TAPAssure: Safe Automation in Smart Homes with LLMs and Formal Verification

Huan Bui[0009−0009−1364−2567], Chenglong Fu[0000−0001−6555−9858], and Fei Zuo[0000−0001−8902−1753]

[1] The University of North Carolina at Charlotte, Charlotte NC 28223, USA
hbui11@charlotte.edu
chenglong.fu@charlotte.edu
[2] The University of Central Oklahoma, Edmond OK 73034, USA
fzuo@uco.edu

Abstract. The integration of large language models (LLMs) in smart home systems offers a promising path for automating Trigger-Action Programming (TAP) rules. However, LLMs often generate incomplete or unsafe rules due to limited awareness of the IoT context. We present TAPAssure, a framework that combines LLM-based rule generation with formal verification to ensure safety and correctness. TAPAssure verifies rules using NuSMV and iteratively refines them based on model checker feedback to eliminate syntax errors and violations of user-defined safety properties. We evaluate TAPAssure using two state-of-the-art LLMs: Meta Llama 3.3 70B Instruct and Qwen2.5-14B-Instruct. Results show that Qwen2.5 resolves syntax errors efficiently (median of 3.5 iterations, average of 9.92 due to occasional outliers), while Llama3.3 excels in minimizing safety violations consistently (average of 5.48 iterations, 100% pass rate). Overall, TAPAssure effectively bridges the gap between intuitive LLM-driven automation and the stringent safety requirements of smart home environments.

Keywords: Trigger-Action Rules, Model Checking, Large Language Models, Formal Verification

1 Introduction

Over the past two years, LLMs have demonstrated impressive capabilities in content generation and reasoning, showing growing potential in smart home IoT systems. Existing research [14, 9, 8] has explored the use of LLMs for automating these systems by assisting users in generating and deploying TAP rules. However, despite their efficiency and convenience, LLMs frequently produce rules with errors or missing information, such as device IDs or installation locations, which can lead to unsafe or unintended behaviors. To address these issues and ensure safety, we introduce TAPAssure, a novel framework that integrates LLM-generated TAP rules with formal verification techniques.

LLMs, when tasked with generating TAP rules, may omit critical information or produce rules that inadvertently introduce unsafe behaviors. For instance, a

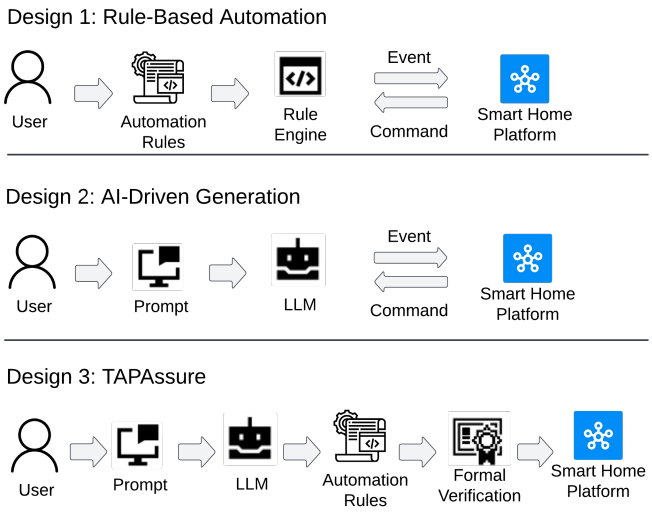


Fig. 1: Comparison of three automation design approaches in smart homes.

rule like "Unlock the front door when I arrive home" may seem intuitive but could pose security risks if deployed without constraints. Such vulnerabilities emphasize the need for a system that not only generates automation rules but also verifies their safety and correctness. While LLMs provide a user-friendly means of generating TAP rules, they lack the capability to detect violations of predefined safety properties or regenerate rules that violate those properties. Furthermore, in large-scale smart home environments with numerous interconnected devices, LLMs cannot effectively manage rule violations, increasing the risk of conflicting or unsafe automation (Figure 1).

