Automated Generation of Exploratory Test Cases Using Prompt Chaining and Reflective Evaluation

Igor Lima
Universidade Federal do Amazonas
Manaus, Amazonas, BR
igor.lima@icomp.ufam.edu.br

André Carvalho
Universidade Federal do Amazonas
Manaus, Amazonas, BR
andre@icomp.ufam.edu.br

Yan Soares
Universidade Federal do Amazonas
Manaus, Amazonas, BR
yan.soares@icomp.ufam.edu.br

Gabriel Pacheco
Universidade Federal do Amazonas
Manaus, Amazonas, BR
gabriel.pacheco@icomp.ufam.edu.br

Andrés Peralta
Universidade Federal do Amazonas
Manaus, Amazonas, BR
andres@icomp.ufam.edu.br

Hallyson Melo
INDT
Manaus, Amazonas, BR
hallyson.melo@indt.org.br

Nikson Ferreira
INDT
Manaus, Amazonas, BR
nikson.ferreira@indt.org.br

Bruno Costa
INDT
Manaus, Amazonas, BR
bruno.costa@indt.org.br

ABSTRACT

Exploratory testing uncovers latent defects, but the design of Exploratory Test Cases (ETC) is largely manual, incurring high costs, reliance on expert knowledge, and limited reproducibility. We propose **IGEX**, a fully automated approach to generate ETCs using Large Language Models. **IGEX** models test generation as a structured chain of prompts, leveraging Chain of Thought reasoning and learning in context. To ensure quality, a Reflective Evaluator scores ETCs according to expert criteria, triggering refinements as needed. In line with Whittaker's hybrid ETC definition, our method combines structured scripting with tester-driven exploratory detours, enabling automation. In experiments with 300 Android test scenarios, **IGEX** achieved 97.67% accuracy before reflection and 100% after iterative evaluation. Although dataset-agnostic and extensible, current validation is limited to mobile applications. Furthermore, **IGEX** inherits the challenges of LLM, including evaluation subjectivity and computational costs. These results demonstrate the potential of LLM for scalable exploratory testing with reduced manual overhead.

KEYWORDS

Exploratory Test Cases, Large Language Models, Prompt Chaining, Automated Test Generation, Test Automation, Reflective Evaluation Mechanism.

1 Introduction

Ensuring software reliability requires rigorous testing. Industry reports indicate that up to 50% of software development resources are dedicated to testing and validation activities [11–13], making test design a central concern in modern development pipelines. While traditional testing often relies on scripted test cases [1], defined sequences of actions and expected outcomes [7], this rigidity can overlook unexpected behaviors and real-world usage patterns [9, 14, 25].

Exploratory testing addresses these limitations by allowing testers to interact dynamically with the software, discovering edge cases and defects through investigation rather than predefined steps. It emphasizes simultaneous learning, test design, and execution [6], and is particularly valuable in identifying failures that scripted testing may miss [5, 14]. However, exploratory testing remains a predominantly manual, labor-intensive process [16], requiring substantial tester expertise and offering limited repeatability [8].

The creation of Exploratory Test Cases (ETCs) often lacks structure, is time-consuming, and is difficult to scale, particularly in the context of increasingly complex systems and continuous development pipelines [19]. Manual ETC formulation also tends to result in low traceability, complicating documentation and reducing the reliability of regression efforts.

To address these limitations and augment the application of exploratory testing, automation of test generation, not test execution, emerges as a promising avenue. By automating the generation of ETCs from general scenario descriptions, it becomes possible to systematically and consistently incorporate exploratory testing into broader validation workflows, even in contexts where experienced testers may be scarce.

In this paper, we present **IGEX**, a novel method that leverages Large Language Models (LLMs) [28] to automatically generate ETCs from high-level test descriptions. Unlike previous approaches that depend on extensive example data or domain-specific knowledge structures, IGEX operates through a structured *prompt chaining* mechanism that decomposes the generation process into manageable, logically connected steps. The method integrates Chain of Thought prompting [24] and In-Context Learning (ICL) [26] to guide the model's reasoning and outputs, progressively refining general prompts into detailed, structured exploratory test scripts.

A distinguishing feature of IGEX is the inclusion of a **Reflective Evaluator**, which assesses the generated ETCs against expert-defined exploratory criteria. If the test case does not meet these criteria, the evaluator triggers iterative refinement cycles, progressively adjusting the prompts until a valid ETC is produced. This mechanism ensures not only the syntactic coherence but also the semantic alignment of the generated test cases with established exploratory testing principles.

The goal of this study is twofold: *(1)* to demonstrate that LLMs can autonomously generate structured and diverse ETCs from high-level descriptions, and *(2)* to evaluate how prompt chaining and reflective evaluation contribute to the effectiveness and quality of generated ETCs.

