Test Generation from Use Case Specifications for IoT Systems: Custom, LLM-Based, and Hybrid Approaches

1st Zacharie Chenail-Larcher
Département de Génie Logiciel et des TI
École de Technologie Supérieure
Montreal, QC, Canada

2nd Jean Baptiste MINANI
Computer Science and SE Department
Concordia University
Montreal,QC, Canada

3rd Naouel Moha
Département de Génie Logiciel et des TI
École de Technologie Supérieure
Montreal, QC, Canada

2025 IEEE Conference on Software Testing, Verification and Validation (ICST) | 979-8-3315-0814-2/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICST62969.2025.10988996

***Abstract*—IoT systems are increasingly developed and deployed across various domains, where End-to-End (E2E) testing is critical to ensure reliability and expected behavior. However, generating comprehensive tests remains challenging due to the heterogeneity, distributed nature, and unique characteristics of IoT systems, which limit the effectiveness of generic test generation approaches. Recent studies demonstrated the effectiveness of Large Language Models (LLM) for test generation in traditional software systems. Building on this foundation, this study explores and evaluates four distinct approaches for generating E2E tests from use case specifications (UCSs) tailored to IoT systems. These include (1) a custom, (2) a single-stage LLM, (3) a multi-stage LLM, and (4) a hybrid approach combining custom and LLM capabilities. We evaluated these approaches on an IoT system, focusing on correctness and scenario coverage criteria. Experimental results indicate that all approaches perform well, with notable variations in specific aspects of test generation. The custom and hybrid approaches are more reliable in producing correctly structured and complete tests, with the hybrid approach slightly outperforming others. This study is a work in progress, requiring further investigation to fully realize its potential.**

***Index Terms*—Automated Testing, Test Generation, IoT System Testing, End-to-End Testing.**

I. Introduction

As IoT systems grow in complexity and ubiquity, testing such systems is a key element to their success in maximizing their benefits and minimizing risks [1]. IoT systems are inherently heterogeneous, comprising diverse devices, protocols, sensors, and actuators [2]. This complexity necessitates rigorous testing to validate their functionality and reliability [3]–[5]. However, testing these systems is still challenging due to their dynamic and complex nature plus the need to test multiple layers [6]–[8]. Many IoT systems are composed of four layers: sensing layer, network layer, cloud layer, and application layer [8], [9]. Some studies focus on testing specific layers, such as testing devices [10]–[12], while others focus on testing specific aspects of IoT systems such as performance [13]–[15], interoperability [13], [16], or security [17]–[19]. Testing the entire IoT system, referred to in this paper as End-to-End (E2E) testing, involves evaluating the complete flow, from the sensing layer to the delivery of services to end-users. Creating tests for E2E testing is a complex task, particularly for IoT systems with diverse communication protocols, devices, and interactions—both between devices themselves and between devices and other layers [20]. Existing test generation approaches and tools fall short of addressing the specific requirements of IoT systems [8]. Many existing test generation approaches rely on having access to the system's source code or detailed behavioral model of the system under test (SUT). However, in IoT systems, source code for some devices may be unavailable [21], and many small devices lack the resources to run tests like unit tests, highlighting the need for IoT-specific solutions. Moreover, the behavioral models may lack sufficient detail to support automated testing. This paper explores approaches to automate E2E test generation for IoT systems from use case specifications. We propose and evaluate four approaches:

- **Custom approach (CA)** uses logical tree traversal and data extraction process to generate tests.
- **Single-Stage LLM approach (SSLA)** processes the entire task with one detailed prompt in one step.
- **Multi-Stage LLM approach (MSLA)** involves chaining multiple LLMs where the output of one LLM serves as the input for another, enabling complex task decomposition.
- **Hybrid approach (HA)** combines custom and LLM approaches for better accuracy.

We conducted experiments and evaluated each approach. We formulated the following research questions:

- **RQ1:** How accurate is the scenario coverage from UCS?
- **RQ2:** How is the correctness of the generated tests?