To address these challenges, TAPAssure systematically integrates formal verification methods with LLM-generated TAP rules. TAPAssure converts user input in natural language into a formal specification, applies model checking to detect and minimize violations of safety properties, and then regenerates TAP rules that comply with these properties. Specifically, the system utilizes NuSMV [5], a symbolic model checker, to formally verify that generated rules adhere to linear temporal logic (LTL) safety constraints. If a rule fails verification, TAPAssure iteratively refines it by incorporating counterexamples provided by the model checker, ensuring that the final rule set is both functional and safe. The validated transition rules are then translated back into natural language, forming the refined automation scenarios.

Unlike existing platforms that rely on rigid, predefined rule templates, TAPAssure maintains the flexibility of natural language interaction while guaranteeing safety through formal methods. The framework's iterative refinement process, guided by model checking feedback, systematically eliminates syntax errors and safety violations that would otherwise compromise system security.

We evaluate TAPAssure with Meta Llama 3.3 and Qwen2.5, two state-of-the-art open-weight models that enable reproducible experimentation without reliance on proprietary APIs. Although popular models like ChatGPT or Claude are strong candidates, their closed nature limits systematic evaluation. Our current prototype relies on cloud-based inference, which raises privacy concerns for sensitive smart home data; future work will explore local deployment of smaller models or secure execution environments. Scalability also remains a challenge, as certain cases (e.g., 53 iterations with Qwen2.5) highlight the need for more efficient prompting and constrained decoding.

Our main contributions are:

- The application and evaluation of an architecture combining LLMs with formal verification specifically for smart home automation, addressing the fundamental tension between usability and safety.
- An iterative refinement mechanism that uses model checker counterexamples to guide LLM-based rule regeneration, ensuring convergence to safe configurations.
- A comprehensive evaluation demonstrating TAPAssure's effectiveness across 120 test cases using Meta Llama 3.3 and Qwen2.5, achieving up to 100% success in finding a safe, verifiable configuration.
- Practical insights into the strengths and limitations of different LLMs for safety-critical rule generation, informing future system design.

The remainder of this paper is organized as follows: Section 2 provides background on smart home automation and formal verification. Section 3 presents our system model and problem formulation. Section 4 details TAPAssure's architecture. Section 5 evaluates our approach. Section 6 discusses related work, and Section 7 concludes.

2 Background

In this section, we provide a comprehensive overview of the foundational concepts relevant to TAPAssure, including smart home environments, Trigger-Action Programming (TAP), large language models (LLMs), and formal verification methods.

2.1 Smart Home Automation Platforms

Smart home technology integrates various IoT devices, allowing automated control and enhanced interaction to improve comfort, convenience, energy efficiency, and security. Modern smart homes often involve numerous interconnected devices such as smart thermostats, lighting systems, cameras, sensors, and smart locks [1, 16]. These devices communicate through centralized or distributed IoT platforms like Amazon Alexa, Google Home, and Samsung SmartThings, offering users intuitive interfaces and automation capabilities.

While these systems significantly simplify daily routines, the complexity of inter-device interactions increases the likelihood of conflicts and safety risks. Therefore, ensuring robust automation through reliable rules generation and rigorous validation mechanisms remains critical for safe and effective deployment [12].

2.2 Trigger-Action Programming (TAP)

TAP is a widely adopted paradigm in smart home automation due to its intuitive and accessible structure, which pairs specific triggers (conditions or events) with corresponding actions [7, 17]. TAP allows users without extensive technical backgrounds to define complex automated behaviors easily. However, the simplicity and intuitiveness of TAP can lead to unintended interactions, ambiguities, or safety hazards if the rules generated are incomplete, conflicting, or insufficiently verified [11].

Existing studies, such as *Soteria* [10] and *IoTGuard* [4], have highlighted critical safety concerns within TAP-based systems. Soteria focuses on static analysis of IoT application source code to detect security and safety policy violations, whereas IoTGuard employs dynamic runtime enforcement of user-defined policies. However, both approaches have limitations: neither systematically applies formal verification techniques to the automation rules prior to their deployment. Consequently, automation rules validated only by static or runtime checks without formal verification may still fail to detect complex or latent violations, thus inadvertently breaching user-specified safety properties.