Our contributions are: ***1.** A modular architecture for LLM-driven ETC generation using structured prompt chains. **2.** A reflective evaluation mechanism based on expert-informed criteria to validate test case quality. **3.** item An empirical study on 300 Android exploratory testing scenarios, including an ablation analysis of **IGEX** components. **4.** Practical insights on the reproducibility, adaptability, and quality assurance of LLM-generated ETCs.*

2 Background and Related Work

This section aims to provide an understanding of the fundamental concept and crucial context for this study. It delves into the definition and concepts of ETCs in Section 2.1. We also present previous work related to automated generation of ETCs and positioning of **IGEX** in the literature in Sections 2.2 and 2.3, respectively.

2.1 Exploratory Testing Concepts and Definitions

Exploratory Test Cases are a central concept in testing strategies that prioritize adaptability and learning during the testing process. However, their definition in the literature is broad and sometimes ambiguous [15]. James Bach [6] describes exploratory testing as *"simultaneous learning, test design, and execution,"* emphasizing its informal and intuitive nature. Tinkham and Kaner [23] expand on this by defining ETCs as tests in which the tester *"actively controls the process, designing new tests while executing existing ones and using information obtained during the process to improve them."*

Jonathan Bach [5] views ETC primarily as a strategy for defect detection, noting that the goal is to adjust the testing process in real time to focus on areas of high risk and uncertainty. This implies a highly dynamic and heuristic-driven testing process. Fredrik Asplund [4] also characterizes ETCs as test artifacts deeply intertwined with human judgment, tester expertise and domain familiarity, usually non-repeatable, and rarely formally documented.

In contrast, Whittaker [25] proposes a more structured interpretation, suggesting that exploratory testing case occurs when formal scripts are used as a flexible baseline. In this view, testers begin with a predefined goal but are encouraged to deviate based on system feedback, exploring alternative paths and unexpected scenarios. Whittaker argues that scripts can support exploration by providing a frame of reference, while exploratory behavior introduces variation and depth. This hybrid perspective allows for both repeatability and adaptability, an ideal balance for automated approaches using LLMs.

Igor Ernesto [8] adopts an approach similar to Whittaker's, defining exploratory testing as a process in which learning, design, and execution occur simultaneously. However, to max- imize its effectiveness, the author suggests that following structured guidelines can facilitate the success of the testing process. According to Ernesto, the effectiveness of exploratory testing largely depends on the knowledge and experience of the software engineer, and the adoption of structured activities can help optimize this process, ensuring a balance between flexibility and direction.

For the purposes of this study, we adopt Whittaker's model as the guiding definition. It enables the combination of scripted scaffolding with creative exploration, aligning well with IGEX's use of prompt-driven generation and reflective refinement. Moreover, it allows the test case to remain traceable and structured, while promoting the tester's (or model's) autonomy to explore dynamically.

2.2 Automated Exploratory Test Case Generation

Despite its importance, the automation of ETC generation remains underexplored due to the inherent subjectivity of the process [2]. Traditional approaches, such as the statistical randomization by Sternerson and Firoozfam [21], and the Petri net modeling in MISTA [27], attempt to structure exploratory testing through formal modeling. These techniques can be effective in safety-critical or deterministic environments, but require significant domain-specific effort and lack flexibility.

In 2016, Makondo et al. [17] introduced the use of multilayer perceptron neural networks to predict software behavior and act as oracles for exploratory tests. However, this approach is limited by the generalizability of the trained model and the need for large-scale labeled data. Later, Nishi and Shibasaki [18] leveraged Word2Vec embeddings to suggest semantically diverse test scenarios, encouraging exploration based on linguistic variation. While innovative, this method is highly dependent on the training corpus quality.

Recent advances in natural language processing have inspired the use of Large Language Models for script test generation. Sami et al. [20] proposed a tool that transforms natural language requirements into test scripts. While not focused on ETCs, their work shows that LLMs can effectively translate informal specifications into structured outputs, reducing manual effort.

The first study to utilize LLMs for generating ETCs was published in April 2024. Su et al. [22] introduced an ap- proach that combines Large Language Models with Knowl- edge Graphs to enhance exploratory testing. It represents a significant advancement in the automation of exploratory testing by employing Large Language Models to construct a system knowledge graph. This graph is derived from failure reports and aims to capture usage patterns and common errors to support the automated generation of ex- ploratory test scenarios.

(1) Fragile report interpretation: The approach relies on segmenting failure reports into structured elements (e.g., Steps to Reproduce, Expected and Observed Behaviors), but struggles with complex sentences and contextual dependencies. Errors during this process can distort event sequences and compromise test scenario validity.