The contributions of this paper are twofold: (1) We introduce an IoT system UCS description format that can be understood by LLMs to extract the necessary information for test generation. (2) We propose various approaches to generate IoT system-specific tests from the given UCSs and evaluate their accuracy. The rest of this paper is organized as follows: Section II provides the background of key concepts used in this paper. Section III reviews closely related work. Section IV describes our approaches. Section V presents our preliminary results. Section VII discusses threats to validity, and Section VIII concludes the paper with suggestions for future work.

II. Background

This section explains two concepts used in this paper: use case specifications (UCSs) and tests for testing of IoT systems.

- **UCSs** describe the interactions between users, devices, and system components to achieve specific goals. These specifications outline functional requirements by detailing the roles of actors (e.g., users, sensors, devices, cloud services), communication flows, conditions, and expected outcomes. They often include aspects unique to IoT systems, such as devices, and protocol usage, ensuring clarity in the testing of complex IoT ecosystems. It retains the same fields as the widely used ones (UML, RUCM [22], [23]), like use case name, description, basic and alternative flows (comprised of steps), etc. However, it also has additional fields:
  - Components include IoT devices, cloud applications, gateways, or other entities within the IoT ecosystem.
  - Protocol, method, operation, component, and inputs fields inside the input flow steps.
  - Outputs field inside the output flow steps (responses of actions) to formally document the outputs.
  - Condition field inside the alternative flows to specify the branching conditions as JSON Schema properties.
- **Test Scenarios** are abstract paths extracted from UCSs.
- A **Test** is a sequence of executable instructions that trigger the system under test. It is a sequence of operations executed in the system under test (SUT) to verify the test target [24]. It includes one or more executable test oracles [25]. Below is an example of one of the generated tests by our approaches.

```
{
    "TC_ID": "TC001",
    "name": "Heartrate Check",
    "steps": [
        {
            "operation": "getFitbitData",
            "target": {
                "protocol": "HTTP",
                "method": "POST",
                "name": "Fitbit"
            },
            "inputs": {
                "data": bpm
            },
            "expectations": {
                "msg": "SUCCESS"
            }
        }
    ]
}
```

A complete test may comprise multiple steps, each including the operation being tested, the target (protocol, method, and component), and the inputs. More details in our Zenodo repository [26].

III. Related Work

Recent research has explored various approaches to testing IoT systems, with a focus on either comprehensive system-level testing or automated test generation from use case specifications. Medhat et al. [27] prioritized IoT test cases using deep learning and search-based techniques, focusing on test execution optimization, whereas we emphasize E2E test case generation from use case specifications (UCSs) using LLMs. Jean et al. [28] generated unit tests for IoT devices from source code, which is often inaccessible in IoT systems, unlike our system-level E2E testing approach. Garn et al. [29] combined recursive algorithms, logic trees, and LLMs for combinatorial parameter testing; in contrast, we focus on entire execution flows derived from UCSs. Piparia et al. [30] automated test generation for Android applications based on user-context changes, while we address IoT-specific E2E tests. Yang et al. [31] tackled IoT security through firmware fuzzing, whereas we focus on general functionality testing. Jiang et al. [32] and Wang et al. [25] generated tests from UCSs but lacked support for complex IoT systems, a gap we address with IoT-specific methodologies. Naimi et al. [33] used LLMs to generate UML-based tests, while we propose a modular pipeline for IoT UCSs. Bhavya et al. [34] used RNNs for device-specific testing, unlike our comprehensive system-level focus. Olianas et al. [35] and Leotta et al. [36], [37] generated tests from UML models but were limited by incomplete models; our UCS-based approach overcomes this issue. The CT-IoT framework [38] optimized testing paths using combinatorial techniques, whereas we focus on UCS-driven E2E test generation. Lonetti et al. [24] provided a review on model-based security testing. Finally, Pontes et al. [39] introduced IoT-TaaS for interoperability testing, distinct from our UCS-based functional test generation. Table I summarizes the related work closely related to our study. While prior research has explored specific aspects of IoT testing, our study focuses on generating E2E test cases directly from UCSs, leveraging custom and LLM-based approaches.

