programmer: Asset or liability? *Journal of Systems and Software*, 203, 111734.

- Dosunmu, A. A., & Ogundele, P. O. (2025c). Security orchestration and automation models for accelerating incident detection and response. *Computer Science and IT Research Journal*, 6(11), 878–894. <https://doi.org/10.51594/csitrj.v6i11.2163>
- Edivri, J., & Oteri, O. (2022). Predictive capacity planning and resource utilization forecasting models for multi-program and multi-stakeholder IT portfolios. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 8(4), 826–845.
- Edivri, J., & Oteri, O. (2024). A conceptual KPI-driven decision and optimization framework for IT service delivery, portfolio performance, and adoption. *Gyanshauryam, International Scientific Refereed Research Journal*, 7(1), 167–187. <https://doi.org/10.32628/GISRRJ246643>
- Eyetsemitan, R. A., Ambali, K. B., Oyeleye, A. O., & Fadayomi, O. (2023a). Change management in small business digital transformation: A systematic review and lean change adoption framework. *Gyanshauryam, International Scientific Refereed Research Journal*, 6(6), 521–550. <https://doi.org/10.32628/GISRRJ236648>
- Eyetsemitan, R. A., Ambali, K. B., Oyeleye, A. O., & Fadayomi, O. (2023b). User acceptance testing in small business technology deployment: A structured validation framework for lean operational environments. *International Journal of Multidisciplinary Research and Growth Evaluation*, 4(6), 1512–1531. <https://doi.org/10.54660/IJMRGE.2023.4.6.1512-1531>
- Eyetsemitan, R. A., Ambali, K. B., Oyeleye, A. O., & Fadayomi, O. (2025). An integrated lean-digital framework for scaling small business operations: Synthesizing SOP design, automation, compliance, and change management. *International Journal of Multidisciplinary Research and Growth Evaluation*, 6(6), 1341–1360. <https://doi.org/10.54660/IJMRGE.2025.6.6.1341-1360>
- Fan, Z., Gao, X., Mirchev, M., Roychoudhury, A., & Tan, S. H. (2023). Automated repair of programs from large language models. *Proceedings of the International Conference on Software Engineering*, 1469–1481.
- Feng, Z., Guo, D., Tang, D., et al. (2020). CodeBERT: A pre-trained model for programming and natural languages. *Findings of the Association for Computational Linguistics: EMNLP 2020*, 1536–1547.
- Fobellah, A. N. (2025b). Navigating digital transformation: Auditing artificial intelligence-powered financial systems: A conceptual review. *International Journal of Science and Research Archive*, 16(2), 23–28. <https://doi.org/10.30574/ijrsra.2025.16.2.2274>
- Fowler, M. (2018). *Refactoring: Improving the design of existing code* (2nd ed.). Addison-Wesley.
- Guo, D., Ren, S., Lu, S., et al. (2021). GraphCodeBERT: Pre-training code representations with data flow. *Proceedings of the International Conference on Learning Representations*.
- Hendrycks, D., Basart, S., Kadavath, S., et al. (2021). Measuring coding challenge competence with APPS. *Proceedings of the NeurIPS Datasets and Benchmarks Track*.
- Hussain, N., & Adebayo, A. (2026). The role of artificial intelligence in strengthening modern cybersecurity systems. *World Journal of Innovation and Modern Technology*, 10(6), 125–181. <https://doi.org/10.56201/wjimt.v10.no6.2026.pg125.181>
- Idika, C. N., Salami, E. O., Ijiga, O. M., & Enyejo, L. A. (2021). Deep learning driven malware classification for cloud-native microservices in edge computing architectures. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 7(4).
- Jain, N., Han, K., Gu, A., et al. (2024). LiveCodeBench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.
- Jia, Y., & Harman, M. (2011). An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, 37(5), 649–678.
- Jimenez, C. E., Yang, J., Wettig, A., et al. (2024). SWE-bench: Can language models resolve real-world GitHub issues? *Proceedings of the International Conference on Learning Representations*.