2.3 Large Language Models (LLMs)

LLMs represent a major advancement in natural language processing, significantly improving capabilities in language understanding, generation, and reasoning. Trained on extensive datasets, LLMs excel in interpreting complex natural language instructions and generating contextually appropriate responses [3, 13]. Their versatility makes them well-suited for tasks requiring intuitive interaction, such as generating TAP rules from user commands in smart homes.

Despite these strengths, LLMs have limitations, including potential inaccuracies, omissions, and vulnerabilities to adversarial inputs (e.g., prompt injections) [2, 19]. Thus, deploying LLMs in safety-critical applications, such as smart home automation, necessitates additional validation layers to ensure their outputs are secure and reliable.

2.4 Formal Verification and Model Checking

Formal verification methods provide rigorous frameworks for verifying system behaviors against predefined specifications, playing a crucial role in ensuring safety-critical applications operate correctly and reliably. Model checking, a formal verification technique, systematically explores all possible states of a system

to validate correctness properties expressed in formal logic (e.g., linear temporal logic or LTL) [6].

NuSMV, a widely-used symbolic model checker, facilitates efficient validation of system models against specified safety constraints, enabling the detection of potential rule conflicts or safety violations before deployment [5]. Integrating formal verification with rule generation processes in smart home systems is critical for guaranteeing that automated behaviors comply strictly with safety and security standards.

3 System Model and Problem Statement

In this section, we clearly define the system model utilized by TAPAssure and identify key problems that motivate our research.

3.1 System Model

The TAPAssure system operates within a smart home IoT environment comprising diverse interconnected devices, such as smart lights, thermostats, motion sensors, smart locks, and other home appliances. Users provide natural language commands specifying desired automation behaviors. These commands are processed by LLMs to generate initial TAP rules.

The smart home context, including device attributes (e.g., device IDs, capabilities, locations) and the home layout, significantly influences TAP rule generation accuracy. TAPAssure leverages this contextual information, ensuring generated rules are practical and operationally coherent. Once generated, TAP rules are translated into formal specifications represented using linear temporal logic (LTL), enabling rigorous verification through formal methods.

To validate these rules, TAPAssure integrates the NuSMV symbolic model checker, systematically verifying rules against specified safety properties. If violations or conflicts are identified, the TAP rules undergo iterative refinement informed by model checking results, ensuring final rules adhere strictly to safety requirements.

3.2 Problem Statement

The primary issue addressed in this research is the high potential for automation rules generated by LLMs to contain violations of predefined safety properties or to include syntax errors. Despite their efficiency in translating natural language commands into TAP rules, LLM-generated rules frequently exhibit errors such as incomplete information, missing critical details (e.g., device identifiers or exact locations), and unintended or unsafe behaviors.

Additionally, the lack of inherent verification mechanisms in existing LLM-driven automation systems substantially increases the likelihood of safety violations and security vulnerabilities. This risk is heightened in complex, large-scale

smart home systems, where interactions among numerous devices can lead to conflicting rules, unintended activations, or security breaches.

Therefore, it is crucial to develop robust verification processes capable of rigorously validating generated TAP rules for both syntactic correctness and adherence to safety constraints prior to deployment. TAPAssure addresses this critical challenge by integrating formal verification methods, systematically mitigating syntax errors and ensuring compliance with safety properties, thereby enhancing the reliability and security of smart home automation.

For evaluation, we constructed a dataset consisting of three scenario sets and four safety property sets. Each safety set contained between three and five distinct properties, covering constraints such as occupancy-aware lighting, temperature thresholds, and energy conservation. Combining these sets yielded twelve scenario–property pairs, which served as the benchmark for testing. To bound runtime, the iterative regeneration process was capped at 60 attempts per trial. In practice, most scenarios converged well before this threshold. This cap ensures that extreme outliers do not result in unbounded computational cost while still allowing sufficient opportunities for refinement.