IV. Approaches

We explore 4 approaches to generate tests from UCS, each offering distinct characteristics and benefits: Custom Approach (CA), Single-Stage LLM Approach (SSLA), Multi-Stage LLM Approach (MSLA), and Hybrid Approach (HA).

A. *Custom Approach (CA)*

The Custom Approach (CA) shown in Figure 1 uses logic trees and parsing algorithms to extract data directly from use case structures. Though partially inspired by recent studies on test generation from UCS for embedded systems [25], it does not use any advanced AI or NLP techniques. The CA implementation consists of five steps: use case extraction, test model generation, scenario extraction, node pairing, and test generation. The CA begins by the **extraction of the use case** from its JSON format. Its file is read, and steps from all execution flows are classified into types: INPUT (IoT communication), CONDITION (branch start), OUTPUT (response), STEP (internal action), ABORT (premature end), and EXIT (normal end). Classification is based on step position, content, fields, and branching. Next is the **generation of a Use Case Test Model (UCTM)**, a logic tree representing the use case for easy traversal and analysis. Each flow step becomes a node containing its extracted information and each branching relationship is stored as a list of strings. Most Nodes only have a single empty list, but CONDITION nodes have multiple lists of strings, each containing the data constraints

TABLE I
CLOSELY RELATED WORK

					Output				OUT		Automation		
Study	Year	Target Systems	Focus	Inputs	Test Scenarios	Test Cases	Test Oracle	Test Scripts	Layer	E2E	Manual	Semi	Full
[40]	2024	Web Applications	Functionalities	Code	–	✚	–	–	–	✚	–	–	✚
[41]	2023	IoT Systems	Events	Event Logs	–	✚	–	✚	✚	–	–	–	✚
[25]	2022	Embedded Systems	Functionalities	UCS	–	–	–	–	–	–	–	–	–
[38]	2022	IoT Systems	Execution Paths	UML Behavioral Models	–	–	–	–	–	–	–	–	–
[42]	2022	Embedded Systems	Functionalities	UCS	–	✚	–	✚	–	–	–	–	✚
[35]	2021	IoT Systems	Functionalities	UML Behavioral Models	✚	✚	✚	✚	–	✚	–	✚	–
[43]	2021	IoT Systems	Event Scenarios	Event types	–	✚	–	–	✚	✚	–	–	✚
[44]	2019	IIoT Systems	Security	Commands	–	✚	–	–	✚	✚	✚	–	–
[45]	2015	Software in general	Functionalities	System Models	✚	✚	–	–	–	✚	–	–	✚
[46]	2012	Embedded Systems	Conformance	UML Behavioral Models	–	✚	–	–	–	–	–	✚	✚

✚: Covered in the study. –: Not covered in the study. **OUT**: Object Under Test (A specific layer, such as sensing/device layer or entire system (E2E)).

for the branchings. Constraints for alternate branching are inverted using the JSON Schema "not" clause to ensure only one condition is true at a time, maintaining accurate constraint application across branches. The approach then proceeds to the **scenarios extraction**. A depth-first search (DFS) algorithm parses the UTCM, exploring all branches to create a scenario for each unique execution path, extracting and regrouping Nodes and their relevant data constraints. For each scenario extracted, **key nodes are paired** into groups that will later become test steps. Input nodes are paired with condition and output nodes. Each input node defines the protocol, method, target component, operation, and data being sent. Condition nodes specify data constraints by combining their JSON Schema properties into a complete schema using "allOf" clauses and a schema skeleton. Output nodes detail the test stage's expectations. Finally, we move to the **test generation**. Information is extracted from the paired nodes into test steps. Valid input values are generated using the JSON Schema Faker library[1] with the constructed JSON Schema. The test steps are then combined to form complete test flows. Test IDs are assigned incrementally, and test names are derived from post-conditions.