- Joseph, O. B., & Uzundu, N. C. (2024a). Integrating AI and machine learning in STEM education: Challenges and opportunities. *Computer Science & IT Research Journal*, 5(8), 1732–1750. <https://doi.org/10.51594/csitrj.v5i8.1379>
- Just, R., Jalali, D., & Ernst, M. D. (2014). Defects4J: A database of existing faults to enable controlled testing studies for Java programs. *Proceedings of the International Symposium on Software Testing and Analysis*, 437–440.

- Kim, M., Zimmermann, T., & Nagappan, N. (2014). An empirical study of refactoring challenges and benefits at Microsoft. *IEEE Transactions on Software Engineering*, 40(7), 633–649.
- Komi, N. M., & Adamolekun, A. (2021). Interpretable machine learning for early failure prediction in distributed renewable energy assets. *International Journal of Multidisciplinary Research and Growth Evaluation*, 2(6), 1015–1038. <https://doi.org/10.54660/IJMRGE.2021.2.6.1015-1038>
- Kruchten, P., Nord, R. L., & Ozkaya, I. (2012). Technical debt: From metaphor to theory and practice. *IEEE Software*, 29(6), 18–21.
- Ladapo, O. O., Dosunmu, A. A., Jooda, D., & Abolaji, T. O. (2022). Human-in-the-loop machine learning: A state of the art. *Journal of Frontiers in Multidisciplinary Research*, 3(1), 656–669. <https://doi.org/10.54660/JFMR.2022.3.1.656-669>
- Ladapo, O. O., Jooda, D., Dosunmu, A. A., & Abolaji, T. O. (2024). Keeping humans in the loop: Human-centered automated annotation with generative AI. *International Journal of Multidisciplinary Futuristic Development*, 5(1), 81–95. <https://doi.org/10.54660/IJMFD.2024.5.1.81-95>
- Ladapo, O. O., Jooda, D., Dosunmu, A. A., & Abolaji, T. O. (2026). On the disagreement problem in human-in-the-loop federated machine learning. *International Journal of Multidisciplinary Research and Growth Evaluation*, 7(3), 178–192. <https://doi.org/10.54660/IJMRGE.2026.7.3.178-192>
- Lanham, T., Chen, A., Radhakrishnan, A., et al. (2023). Measuring faithfulness in chain-of-thought reasoning. *arXiv preprint arXiv:2307.13702*.
- Lehman, M. M. (1980). Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9), 1060–1076.
- Li, Y., Choi, D., Chung, J., et al. (2022). Competition-level code generation with AlphaCode. *Science*, 378(6624), 1092–1097.
- Liu, J., Xia, C. S., Wang, Y., & Zhang, L. (2023). Is your code generated by ChatGPT really correct? Rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*, 36, 21558–21572.
- Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., & Liang, P. (2024). Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12, 157–173.
- Lu, S., Guo, D., Ren, S., et al. (2021). CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. *Proceedings of the NeurIPS Datasets and Benchmarks Track*.
- Lu, Y., Bartolo, M., Moore, A., Riedel, S., & Stenetorp, P. (2022). Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 8086–8098.
- Madaan, A., Tandon, N., Gupta, P., et al. (2023). Self-Refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 46534–46594.
- Mayo, W., Ogbale, J. I., Okoruwa, P. O., & Babatope, O. M. (2021). Designing an AI-predictive maintenance model for e-commerce systems using machine learning and cloud analytics. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 7(5), 416–440.