3.3 Threat Model

Our threat model explicitly addresses vulnerabilities associated with TAP rules generated by LLMs, focusing on specific threats that arise due to the inherent characteristics of these models and their interactions with smart home environments. The primary threats considered include:

- **Ambiguity and Misinterpretation**: LLM-generated rules based on ambiguous or underspecified natural language inputs may result in automation behaviors that deviate from user intentions, potentially creating hazardous or unintended outcomes.
- **Incomplete or Incorrect Contextual Information**: Generated TAP rules may lack crucial context such as precise device identifiers or accurate installation locations. Missing or incorrect contextual information can lead to unsafe actions or conflicts within the smart home environment.
- **Syntax Errors in Formal Models**: The translation of natural language commands into formal specifications is inherently complex and can lead to syntax errors, causing misinterpretations or failures in automation deployment.
- **Complex Inter-device Conflicts**: In large-scale smart home systems with multiple interconnected devices, LLM-generated rules might unintentionally produce conflicting scenarios or rule interactions, leading to unpredictable system states and operational hazards.

To effectively mitigate these threats, TAPAssure incorporates a robust formal verification mechanism using NuSMV. This formal verification step systematically detects syntax errors, ambiguity-related issues, and potential safety property violations, allowing iterative refinement of generated TAP rules. Through

this verification-driven feedback loop, TAPAssure significantly reduces the risks associated with LLM-generated automation rules, ensuring safety, security, and operational correctness within smart home environments.

4 Design of TAPAssure

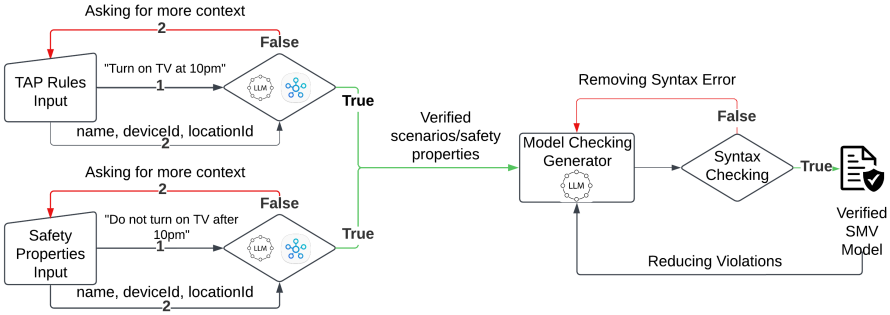


Fig. 2: TAPAssure's Design

TAPAssure consists of two primary processes aimed at ensuring that the generated TAP rules are both contextually accurate and compliant with safety properties. These processes work in tandem to minimize conflicts and enhance automation reliability. The overall design of TAPAssure is depicted in Figure 2, clearly illustrating the workflow from natural language input through formal verification and rule deployment. To illustrate, consider a scenario where "motion in the living room turns on the light" conflicts with a safety property requiring lights to remain off when the home is unoccupied. The initial model may compile but fail verification. TAPAssure uses the counterexample to regenerate a corrected rule such as "turn on the living room light if motion is detected and occupancy = true." This example shows how refinement gradually enforces safety constraints while preserving user intent. If regeneration fails to eliminate violations within the maximum iteration limit, TAPAssure flags the case and returns the partially refined model to the user for manual review, ensuring that unsafe rules are not silently deployed.

4.1 Context Verification and Initial Model Generation

The first process focuses on verifying context information provided by the user. This includes checking whether the mentioned devices exist in the smart home system and validating their installation locations. After this verification step, the user-provided scenarios and safety properties are reviewed to ensure completeness and consistency.

Users input their desired automation scenarios and home safety requirements in natural language. The LLM uses this input to generate TAP automation rules

and LTL formulas for safety properties. However, user inputs may contain incomplete references to IoT devices (e.g., "all lights in the living room"), potentially confusing the LLM and hindering the generation of deployable TAP rules. To address this issue, we integrated the SmartThings API as an external function call accessible by the LLM. This integration enables the LLM to automatically retrieve real-time contextual information directly from the SmartThings platform, including precise device names, unique device IDs, capabilities, and installation locations. If any contextual information remains incomplete or ambiguous after this retrieval, the LLM will prompt users to clarify or provide the missing details before proceeding.