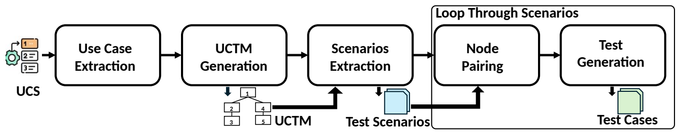

Fig. 1. Custom Approach (CA)

If the input use case is complete, correctly formatted, and defect-free, CA should ensure accurate and exhaustive scenario coverage. By reorganizing formal data and generating values from precise constraints, perfect inputs yield perfect outputs, offering reliability, consistency, and comprehensive coverage. However, the approach is highly sensitive to input quality (i.e., UCSs), with any errors or omissions affecting the results.

B. Single-Stage LLM Approach (SSLA)

The SSLA shown in Figure 2 uses an LLM to generate tests in a single, detailed prompt that includes the use case, test format, and detailed instructions. The use case is inserted into a premade prompt and sent to ChatGPT-4o via its user interface. While flexible and effective for simple use cases, it may struggle with complex scenarios, yielding inconsistent results and limited scenario coverage.

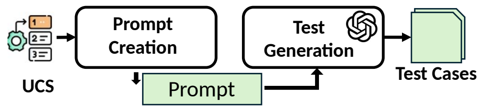

Fig. 2. Single-Stage LLM Approach (SSLA)

C. Multi-Stage LLM Approach (MSLA)

MSLA shown in Figure 3 also uses LLMs, but divides the process into multiple prompts for greater precision and granularity. Reducing the workload per prompt minimizes errors, misunderstandings, and omissions. It first takes the use case from its JSON file and uses it in the prompt, which is sent to ChatGPT-4o via the OpenAI API to identify all scenarios. The API response is then analyzed to separate the scenarios. Each scenario is sent back in a prompt to ChatGPT-4o, and each response contains an executable test. This approach retains SSLA's flexibility while improving accuracy.

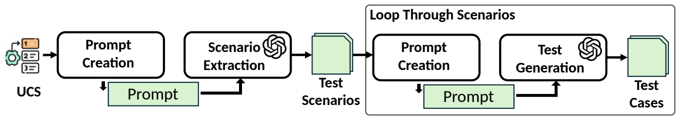

Fig. 3. Multi-Stage LLM Approach (MSLA)

D. Hybrid Approach (HA)

The HA shown in Figure 4 combines elements of CA and LLM-based methods, leveraging the strengths of both. It assigns specific tasks to logical algorithms or LLMs to retain CA's reliability while enhancing flexibility. The first three steps mirror CA exactly, while the final two mirror MSLA instead, using ChatGPT-4o via the OpenAI API. This hybrid model aims to balance reliability, consistency, and flexibility, offering an improved user experience. Though still in its exploratory phase, the implementation highlights the potential of HA.

V. EXPERIMENTATION

To evaluate the different approaches, we used three use cases from the WIMP (Where Is My Professor) IoT System. WIMP is an IoT-based system designed to enable students to track the real-time availability of their professors. It gathers data from

[1]https://github.com/json-schema-faker/json-schema-faker

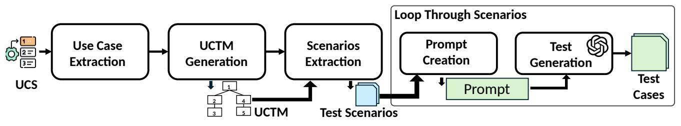

Fig. 4. Hybrid Approach (HA)

various IoT devices to determine the professor's location. We first converted the use case specifications into JSON format, and we manually created the expected test cases to serve as a reference. Using these specifications, we tested all four approaches and compared their outputs with what was expected. The evaluation covered three aspects: scenario coverage, the correctness of generated tests, and step field validation.