- Mbonu, I. S., Aliliele, C., Iwuanyanwu, U., & Uzoka, E. (2021b). Advances in artificial intelligence techniques for secure software testing and automated regression control mechanisms. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 7(5), 468–496. <https://doi.org/10.32628/CSEIT217565>
- Mbonu, I. S., Aliliele, C., Iwuanyanwu, U., & Uzoka, E. (2022). A conceptual framework for AI enabled IT general controls and SOX audit automation processes. *Gyanshauryam, International Scientific Refereed Research Journal*, 5(5), 384–414. <https://doi.org/10.32628/GISRRJ2256239>
- Mbonu, I. S., Iwuanyanwu, U., Aliliele, C., & Uzoka, E. (2020b). Advances in infrastructure as code governance for secure terraform based enterprise cloud deployments. *International Journal of Multidisciplinary Research and Growth Evaluation*, 1(5), 811–828. <https://doi.org/10.54660/IJMRGE.2020.1.5.811-828>
- Mbonu, I. S., Iwuanyanwu, U., Uzoka, E., & Oluoha, O. M. (2019). Advances in enterprise log analytics and automated incident response architectures using Python and SIEM platforms. *Iconic Research and Engineering Journals*, 3(2), 1000–1019. <https://doi.org/10.64388/IREV3I2-1714915>
- Mens, T., & Tourwé, T. (2004). A survey of software refactoring. *IEEE Transactions on Software Engineering*, 30(2), 126–139.

- Murphy-Hill, E., Parnin, C., & Black, A. P. (2012). How we refactor, and how we know it. *IEEE Transactions on Software Engineering*, 38(1), 5–18.
- Nijkamp, E., Pang, B., Hayashi, H., et al. (2023). CodeGen: An open large language model for code with multi-turn program synthesis. *Proceedings of the International Conference on Learning Representations*.
- Nwakamma, S., Ojukwu, J., & Oyesiji, S. O. (2024a). LLM-as-a-judge for automated model governance: A review of methods, biases, and release-gating practices. *World Journal of Innovation and Modern Technology*, 8(6), 185–216. <https://doi.org/10.56201/wjimt.v8.no6.2024.pg185.216>
- Nwakamma, S., Ojukwu, J., & Oyesiji, S. O. (2024b). Securing agentic AI enterprise workflows against prompt injection, tool poisoning, memory manipulation, and excessive agency threats. *World Journal of Innovation and Modern Technology*, 8(6), 217–248. <https://doi.org/10.56201/wjimt.v8.no6.2024.pg217.248>
- Nwakamma, S., Ojukwu, J., & Oyesiji, S. O. (2025). On-device multimodal small language models for privacy-preserving edge intelligence through quantization, pruning, distillation, and energy-efficient inference. *World Journal of Innovation and Modern Technology*, 9(12), 271–302. <https://doi.org/10.56201/wjimt.v9.no12.2025.pg271.302>
- Odejide, G. S. (2023). Artificial intelligence in multilingual and accessible emergency alerting: A scoping review of evidence, applications, and gaps in the United States. *International Journal of Scientific Research in Science, Engineering and Technology*, 10(6), 499–512. <https://doi.org/10.32628/IJSRSET26132152>
- Odejide, G. S. (2024). A trust-centered conceptual model for AI-driven multilingual emergency communication. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 10(6), 2781–2811. <https://doi.org/10.32628/CSEIT2410794>
- Odejobi, O. D., Okonkwo, C. S., Ahiaekwe Patrick, M. C., Okeke, O. T., & Mayo, W. (2025). AI-augmented secure software engineering: Leveraging deep learning for autonomous threat detection and mitigation. *International Journal of Engineering and Modern Technology*, 11(12), 101–121. <https://doi.org/10.56201/ijemt.vol.11.no12.2025.pg101.121>
- Ogundairo, K. M. (2025). Hardening the autonomous value chain: Lean Six Sigma guardrails and multi-enterprise ETL architecture for agentic supply chain orchestration. *IIARD International Journal of Economics and Business Management*, 11(12), 441–478. <https://doi.org/10.56201/ijebm.vol.11.no12.2025.pg441.478>