(2) Dependence on example data: The effectiveness of the method is heavily tied to the availability of well-structured and diverse historical failure reports. In domains with sparse or poorly documented failures, the approach lacks the contextual grounding necessary for effective test generation.

(3) Limited generative flexibility: By focusing on recombining pre-existing report segments, the method is constrained in its ability to generate novel or diverse exploratory scenarios, particularly in systems with complex or emergent behaviors.

2.3 Positioning IGEX

IGEX builds on these insights but eliminates the dependency on structured datasets or prior reports. Instead, it uses a modular prompt chain to guide LLM reasoning from scratch, integrating intermediate scaffolding (scripted planning, deviation generation, risk analysis) with iterative feedback via a Reflective Evaluator. Unlike Su et al.'s approach, **IGEX** is data-agnostic and capable of producing ETCs from minimal initial input, enabling greater flexibility and domain independence.

Moreover, by adopting Whittaker's hybrid model and operationalizing it into prompt-based reasoning criteria, **IGEX** ensures that generated ETCs are not only syntactically coherent but semantically aligned with exploratory testing principles. This balance of structure and creativity is what differentiates **IGEX** in the landscape of automated test generation.

Table 1 summarizes the main differences between **IGEX** and previous approaches.

3 Methodology

Although Su et al. [22] introduced LLMs for ETC generation, their approach relies on structured data and has limited scenario diversity. To address these issues, we propose **IGEX**, a fully LLM-based method that requires no preprocessing or knowledge graphs. **IGEX** uses prompt chaining, chain-of-thought, and iterative reflection to generate ETCs from general descriptions, following six steps: *(1) Scenario Refinement, (2) Initial Script Generation, (3) User Action Generation, (4) Critical Point Identification, (5) ETC Integration, and (6) Reflective Evaluation.* Outputs failing quality criteria are refined iteratively. Sections *3.1* and *3.2* detail the prompt structure and evaluation. Full prompt and questionnaire examples are omitted due to space but available upon request.

3.1 Structured Prompt Chain

The process begins with the refinement of the initial scenario. Given a general intent such as "Validate the Start button in the main menu", the model is prompted to reinterpret the description by introducing specific details, such as component hierarchies, expected behaviors, and relevant context. In this step, the model is guided using In-Context Learning, where examples of scenario refinement are embedded directly in the prompt to anchor the model's reasoning and reduce ambiguity [10]. An example of such a prompt is: *"Here is a general description of a scenario. Create a more specific scenario that includes interaction constraints and expected behavior."* By providing concrete examples in the same prompt, ICL allows the model to infer patterns and expectations even without fine-tuning.

Once the refined scenario is obtained, **IGEX** starts generating a structured test script. This phase builds a test case step by step, asking the model to simulate the tester's mental model in pursuit of the test objective. Again, ICL is applied, embedding a formatted exemplar test case directly into the prompt, ensuring that the output follows the desired model. Additionally, the Chain of Thought technique is employed to enforce structured reasoning: the model explains each test step before presenting it, improving traceability and internal consistency. For example, the prompt might instruct: *"Write an Android test case for the refined objective. Start by explaining the logic behind each test step, and then format the output using the standard test case template."*

The third stage involves the creation of exploratory user actions. The model is instructed to inject deviations and boundary conditions that simulate real user exploration. The prompt provides specific rules and, through ICL, includes curated examples of creative exploratory actions to guide the model in producing varied and contextually relevant deviations. For instance, the prompt may state: *"List atomic actions that could trigger unexpected behavior during test execution. Each should be unique, context-relevant, and framed as a single user action (e.g., 'long-press the back button'). Avoid generic actions."*

Using the output of the second prompt as input, **IGEX** performs the critical point identification step, the objective of which is to locate areas within the test flow that are most likely to fail, considering factors such as high complexity, ambiguous interactions or specific conditions that may not be adequately met. This step is essential to enhance the exploratory testing experience, allowing the model to anticipate potential system vulnerabilities and direct the generation of ETCs in a more strategic and effective manner. By identifying these areas of greatest risk, **IGEX** enables the exploration of plans that are most susceptible to failure, ensuring that the generated tests not only follow a structured script, but also cover the most sensitive points of the application. This analysis is performed using a prompt such as: *"From the generated test case, identify the step that is most susceptible to errors or failures. Justify your selection based on the complexity of the interface or feedback mechanisms."*

With all the parts generated, the structured test case, the exploratory variations, and the identification of critical points, **IGEX** moves on to the integration phase, where these components are consolidated into a complete and coherent ETC. In this step, the final integration prompt guides the model to combine the different outputs, ensuring cohesion and alignment with the established format. To reinforce the consistency of the result, the prompt defines a template to be followed: *"Combine the structured Android test case, the user variations, and the identified critical step into a single ETC. Format using the defined schema."*

3.2 Reflexive Evaluation

The final stage of **IGEX** is the Reflexive Evaluation, which ensures that the generated ETC aligns with established exploratory testing principles. This component is fully automated and implemented through a structured prompt that instructs the LLM to classify the test case as "Exploratory" or "Non-Exploratory," providing a detailed justification for its decision. To address previous limitations of similar approaches, the Reflexive Evaluator in **IGEX** was designed as a transparent mechanism, applying a structured checklist of criteria derived from empirical research and expert practice.