Once the contextual information is validated, the LLM generates the initial version of the NuSMV model. This initial model may contain syntax errors. TAPAssure iteratively regenerates the model until it is syntactically correct, ensuring the NuSMV model is valid before proceeding to the next phase.

4.2 Iterative Model Regeneration to Minimize Violations

In the second process, TAPAssure iteratively regenerates the NuSMV model to minimize violations between the scenarios and safety properties. The generated NuSMV model undergoes formal verification using the NuSMV model checker. If violations are detected, TAPAssure prompts the LLM to regenerate the model, incorporating the counterexamples provided by the model checker. This iterative regeneration process continues until the model satisfies all safety properties, resulting in a set of safe and reliable TAP rules.

After validating the scenarios and safety properties from Section 4.1, the pre-trained model performs prompt-based validation to identify potential violations. TAPAssure employs two validation methods: prompt-based validation and NuSMV-based verification, enhancing the accuracy of the results. Prompt-based validation runs in parallel with the NuSMV model checker, which detects violations and regenerates transition states to minimize their occurrence. This regeneration process can be iterated multiple times to further reduce violations. Once refined, the transition rules are translated back into natural language to form updated scenarios. This dual-validation approach—combining prompt-based validation with formal verification—ensures a comprehensive assessment of TAP rules against safety properties. This iterative process ensures that the generated TAP rules are safe and reliable for smart home automation, as detailed in Algorithm 1. We made the project publicly available on GitHub[3].

If no violations are identified in either validation step, the model proceeds to generate TAP rules based on the verified scenarios and safety properties. These TAP rules are automatically added to the SmartThings system, enabling secure and reliable automation. By integrating user-defined scenarios, device validation, dual-stage verification, and rule generation, this phase enhances the reliability and safety of the smart home environment, streamlining the automation process.

[3] https://github.com/huuhuannt1998/tapassure

Algorithm 1 Verification with Context Validation and Model Checking

Input: Scenario S, Safety properties P **Output:** Verified NuSMV model M
$S \leftarrow \textsc{CompleteContext}(S)$; $P \leftarrow \textsc{CompleteContext}(P)$
$M \leftarrow \textsc{GenerateNuSMV}(S, P)$; $\textsc{EnsureValid}(M)$
repeat
  $V \leftarrow \textsc{ModelCheck}(M, P)$
  if $V = \emptyset$ **then return** M
  end if
  $M \leftarrow \textsc{Regenerate}(M, V)$; $\textsc{EnsureValid}(M)$
until false

where $\textsc{EnsureValid}(M)$: **while** $\neg\textsc{Valid}(M)$ **do** $M \leftarrow \textsc{FixSyntaxWithLLM}(M)$ **end while**
and $\textsc{CompleteContext}(X)$: **if** $\textsc{ContextIncomplete}(X)$ **then** $X \leftarrow \textsc{RequestMoreContext}(X)$; **return** $\textsc{CompleteContext}(X)$; **else return** X

5 Implementation and Evaluation

Tables 1 and 2 use labels formatted as "Scenario-Safety Property" pairs (e.g., "2-3" indicates scenario set 2 combined with safety property set 3). This labeling scheme is consistently applied throughout the evaluation to succinctly denote scenario-property combinations.

5.1 Implementation

The implementation of TAPAssure is structured into multiple designs, clearly demonstrating the evolution of the system from initial rule generation to comprehensive formal verification and refinement (Figure 3). Each design is specifically designed to enhance the accuracy, safety compliance, and reliability of Trigger-Action Program (TAP) rules within smart home environments.

Design Choice 1: Simple Generation (User Input) In the first design, the system accepts user input in the form of a scenario and generates corresponding TAP rules using an LLM. This represents the baseline capability of the LLM to interpret user-provided scenarios without any additional constraints or safety checks. Although simple, this approach may lead to potential issues, such as omitting essential conditions, potentially resulting in unsafe behaviors like continuous activation without proper shutdown conditions. For instance, the system might continuously activate a light whenever motion is detected without ever automatically turning it off.