- **Scenario coverage** measures how well each approach identifies test scenarios from use cases by comparing generated tests to all valid scenarios (RQ1) as shown in Table II.
- **Correctness of generated tests** evaluates how accurately approaches identify the steps of E2E tests for each scenario (RQ2). A step is valid if it represents the correct operation, even if some fields are not entirely accurate.
- **Correctness of each step** assesses whether the fields within identified steps are fully accurate or contain errors (RQ2). It focuses on field-level validation. Table III and IV summarize our preliminary results for RQ2.

TABLE II
SCENARIOS GENERATION (RQ1)

	Scenarios Coverage		Accuracy	
Approach	Valid	Invalid	Precision	Recall
CA	100%	0%	100%	100%
SSLA	100%	0%	100%	100%
MSLA	95%	5%	95%	95%
HA	100%	0%	100%	100%

TABLE III
CORRECTNESS OF GENERATED TESTS (RQ2)

	Test Steps			Accuracy	
Approach	Coverage	Valid	Invalid	Precision	Recall
CA	100%	100%	0%	100%	100%
SSLA	77%	98%	2%	98%	77%
MSLA	100%	96%	4%	96%	100%
HA	100%	100%	0%	100%	100%

TABLE IV
CORRECTNESS OF EACH STEP IN GENERATED TEST (RQ2)

Approach	Operation	Protocol	Method	Node	Input	Expectation
CA	100%	100%	100%	100%	87%	100%
SSLA	100%	100%	100%	100%	100%	100%
MSLA	100%	100%	100%	100%	100%	91%
HA	100%	100%	100%	100%	96%	100%

The results[1] show good performance overall across the board. The scenario identification results demonstrate that in our experiment, all approaches performed perfectly in this aspect across all three use cases, except for the MSLA. This approach failed to identify one of the scenarios correctly, replacing it with one it invented (LLM Hallucination), resulting in coverage, precision, and recall of 95%, which is still good overall. To identify valid steps, both the CA and HA once again performed perfectly, while the SSLA and MSLA had some mistakes. The results for field accuracy reveal that the CA encountered some problems when creating valid inputs and that the MSLA struggled with expectations. The HA performed well except for the inputs, while the SSLA was surprisingly the best performer, achieving perfect scores for all fields.

VI. DISCUSSION

Results show the strengths and weaknesses of our approaches. The CA and HA excelled in scenario and step identification, with the HA performing best across all results. The CA faced input field errors due to JSON schema faker's limitations with complex schemas. LLM-based approaches had mixed results: the SSLA was very accurate but missed some steps and scenarios, while the MSLA covered all steps and scenarios but was less accurate because it broke steps into smaller parts and added incorrect information. Both approaches excelled in test input generation but occasionally produced invalid JSON outputs due to code comments. LLMs demonstrated adaptability with incomplete use cases, generating plausible values but struggling with ambiguous descriptions. To improve automated test generation for IoT systems, key enhancements include better prompt engineering, refining step granularity, optimizing task distribution in HA, and fine-tuning LLMs. Updates to IoT UCS formats and experimentation with multiple LLM models are also recommended to enhance reliability and robustness.

VII. THREATS TO VALIDITY

Our study faces several threats to validity. **Construct validity** is influenced by the quality of prompts and implementations, which directly affect accuracy, and potential biases in adapting UCSs to our IoT system. **Internal validity** is limited by the IoT UCS format, which lacks support for alternative flows related to operations, protocols, and target nodes, and requires a rigid step sequence that may not reflect realistic scenarios. Additionally, the format's complexity increases the risk of errors, and missing alternative flows results in incomplete data rules, reducing accuracy. The reliance on structured data without real textual analysis also restricts adaptability. **External validity** is constrained by the limited scope of our evaluation, using only three simple use cases from the same system, which may not generalize to more complex scenarios.