These criteria were consolidated through a questionnaire developed specifically for this study, which was grounded in both an extensive review of the literature and the practical experience of testers. The questionnaire was applied to three senior software testers, comprising eleven open-ended questions that explored essential aspects of ETCs. These included key characteristics that

Table 1: Comparison of Automated Exploratory Test Generation Approaches

Approach	Automation Scope	Methodology	Structured Data Needed	Flexibility	Iterative Refinement
Sternerson & Firoozfam [21]	ETC	Statistical Randomization	No	Low	No
Xu (MISTA) [27]	ETC	Petri Net Modeling	Yes	Low	No
Makondo et al. [17]	Test Oracle	Neural Network Prediction	Yes	Medium	No
Nishi & Shibasaki [18]	Test Suggestion	Semantic Embeddings	Yes	Medium	No
Sami et al. [20]	General Test Cases	LLM Generation	No	Medium	No
Su et al. [22]	ETC	KG + LLM Recombination	Yes	Low	No
IGEX (ours)	ETC	Generative Prompt Chaining	No	High	Yes

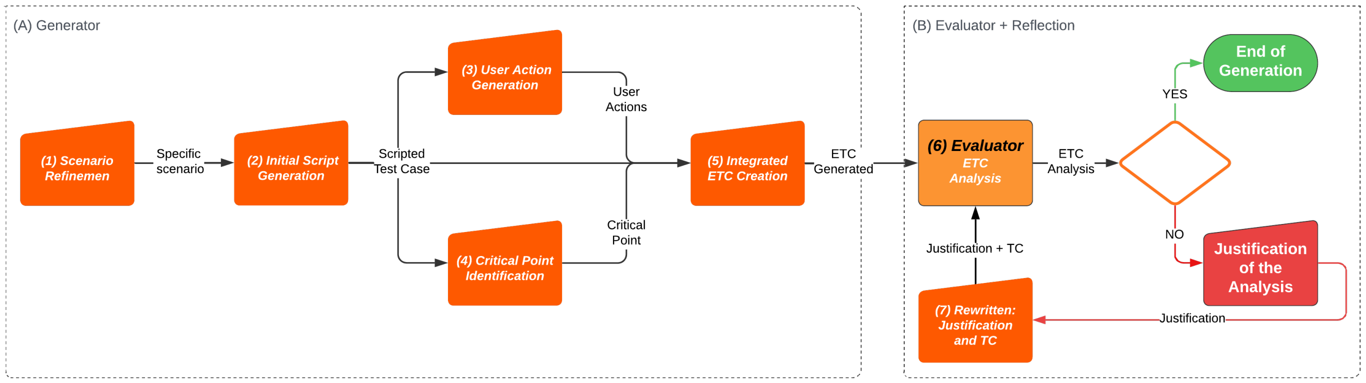
(A) Generator
(1) Scenario Refinemen
Specific scenario
(2) Initial Script Generation
Scripted Test Case
(3) User Action Generation
User Actions
(4) Critical Point Identification
Critical Point
(5) Integrated ETC Creation
ETC Generated
(B) Evaluator + Reflection
(6) Evaluator ETC Analysis
ETC Analysis
YES
End of Generation
NO
Justification of the Analysis
Justification
(7) Rewritten: Justification and TC
Justification + TC

Figure 1: Complete Chain Structure. Each block is a prompt input to the model, each output is forwarded to create a new prompt. Although Steps 3 and 4 appear parallel, they are executed sequentially, as described in Section 3.1

define ETCs, common formulation errors, and best practices for structuring effective cases. An example of such a question is: *"Besides being exploratory, what are the characteristics that an ETC should have?"* The structure and content of the questionnaire were strongly influenced by the guidelines proposed by Costa et al. [8], which investigated the features that define an ETC and how it can be categorized as effective. However, unlike Costa et al.'s work, which provided valuable theoretical guidelines but lacked direct empirical data from testers, our approach aimed to consolidate evaluation criteria through firsthand insights from experienced practitioners.