Design Choice 2: Generation with Safety Property In the second design, users provide both a scenario and an explicit safety property, and the LLM generates rules that aim to comply with these specified constraints. This approach mitigates some risks identified in Design Choice 1 but still has limitations. Safety

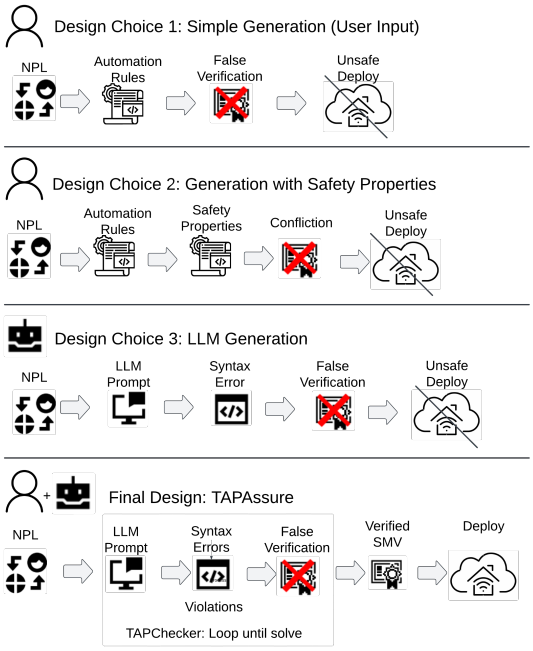


Fig. 3: Illustration of the Four Evolutionary Designs

compliance is not fully verified, and generated rules might include logical conflicts or incomplete constraints. For example, a rule could incorrectly implement timing constraints, leading to delayed responses or unintended interactions between devices.

Design Choice 3: Context Verification and Initial NuSMV Model Generation This design enhances the clarity and correctness of generated TAP rules by verifying contextual details such as device IDs and installation locations. The rules are then translated into formal specifications. However, due to the complexity involved, initial translations often contain syntax errors. These initial formal representations typically require iterative manual or automated corrections to address syntax or semantic errors. An example issue in this design might be an initial NuSMV model failing to compile due to missing variable declarations or incorrect temporal logic syntax.

Final Design: TAPAssure The final design leverages NuSMV for formal verification of TAP rules against specified safety properties. Identified violations during verification prompt iterative regeneration and refinement of TAP rules until all constraints are fully satisfied. This iterative approach ensures that generated rules fully comply with predefined safety constraints, significantly reducing the risk of unsafe behaviors. An advantage of this method is its capability to reliably manage complex scenarios, such as accurately switching off lights after no

motion has been detected for a specified period, thereby ensuring a safe and predictable smart home environment. The Llama3.3 and Qwen2.5 models were used without additional fine-tuning or training. Both models were accessed through their respective APIs in their pre-trained state, leveraging their inherent natural language understanding and code generation capabilities. The models' ability to generate NuSMV specifications and TAP rules relies on their pre-training on diverse programming and formal specification datasets.

5.2 Evaluation

We evaluated TAPAssure through two primary processes: **(1) Syntax Verification and Initial NuSMV Model Generation** and **(2) Iterative Model Regeneration to Minimize Violations**. The evaluation was conducted using two state-of-the-art large language models: Llama3.3 and Qwen2.5.

Dataset In the final deployment of TAPAssure, users are expected to provide their own scenarios and safety properties, which will then be verified as described in Section 4.1. However, for the purposes of evaluation, we used a different set of predefined scenario–safety property pairs to facilitate controlled testing. To systematically evaluate TAPAssure, we constructed a dedicated experimental dataset consisting of realistic smart home scenarios and corresponding safety properties. Specifically, we created 3 distinct sets of scenarios and 4 distinct sets of safety properties. Each scenario set was combined individually with each set of safety properties, resulting in a total of 12 unique scenario-property combinations for testing. We assume that these scenarios and safety properties have been verified for correctness and coherence prior to our evaluation. Each of the 12 combination sets was subjected to rigorous testing, with each combination evaluated through 10 independent runs to ensure robust statistical significance.