- Ogundairo, K. M., & Ayivi-Donkor, S. S. (2024a). Bridging the adoption gap: Why supply chain AI fails without product marketing principles. *International Journal of Marketing and Communication Studies*, 8(5), 169–205. <https://doi.org/10.56201/ijmcs.v8.no5.2024.pg169.205>
- Ogundairo, K. M., & Ayivi-Donkor, S. S. (2024b). The vanishing interface: Why the coming era of autonomous agents demands a paradigm shift in product marketing architecture. *IIARD International Journal of Economics and Business Management*, 10(11), 350–385. <https://doi.org/10.56201/ijebm.v10.no11.2024.pg350.385>
- Ogunwola, T. A. (2026). Security and resilience considerations for software-defined wide area network deployments in multi-site enterprise environments. *International Advanced Research Journal in Science, Engineering and Technology*, 12(1). <https://doi.org/10.17148/iarjset.2025.12150>
- Ojukwu, J., Oyesiji, S. O., & Nwakamma, S. (2025). WebAssembly component model and WASI for portable cloud-to-edge software: Language interoperability, security, performance, and container comparisons. *International Journal of Computer Science and Mathematical Theory*, 11(12), 232–263. <https://doi.org/10.56201/ijcsmt.vol.11.no12.2025.pg232.263>
- Okonkwo, C. S., Agbabiaka, J., Mayo, W., & Okeke, O. T. (2024f). Supply chain automation framework using service management platforms. *Shodhshauryam, International Scientific Refereed Research Journal*, 7(2), 157–177. <https://doi.org/10.32628/SHISRRJ2472157>
- Olausson, T. X., Inala, J. P., Wang, C., Gao, J., & Solar-Lezama, A. (2024). Is self-repair a silver bullet for code generation? *Proceedings of the International Conference on Learning Representations*.
- Ominyi, M., Anichukwueze, C. C., & Uzougbo, N. S. (2024b). A systematic review of organizational readiness for the EU Artificial Intelligence Act in multinational enterprises. *World Journal of Innovation and Modern Technology*, 8(6), 185–203. <https://doi.org/10.56201/wjimt.v8.no6.2024.pg185.203>
- Ominyi, M., Anichukwueze, C. C., & Uzougbo, N. S. (2025c). Reviewing legal privilege and confidentiality challenges in enterprise deployment of large language models. *Journal of Law and Global Policy*, 10(3), 150–176. <https://doi.org/10.56201/jlgp.vol.10.no3.2025.pg150.176>

- Onwubuya, L. I., Fawehinmi, Y. O., Sunday, D., Igwenagu, M. O., Adeniran, A. O., & Agyapong, E. A. (2026). Adaptive learning systems powered by artificial intelligence for STEM education improvement. *Asian Journal of Education and Social Studies*, 52(6), 553–566. <https://doi.org/10.9734/ajess/2026/v52i63115>
- Opdyke, W. F. (1992). Refactoring object-oriented frameworks [Doctoral dissertation, University of Illinois at Urbana-Champaign].
- OpenAI. (2023). GPT-4 technical report. arXiv preprint arXiv:2303.08774.
- Ouyang, L., Wu, J., Jiang, X., et al. (2022). Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35, 27730–27744.
- Oyesiji, S. O., Nwakamma, S., & Ojukwu, J. (2023). Parameter-efficient multilingual adaptation of on-device language models: A survey. *International Journal of Engineering and Modern Technology*, 9(3), 287–320. <https://doi.org/10.56201/ijcsmt.v9.no3.2023.pg287.320>
- Oyesiji, S. O., Nwakamma, S., & Ojukwu, J. (2024). A conceptual framework for context-aware data curation in fine-tuning pipelines. *International Journal of Engineering and Modern Technology*, 10(11), 197–228. <https://doi.org/10.56201/ijemt.v10.no11.2024.pg197.228>
- Oyesiji, S. O., Nwakamma, S., & Ojukwu, J. (2025). Compressing recommender and ranking models for edge deployment: A survey. *International Journal of Engineering and Modern Technology*, 11(12), 205–237. <https://doi.org/10.56201/ijemt.vol.11.no12.2025.pg205.237>
- Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12), 1053–1058.
- Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., & Karri, R. (2022). Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions. *Proceedings of the IEEE Symposium on Security and Privacy*, 754–768.
- Perry, N., Srivastava, M., Kumar, D., & Boneh, D. (2023). Do users write more insecure code with AI assistants? *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2785–2799.
- Quainoo, R., Ogundapo, O., & Asiedu, W. A. (2025b). Test automation in wireless hardware engineering: A comprehensive review of scripting frameworks and instrument control strategies. *International Journal of Engineering and Modern Technology*, 11(10), 405–464. <https://doi.org/10.56201/ijemt.vol.11.no10.2025.pg405.464>
- Rozière, B., Gehring, J., Gloeckle, F., et al. (2023). Code Llama: Open foundation models for code. arXiv preprint arXiv:2308.12950.
- Sanni, J. O., Iwuanyanwu, U. A., & Essien, M. A. (2025). Problem-oriented process mining for auditable marketing automation lifecycle control. *International Journal of Advanced Multidisciplinary Research and Studies*, 5(6), 1933–1947. <https://doi.org/10.62225/2583049X.2025.5.6.5650>
- Sanni, J. O., Iwuanyanwu, U. A., & Essien, M. A. (2026). Designing explainable AI based marketing automation architectures for healthcare and financial applications. *World Scientific News*, 213, 119–148. <https://doi.org/10.65770/MJFK8593>
- Sclar, M., Choi, Y., Tsvetkov, Y., & Suhr, A. (2024). Quantifying language models' sensitivity to spurious features in prompt design. *Proceedings of the International Conference on Learning Representations*.
- Seun, A., Ajayi, A., Yeboah, T. J., Ofori, S. D., Elumilade, R. A., Apelehin, A. A., Abutu, D. E., & Bobga, M. A. (2024). The PearlAI framework: Leveraging artificial intelligence to transform education in Africa. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 10(3), 1120–1132. <https://doi.org/10.32628/CSEIT25113581>
- Shah, R., Oziri, S. T., & Seyi-Lande, O. B. (2025). A framework for leveraging artificial intelligence in strategic business decision-making. *Gulf Journal of Advance Business Research*, 3(11), 1517–1558.
- Sharma, M., Tong, M., Korbak, T., et al. (2024). Towards understanding sycophancy in language models. *Proceedings of the International Conference on Learning Representations*.
- Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., & Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 8634–8652.
- Sobania, D., Briesch, M., Hanna, C., & Petke, J. (2023). An analysis of the automatic bug fixing performance of ChatGPT. *Proceedings of the IEEE/ACM International Workshop on Automated Program Repair*, 23–30.
-

- Tsantalis, N., Mansouri, M., Eshkevari, L. M., Mazinanian, D., & Dig, D. (2018). Accurate and efficient refactoring detection in commit history. *Proceedings of the International Conference on Software Engineering*, 483–494.
- Turpin, M., Michael, J., Perez, E., & Bowman, S. R. (2023). Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems*, 36, 74952–74965.
- Vaithilingam, P., Zhang, T., & Glassman, E. L. (2022). Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, 1–7.
- Wang, Y., Wang, W., Joty, S., & Hoi, S. C. H. (2021). CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 8696–8708.
- Wedraogo, L., & Sanni, J. O. (2024). Machine learning models addressing uncertainty in cross-channel campaign performance forecasting accuracy. *International Journal of Advanced Multidisciplinary Research and Studies*, 4(6), 2875–2890. <https://doi.org/10.62225/2583049X.2024.4.6.5648>
- Xia, C. S., & Zhang, L. (2022). Less training, more repairing please: Revisiting automated program repair via zero-shot learning. *Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 959–971.
- Yoo, S., & Harman, M. (2012). Regression testing minimization, selection and prioritization: A survey. *Software Testing, Verification and Reliability*, 22(2), 67–120.
- Zheng, L., Chiang, W.-L., Sheng, Y., et al. (2023). Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. *Advances in Neural Information Processing Systems*, 36, 46595–46623.