From the qualitative analysis of the testers' responses, we extracted a set of key attributes that form the basis of the Reflective Evaluator. These include the presence of purposeful deviations or unscripted steps, the tester's autonomy to adapt actions based on system feedback, the encouragement of exploring alternative paths within or beyond the primary scenario, and the prioritization of critical areas that maximize defect detection and coverage. The evaluator simulates an expert assessment by prompting the model with reflective questions framed explicitly around these attributes. For example, the model is instructed to analyze whether the test case includes deviations from the standard flow, whether it invites the tester to react dynamically to the system's behavior, and whether the test steps demonstrate flexibility and judgment rather than rigid adherence to a pre-defined sequence.

To ensure consistency and minimize ambiguity in the classification, the prompt applies the In-Context Learning technique, incorporating carefully selected examples of exploratory and non-exploratory test cases. This approach allows the LLM to compare the generated ETC with established standards of testing practices, aligning its reasoning with the quality expectations established by the experts. If the evaluator classifies the test case as non-exploratory, **IGEX** automatically restarts the chain of prompts, starting from Prompt 7, and integrating the evaluation feedback into the refinement process. This review cycle is iterative and can be repeated as many times as necessary until an ETC is positively evaluated.

Throughout this process, no human intervention occurs after the initial definition of the criteria and prompt templates, ensuring a fully automated, scalable, and replicable evaluation cycle. While the evaluator is grounded in expert-informed guidelines, we recognize that it inherits limitations from the model's subjectivity and the dependency on the quality of the prompts and examples provided.

3.3 Experimental Setup

To ensure consistency and representativeness, we used a dataset of 300 general test scenarios extracted from an industrial Android Charter. These scenarios reflect common mobile interface validation goals and were provided by a partner company under confidentiality agreement. All tests were conducted using the same dataset across configurations, ensuring comparability.

ETCs were generated using the LLaMA 3.1 8B INSTRUCT model [3], selected for its strong performance, open availability, and ability to run securely in local environments, an essential requirement given the confidentiality of the test data. This was necessary to preserve data privacy, making cloud-based models unfeasible. The generation process employed fixed parameters (temperature = 0.4–0.6, top-p = 0.85), without fine-tuning.

Following the approach of Su et al. [22], we define *accuracy* as the percentage of input scenarios that result in ETCs classified as "exploratory" by the Reflective Evaluator. Unlike binary pass/fail assessments, this metric captures alignment with exploratory testing principles as defined by our expert-informed criteria. Thus, a test is only considered accurate if it fully meets the this standard.

4 Results

This section presents the empirical results obtained from evaluating **IGEX** over a curated dataset of real-world test scenarios. The experiments aim to assess the effectiveness of **IGEX** in generating valid ETCs and to measure the individual contribution of each component in the methodology. The evaluation includes an ablation study and an analysis of the iterative behavior of the Reflective Evaluator.

4.1 Performance and Ablation Results

The complete **IGEX**, prior to reflective refinement, achieved 97.67% accuracy. This demonstrates that the prompt chaining architecture, combined with Chain of Thought and In-Context Learning, is already capable of generating highly compliant ETCs.

To understand the impact of each **IGEX** component, we performed a controlled ablation study with seven configurations:

(1) **Complete algorithm without reflection:** evaluates the impact of omitting reflective validation.
(2) **Single Prompt Generation:** tests whether a monolithic prompt can substitute the prompt chain.
(3) **No Scenario Refinement:** skips the detailing of the initial user input.
(4) **No Scripted Test Generation:** removes intermediate scaffolding before user actions.
(5) **No User Action Generation:** excludes deviations and exploratory inputs.
(6) **No Critical Point Identification:** omits risk analysis of vulnerable interface points.
(7) **No In-Context Learning and Chain of Thought:** retains the prompt chain but disables prompting strategies.

These variations aim to isolate the role of each element and to evaluate how each contributes to the structural, contextual, and exploratory richness of the generated ETCs. The results allow us to understand not only which components improve accuracy, but also what characteristics are lost when they are absent.

The results are shown in Figure 2. The most significant degradation occurred when the prompt chain was removed and replaced with a single generation prompt, reducing accuracy to 25.67%. This indicates that modular structuring is critical for maintaining coherence and exploratory depth.

Eliminating the Scripted Test Generation stage also reduced performance substantially (79.33%), highlighting the role of intermediate structure in organizing test logic. Similarly, omitting the User Action Generation and Critical Point Identification stages led to accuracies of 76.33% and 57.67%, respectively, demonstrating that these components are essential for achieving diversity and strategic depth in ETCs.

Interestingly, removing Scenario Refinement had minimal impact (96.00%), likely due to the model's ability to infer contextual detail in subsequent prompts. Finally, when both ICL and Chain of Thought were excluded, accuracy dropped to 65.00%, indicating that advanced reasoning techniques substantially improve test consistency and logical flow.