VIII. CONCLUSION AND FUTURE WORK

This study[1] explored four approaches for generating E2E tests for IoT systems from UCSs: CA, SSLA, MSLA, and HA. Each approach was evaluated on its ability to address IoT-specific challenges, including protocol diversity, device interoperability, and scenario coverage. The evaluation highlighted the strengths and limitations of each approach. HA demonstrated superior effectiveness in addressing these challenges by combining CA with LLM techniques. These findings provide valuable insights for researchers and practitioners aiming to enhance automated testing of IoT systems. Future work involves refining the current implementations and conducting evaluations on larger, more complex IoT systems to validate the generalizability.

References

[1] N. Medhat, S. Moussa, N. Badr, and M. F. Tolba, "Testing techniques in iot-based systems," in *2019 Ninth International Conference on Intelligent Computing and Information Systems (ICICIS)*, pp. 394–401, IEEE, 2019.

[2] X. Wu, J. Wang, P. Li, X. Luo, and Y. Yang, "Internet of things as complex networks," *IEEE Network*, vol. 35, no. 3, pp. 238–245, 2021.

[3] L. Antao, R. Pinto, J. Reis, and G. Gonçalves, "Requirements for testing and validating the industrial internet of things," in *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 110–115, IEEE, 2018.

[4] H. Kim, A. Ahmad, J. Hwang, H. Baqa, F. Le Gall, M. A. R. Ortega, and J. Song, "Iot-taas: Towards a prospective iot testing framework," *IEEE Access*, vol. 6, pp. 15480–15493, 2018.

[5] I. Haddou-Oumouloud, A. Kriouile, S. Hamida, and A. Ettalbi, "Towards secure and reliable iot systems: A comprehensive review of formal methods applications," *IEEE Access*, 2024.

[6] E. J. Marinissen, Y. Zorian, M. Konijnenburg, C.-T. Huang, P.-H. Hsieh, P. Cockburn, J. Delvaux, V. Rožić, B. Yang, D. Singelée, *et al.*, "Iot: Source of test challenges," in *2016 21th IEEE European test symposium (ETS)*, pp. 1–10, IEEE, 2016.

[7] P. M. Pontes, B. Lima, and J. P. Faria, "Test patterns for IoT," in *Proceedings of the 9th ACM SIGSOFT international workshop on automating TEST case design, selection, and evaluation*, pp. 63–66, 2018.

[8] J. B. Minani, F. Sabir, N. Moha, and Y.-G. Guéhéneuc, "A systematic review of iot systems testing: Objectives, approaches, tools, and challenges," *IEEE Transactions on Software Engineering*, 2024.

[9] J. B. Minani, F. Sabir, N. Moha, and Y.-G. Guéhéneuc, "A multi-method study of internet of things systems testing in industry," *IEEE Internet of Things Journal*, 2023.

[10] J. B. Minani, F. Sabir, Y. El Fellah, and N. Moha, "Towards an automated approach for testing iot devices," in *Proceedings of the ACM/IEEE 6th International Workshop on Software Engineering Research & Practices for the Internet of Things*, pp. 22–29, 2024.

[11] W.-K. Chen, C.-H. Liu, W. W.-Y. Liang, and M.-Y. Tsai, "Icat: An iot device compatibility testing tool," in *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pp. 668–672, IEEE, 2018.

[12] D.-H. Gong, "Iot device testing for efficient iot device framework," *International Journal of Internet, Broadcasting and Communication*, vol. 12, no. 2, pp. 77–82, 2020.

[13] E. E. Kim and S. Ziegler, "Towards an open framework of online interoperability and performance tests for the internet of things," in *2017 Global Internet of Things Summit (GIoTS)*, pp. 1–6, IEEE, 2017.

[14] L. E. Kane, J. J. Chen, R. Thomas, V. Liu, and M. Mckague, "Security and performance in iot: A balancing act," *IEEE access*, vol. 8, pp. 121969–121986, 2020.