Metrics We evaluated TAPAssure's performance using two primary metrics:

- **Pass Rate**: Measures the effectiveness of the LLMs in successfully minimizing safety violations as determined by the NuSMV verification process. The pass rate is calculated as the percentage of runs where the generated rules fully complied with all safety properties after iterative regeneration. A run is considered successful only when all safety violations are eliminated, regardless of the number of regeneration iterations required.
- **Regeneration Failure Rate**: Assesses instances where the LLMs fail to effectively reduce or eliminate safety violations through the iterative process. Regeneration failures occur in two scenarios: (1) when the LLM fails to generate syntactically valid rules that can be verified by NuSMV, or (2) when subsequent regeneration attempts result in either no improvement or an increased number of violations compared to the previous iteration. The regeneration failure rate is computed as the percentage of runs where the LLM-generated rules did not successfully resolve all identified violations

within the maximum number of allowed iterations. These two metrics are complementary, with Pass Rate + Regeneration Failure Rate = 100%, providing a comprehensive view of the LLMs' capability to generate and refine safety-compliant rules through iterative feedback.

In addition to the pass rate, we report several complementary measures:

- **Average number of iterations** required to reach a valid and safe model, capturing efficiency.
- **Runtime per iteration and total runtime**, reflecting practical deployment costs.

These measures provide a richer understanding of how TAPAssure balances correctness, efficiency, and scalability across models. For example, while Llama3.3 consistently reached a safe configuration in fewer iterations, Qwen2.5 showed faster syntax correction but occasional outliers requiring significantly more iterations. Including these metrics highlights trade-offs between reliability and efficiency.

The Llama3.3 model achieved an exceptional **100% pass rate**, successfully minimizing safety violations in all 120 test cases without encountering any regeneration failures. On the other hand, Qwen2.5 successfully minimized violations in 112 out of 120 test cases, achieving a pass rate of **93.3%**. However, Qwen2.5 exhibited a regeneration failure rate of **6.7%**, corresponding to 8 test cases in which the number of violations increased.

Syntax Verification and Initial NuSMV Model Generation TAPAssure translates user-provided scenarios and safety properties into an initial NuSMV model. Due to the inherent complexity of converting natural language instructions into formal specifications, syntax errors frequently arise. Table 1 summarizes the performance of both LLMs in terms of iterations required to eliminate syntax errors:

From Table 1, Qwen2.5 generally requires fewer iterations than Llama3.3, demonstrating higher efficiency in most scenarios. However, notable exceptions occur in scenarios 2-4 and 3-2, where Qwen2.5 exhibited significantly higher iteration counts (22 and 53 iterations, respectively), suggesting occasional sensitivity to complex structures. Table 1 and Figure 4a show that the notably high iteration count for Qwen2.5 (53 iterations in scenario 3-2) underscores sensitivity to prompt phrasing. For instance, minor variations such as "activate A/C if temperature exceeds 78°F" compared to "A/C should activate when temperature is above 78°F" notably impacted iteration efficiency, indicating a need for consistent prompt engineering.

Higher iteration counts may also affect practical usability through increased computational load and delayed responsiveness in smart home contexts, emphasizing the importance of efficient prompt strategies.

Instead, this behavior likely reflects Qwen2.5's sensitivity to subtle variations in phrasing or prompt formatting during the generation process. Such instability

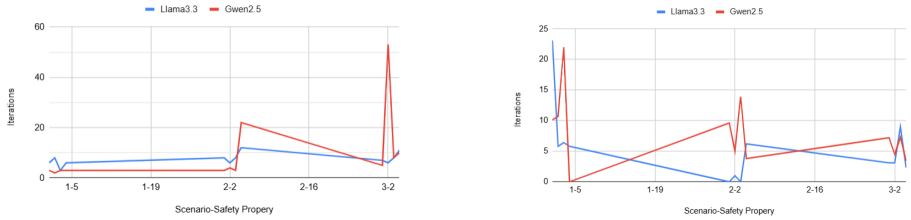
Table 1: Iterations to Eliminate Syntax Errors