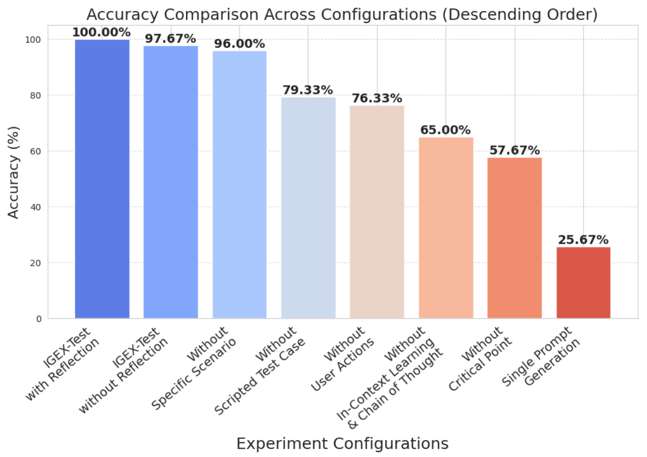

Figure 2: Accuracy of IGEX across different ablation configurations.

The obtained results highlight that each component of the approach significantly contributes to the generation of ETCs, with only scenario specification having a small contribution overall. The degradation in performance across different con- figurations reinforces the essential role of a structured and iterative process in exploratory test generation. The main conclusions drawn are: *(1)* The Prompt Chain is indispens- able, since without it, the generated tests are superficial and inconsistent. *(2)* The Scripted Test Case is essential, since It acts as an organizing foundation, preventing the generation of disconnected tests. *(3)* Critical Point Identification and User Actions are crucial, since they ensure diversity and depth in exploration. *(4)* Advanced Techniques Improve Quality, since Chain of Thought and In-Context Learning enhance the coherence and accuracy of ETCs.

The findings reinforce that the proposed approach maxi- mizes the effectiveness of automated ETC generation, making it a robust alternative for exploratory test automation. The integration of advanced prompt engineering techniques enables a scalable and efficient model, with a high capacity to adapt to different software validation contexts.

4.2 Reflective Iterations Analysis

We also analyzed the impact of the reflective feedback loop on test quality. Although the final **IGEX** configuration achieved 100% accuracy after iterative refinement, six ETCs initially failed to meet the exploratory criteria. The Reflective Evaluator triggered regeneration cycles until each test passed, with most corrections occurring within four iterations. However, two specific cases required up to seven and eight iterations, respectively.

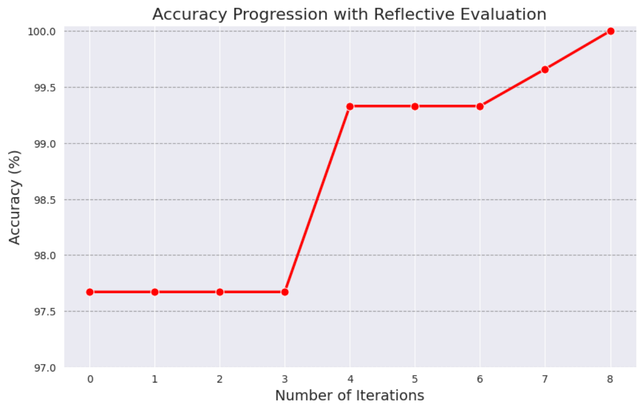

Figure 3: Convergence of ETCs under reflective iteration.

This pattern indicates that while many failures can be resolved with minor adjustments, others require deeper reformulation. Although the iterative process ensures high quality, it introduces computational cost and variability in convergence speed. A detailed iteration curve is presented in Figure 3, illustrating the gradual increase in accuracy over successive cycles.

5 Discussion

The results highlight the robustness of **IGEX** in generating valid ETCs from high-level descriptions. Beyond performance metrics, these outcomes inform broader considerations about methodology, the role of LLMs, and directions for future automation.

A key strength of **IGEX** lies not only in leveraging LLM capabilities, but in its structured architectural design. The modular prompt chain, combined with techniques such as In-Context Learning and Chain of Thought prompting, enables the model to simulate reasoning processes similar to those of human testers, transforming it from a general-purpose generator into a task-specific agent. Although the LLaMA 3.1 model lacks explicit training in software testing, its general-purpose corpus captures sufficient latent knowledge to approximate tester reasoning when properly guided. Nonetheless, this competence remains functional rather than conceptual.