[15] J. Esquiagola, L. C. de Paula Costa, P. Calcina, G. Fedrecheski, and M. Zuffo, "Performance testing of an internet of things platform.," in *IoTBDS*, pp. 309–314, 2017.

[16] S. T. Demirel, M. Demirel, I. Dogru, and R. Das, "Interopt: A new testing platform based on onem2m standards for iot systems," in *2019 International Symposium on Networks, Computers and Communications (ISNCC)*, pp. 1–6, IEEE, 2019.

[17] A. R. Chandan and V. D. Khairnar, "Security testing methodology of iot," in *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*, pp. 1431–1435, IEEE, 2018.

[18] P. A. Abdalla and C. Varol, "Testing iot security: The case study of an ip camera," in *2020 8th International Symposium on Digital Forensics and Security (ISDFS)*, pp. 1–5, IEEE, 2020.

[19] N. Khezemi, J. B. Minani, F. Sabir, N. Moha, Y.-G. Guéhéneuc, and G. El Boussaidi, "A systematic literature review of IoT system architectural styles and their quality requirements," *IEEE Internet of Things Journal*, vol. 11, no. 23, pp. 37599–37616, 2024.

[20] J. B. Minani, F. Sabir, Y. El Fellah, and N. Moha, "Practical guidance for iot systems testing: A taxonomy," in *Proceedings of the ACM/IEEE 6th International Workshop on Software Engineering Research & Practices for the Internet of Things*, pp. 57–64, 2024.

[21] C. Wang, F. Pastore, A. Goknil, and L. C. Briand, "Automatic generation of acceptance test cases from use case specifications: an nlp-based approach," *IEEE Transactions on Software Engineering*, vol. 48, no. 2, pp. 585–616, 2020.

[22] T. Yue, L. C. Briand, and Y. Labiche, "A use case modeling approach to facilitate the transition towards analysis models: Concepts and empirical evaluation," in *Model Driven Engineering Languages and Systems* (A. Schürr and B. Selic, eds.), (Berlin, Heidelberg), pp. 484–498, Springer Berlin Heidelberg, 2009.

[23] T. Yue, L. C. Briand, and Y. Labiche, "Facilitating the transition from use case models to analysis models: Approach and experiments," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 22, no. 1, pp. 1–38, 2013.

[24] F. Lonetti, A. Bertolino, and F. Di Giandomenico, "Model-based security testing in IoT systems: A Rapid Review," *Information and Software Technology*, p. 107326, 2023.

[25] C. Wang, F. Pastore, A. Goknil, and L. C. Briand, "Automatic generation of acceptance test cases from use case specifications: an nlp-based approach," *IEEE Transactions on Software Engineering*, vol. 48, no. 2, pp. 585–616, 2020.

[26] Z. Chenail-Larcher, J. B. MINANI, and N. Moha, "Supplemental material / artefacts - test generation from use case specifications for iot systems: Custom, llm-based, and hybrid approaches.." https://doi.org/10.5281/zenodo.14751148, 2025.

[27] N. Medhat, S. M. Moussa, N. L. Badr, and M. F. Tolba, "A framework for continuous regression and integration testing in iot systems based on deep learning and search-based techniques," *IEEE Access*, vol. 8, pp. 215716–215726, 2020.

[28] J. B. Minani, F. Sabir, Y. El Fellah, and N. Moha, "Towards an automated approach for testing iot devices," in *Proceedings of the ACM/IEEE 6th International Workshop on Software Engineering Research & Practices for the Internet of Things*, pp. 22–29, 2024.

[29] B. Garn, D.-P. Schreiber, D. E. Simos, R. Kuhn, J. Voas, and R. Kacker, "Summary of combinatorial methods for testing internet of things smart home systems," in *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 266–267, IEEE, 2023.

[30] S. Piparia, D. Adamo, R. Bryce, H. Do, and B. Bryant, "Combinatorial testing of context aware android applications," in *2021 16th Conference on Computer Science and Intelligence Systems (FedCSIS)*, pp. 17–26, IEEE, 2021.