The Reflective Evaluator is central to embedding exploratory testing principles into the generation loop, providing a mechanism to assess and iteratively refine ETCs based on explicit criteria. However, this process simulates expert reasoning and does not replace human judgment. By shifting exploratory test design from a task requiring senior expertise to an automated process accessible to novice testers, **IGEX** broadens the potential for integrating exploratory testing into diverse development contexts. However, the 'accuracy' metric reflects alignment with exploratory criteria, not fault detection capability, indicating the need for further validation.

Threats to Validity. As with any generative model evaluation, our study faces several limitations: - *Internal Validity:* Potential variability in classification due to reliance on prompt-based reasoning. - *Construct Validity:* Subjectivity in defining exploratory test quality. - *External Validity:* Evaluation limited to mobile scenarios. - *Conclusion Validity:* Final accuracy must be cautiously interpreted as classification success, not empirical efficacy.

Reflexive Evaluator Accuracy Assessment. To further assess the reliability of the Reflective Evaluator, we conducted an additional validation test. From all of the seven experimental configurations in our ablation study, we randomly selected 100 ETCs and their respective evaluations, resulting in a total of 50 positive evaluations (classified as exploratory) and 50 negative evaluations (classified as non-exploratory). This yielded a set of 100 cases independently re-assessed to verify the accuracy of the automatic classification.

The manual review showed that 85 of the 100 evaluations were correctly classified by the Reflective Evaluator, while 15 were misclassified. Among the 50 cases automatically classified as exploratory, 43 were validated as correct (86%), with 7 identified as incorrect (14%). Similarly, among the 50 cases classified as non-exploratory, 42 were confirmed as correct (84%) and 8 were misclassified (16%).

While an overall accuracy of 85% indicates a promising level of performance, it also underscores the need for caution when relying solely on automated evaluations for quality assurance. These findings reinforce the utility of the Reflective Evaluator within **IGEX** as a supportive mechanism for iterative refinement, though they do not eliminate the potential value of incorporating human oversight or additional contextual reasoning techniques in future work.

6 Conclusion

This paper introduced **IGEX**, a method that combines structured prompt chaining and reflective evaluation to automate the generation of ETCs from general descriptions. Our results show that large language models, when guided by modular prompts and reflective feedback, can produce structured test cases aligned with expert-defined exploratory criteria. While further validation is needed in practical testing environments, **IGEX** represents a promising step toward scalable and consistent exploratory test generation. Future work includes extending **IGEX** to other domains, assessing its defect-finding capabilities, and refining the evaluation process with human-in-the-loop strategies.

ARTIFACT AVAILABILITY

In compliance with confidentiality policies and data protection agreements, datasets used in this study cannot be publicly disclosed, as they contain sensitive information and internal processes, including mobile device validation flows, file naming conventions, and proprietary commands. Access to these datasets has been strictly limited to internal research and development purposes, in accordance with the terms set forth in confidentiality agreements (NDAs).

ACKNOWLEDGMENTS

This work was supported by the following Brazilian agencies: the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior-Brasil (CAPES-PROEX)- Finance Code 001, the National Council for Scientific and Technological Development (CNPq), and Amazonas State Research Support Foundation- FAPEAM- through the POSGRAD project 2024/2025, and by Motorola Mobility Comércio de Produtos Eletrônicos Ltda, under the terms of Federal Law No. 8.387/1991, through agreement No. 02/2021 with ICOMP/UFAM.

REFERENCES

[1] W. R. Adrion, M. A. Branstad, and J. C. Cherniavsky. 1982. Validation, Verification, and Testing of Computer Software. *Comput. Surveys* 14, 2 (June 1982), 159–192. doi:10.1145/356876.356879

[2] Wasif Afzal, Ahmad Nauman Ghazi, Juha Itkonen, Richard Torkar, Anneliese Andrews, and Khurram Bhatti. 2014. An Experiment on the Effectiveness and Efficiency of Exploratory Testing. *Empirical Software Engineering* 20 (06 2014). doi:10.1007/s10664-014-9301-4

[3] Meta AI. 2024. Llama 3 Technical Report. https://scontent.fmao1-1.fna.fbcdn.net/v/t39.2365-6/468347782_9231729823505907_4580471254289036098_n.pdf. Accessed: 2024-05-30.

[4] F. Asplund. 2018. *Exploratory testing: Do contextual factors influence software fault identification?* Ph. D. Dissertation. KTH Royal Institute of Technology, Stockholm, Sweden.

[5] J. Bach. 2000. Session-Based Test Management. *Software Testing and Quality Engineering (STQE)*, vol. 2, no. 6.

[6] J. Bach. 2004. Exploratory Testing. 253–265 pages. Chapter in edited book.

[7] K. Brush. 2024. Test Case Definition. https://www.techtarget.com/searchsoftwarequality/definition/test-case Accessed: Oct. 31, 2024.

[8] I. E. F. Costa, A. C. dos Santos, F. A. B. Silva, and E. B. Araujo. 2023. Using Active Methodologies for Teaching and Learning of Exploratory Test Design and Execution. *Education Sciences*, vol. 13, art. no. 115. Retrieved May 12, 2025 from https://doi.org/10.3390/educsci13020115

[9] R. D. Craig and S. P. Jaskiel. 2002. *Systematic Software Testing.* Artech House Publishers, Boston.

[10] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. 2024. A Survey on In-context Learning. arXiv:2301.00234 [cs.CL] https://arxiv.org/abs/2301.00234

[11] M. Ellims, J. Bridges, and D. C. Ince. 2006. The Economics of Unit Testing. *Empirical Software Engineering*, vol. 11, no. 1, pp. 5–31. Retrieved May 12, 2025 from https://doi.org/10.1007/s10664-006-7585-6

[12] T. Ericson, A. Subotic, and S. Ursing. 1997. Tim: A Test Improvement Model. *Software Testing, Verification and Reliability*, vol. 7, no. 4, pp. 229–246. Retrieved May 12, 2025 from https://doi.org/10.1002/(SICI)1099-1689(199712)7:4<229::AID-STVR148>3.0.CO;2-V

[13] B. Hailpern and P. Santhanam. 2002. Software Debugging, Testing, and Verification. *IBM Systems Journal*, vol. 41, no. 1, pp. 4–12. Retrieved May 12, 2025 from https://doi.org/10.1147/sj.411.0004

[14] J. Itkonen and M. V. Mäntylä. 2014. Are test cases needed? Replicated comparison between exploratory and test-case-based software testing. *Empirical Software Engineering* 19 (2014), 303–342. doi:10.1007/s10664-013-9266-8

[15] J. Itkonen and K. Rautiainen. 2005. Exploratory testing: A multiple case study. In *Proc. 2005 Int. Symp. Empirical Software Engineering (ISESE).* 1–10. doi:10.1109/ISESE.2005.1541817

[16] M. Keployio. 2024. Exploring test case generators: Revolutionizing software testing. https://medium.com/@keployio/exploring-test-case-generators-revolutionizing-software-testing-9a8f80153dc8. Accessed: May 21, 2024.

[17] W. Makondo et al. 2016. Exploratory Test Oracle Using MLP Neural Networks. In *Proceedings of the International Conference on Advances in Computing, Communications and Informatics (ICACCI).*

[18] Y. Nishi and Y. Shibasaki. 2021. Boosted Exploratory Test Architecture. In *IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW).*

[19] H. H. Olsson, H. Alahyari, and J. Bosch. 2012. Climbing the 'stairway to heaven' – a multiple-case study exploring barriers in the transition from agile development towards continuous deployment of software. In *Proc. 38th EUROMICRO Conf. Software Engineering and Advanced Applications (SEAA).* 392–399.

[20] A. M. Sami et al. 2024. A Tool for Test Case Scenarios Generation Using LLMs. *arXiv preprint arXiv:2406.07021* (2024). https://arxiv.org/abs/2406.07021

[21] J. Sternerson and M. Firoozfam. 2010. Statistical Testing using Automated Randomization. Technical report or workshop paper, details unavailable.

[22] Y. Su et al. 2024. Enhancing Exploratory Testing by Large Language Model and Knowledge Graph. In *Proceedings of the 46th International Conference on Software Engineering (ICSE).*

[23] A. Tinkham and C. Kaner. 2003. Exploring Exploratory Testing. http://kaner.com/pdfs/ExploringExploratoryTesting.pdf. Accessed: Nov. 4, 2024.

[24] J. Wei, X. Wang, D. Schuurmans, and Q. Le. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv preprint arXiv:2201.11903. Retrieved May 12, 2025 from https://arxiv.org/abs/2201.11903

[25] J. A. Whittaker. 2011. Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design. Retrieved May 12, 2025 from https://dokumen.pub/exploratory-software-testing-tips-tricks-tours-and-techniques-to-guide-test-design-9780321636416-0321636414.html Accessed online.

[26] T. Wu, L. Wang, X. Liu, Z. Hu, and M. Sun. 2022. PromptChainer: Chaining Large Language Model Prompts through Visual Programming. arXiv preprint arXiv:2203.06566. Retrieved May 12, 2025 from https://arxiv.org/abs/2203.06566

[27] D. Xu. 2015. *An Automated Test Generation Technique for Software QA.* Master's thesis. Boise State University.

[28] W. X. Zhao, J. Li, Y. He, X. Yan, M. Zhou, and H. Chen. 2024. A Survey of Large Language Models. arXiv preprint arXiv:2303.18223. Retrieved May 12, 2025 from https://arxiv.org/abs/2303.18223