[31] W. Yang, H. Luo, C. Hu, and F. He, "A coverage-oriented fuzzing test method for embedded firmware," in *2024 10th International Symposium on System Security, Safety, and Reliability (ISSSR)*, pp. 244–250, IEEE, 2024.

[32] M. Jiang and Z. Ding, "Automation of test case generation from textual use cases," in *The 4th International Conference on Interaction Sciences*, pp. 102–107, IEEE, 2011.

[33] L. Naimi, M. Manaouch, A. Jakim, *et al.*, "A new approach for automatic test case generation from use case diagram using LLMs and prompt engineering," in *2024 International Conference on Circuit, Systems and Communication (ICCSC)*, pp. 1–5, IEEE, 2024.

[34] M. Bhavya, A. Damodaran, and S. Ranganath, "An ai based smart test case generator for embedded device," in *2022 Second International Conference on Power, Control and Computing Technologies (ICPC2T)*, pp. 1–5, IEEE, 2022.

[35] D. Olianas, M. Leotta, and F. Ricca, "MATTER: A tool for generating end-to-end IoT test scripts," *Software Quality Journal*, vol. 30, no. 2, pp. 389–423, 2022.

[36] M. Leotta, D. Clerissi, D. Olianas, F. Ricca, D. Ancona, G. Delzanno, L. Franceschini, and M. Ribaudo, "An acceptance testing approach for Internet of Things systems," *IET Software*, vol. 12, no. 5, 2018.

[37] M. Leotta, F. Ricca, D. Clerissi, D. Ancona, G. Delzanno, M. Ribaudo, and L. Franceschini, "Towards an acceptance testing approach for Internet of Things systems," in *Current Trends in Web Engineering*, pp. 125–138, Springer, 2018.

[38] L. Hu, W. E. Wong, D. R. Kuhn, R. N. Kacker, and S. Li, "CT-IoT: a combinatorial testing-based path selection framework for effective IoT testing," *Empirical Software Engineering*, vol. 27, pp. 1–38, 2022.

[39] P. M. Pontes, B. Lima, and J. P. Faria, "Izinto: A Pattern-Based IoT Testing Framework," in *Companion Proceedings for the ISSTA/ECOOP 2018 Workshops*, pp. 125–131, 2018.

[40] P. Alian, N. Nashid, M. Shahbandeh, T. Shabani, and A. Mesbah, "A feature-based approach to generating comprehensive end-to-end tests," *arXiv preprint arXiv:2408.01894*, 2024.

[41] S. Salva and J. Sue, "Automated test case generation for service composition from event logs," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*, pp. 127–134, IEEE, 2023.

[42] M. Bhavya, A. Damodaran, and S. Ranganath, "An ai based smart test case generator for embedded device," in *2022 Second International Conference on Power, Control and Computing Technologies (ICPC2T)*, pp. 1–5, IEEE, 2022.

[43] A. Velez-Estevez, L. Gutiérrez-Madroñal, and I. Medina-Bulo, "Iot-teg 4.0: a new approach 4.0 for test event generation," *IEEE Transactions on Reliability*, vol. 71, no. 3, pp. 1368–1380, 2021.

[44] S. Marksteiner, R. Ramler, and H. Sochor, "Integrating threat modeling and automated test case generation into industrialized software security testing," in *Proceedings of the Third Central European Cybersecurity Conference*, pp. 1–3, 2019.

[45] T. Yue, S. Ali, and M. Zhang, "RTCM: a natural language based, automated, and practical test case generation framework," in *Proceedings of the 2015 international symposium on software testing and analysis*, pp. 397–408, 2015.

[46] V. Chimisliu and F. Wotawa, "Model based test case generation for distributed embedded systems," in *2012 IEEE International Conference on Industrial Technology*, pp. 656–661, IEEE, 2012.