Scen.-SafetyP.	Llama3.3	Qwen2.5
1-1	6	3
1-2	8	2
1-3	3	3
1-4	6	3
2-1	8	3
2-2	6	4
2-3	8	3
2-4	12	22
3-1	7	5
3-2	6	53
3-3	8	8
3-4	11	10

Table 2: Iterations to Minimize Violations

Scen.-SafetyP.	Llama3.3	Qwen2.5
1-1	23.1	10.1
1-2	5.8	10.7
1-3	6.4	22
1-4	5.8	0
2-1	0	9.6
2-2	1	5
2-3	0	13.9
2-4	6.2	3.8
3-1	3.1	7.2
3-2	3.1	4.5
3-3	9	7.3
3-4	2.3	3.4

is characteristic of LLMs, which can sometimes produce inconsistent or suboptimal outputs even when given similar inputs. These outlier cases highlight the importance of incorporating robust validation mechanisms, such as iterative syntax checking, to ensure model correctness despite occasional unpredictability.



(a) Fixing Syntax Error

(b) Minimizing Violations

Fig. 4: LLM's Performance

Iterative Model Regeneration to Minimize Violations Once a syntactically correct NuSMV model is established, TAPAssure employs iterative regeneration to minimize safety property violations detected by the NuSMV model checker. Table 2 presents these results:

The data in Table 2 indicates that Llama3.3 excels in efficiently resolving violations across multiple scenarios, notably sets 2-1, 2-2, and 2-3. Conversely, Qwen 2.5 shows higher variability in performance but achieves exceptional efficiency in set 1-4.

5.3 Statistical Analysis and Insights

A deeper statistical analysis of the evaluation data, including the mean, median, and standard deviation, provides a more nuanced understanding of each LLM's

performance. In the context of our framework, we define a model's precision by two key characteristics: its *efficiency* (the number of iterations required to find a solution) and its *reliability* (the consistency of its performance and its success rate).

Analysis of Syntax Error Resolution (Table 1) For the initial task of correcting syntax errors in the NuSMV model, the statistical results reveal a clear trade-off between speed and consistency.

- **Efficiency:** While Llama3.3 appears more efficient based on the mean (7.42 vs. 9.92), the median tells a different story. Qwen2.5's low median of 3.5 indicates that in a typical case, it is significantly more efficient, resolving syntax errors more than twice as fast as Llama3.3's typical performance (median of 7.5).
- **Reliability:** The standard deviation highlights a critical difference in precision. Llama3.3 exhibits a very low standard deviation of 2.54, signifying **high precision and predictable performance**. Its results are tightly clustered, meaning users can expect a consistent experience. Conversely, Qwen2.5 shows an extremely high standard deviation of 14.40. This volatility, driven by the outlier case of 53 iterations, indicates a lack of reliability. While often fast, it is prone to unpredictable and inefficient performance loops.

Analysis of Safety Violation Minimization (Table 2) For the more critical task of ensuring safety, the results demonstrate a clear superiority in Llama3.3's ability to perform precise and reliable work.

- **Efficiency:** In this complex reasoning task, Llama3.3 is unequivocally more efficient. Its low mean of 5.48 and median of 4.45 show that it consistently resolves safety-critical violations in fewer steps than Qwen2.5 (mean of 8.13, median of 7.25).
- **Reliability:** While both models have a similar standard deviation for this task (5.96 for Llama3.3 and 5.55 for Gwen2.5), the key differentiator is the overall success rate. Llama3.3's performance profile—combining its efficiency with the **perfect 100% pass rate** across all 120 test cases—is the ultimate proof of its precision. It demonstrates that Llama3.3 can not only find a safe configuration efficiently but can do so with complete reliability in our test suite. This stands in contrast to Qwen2.5, which was both slower and less reliable, failing in 6.7% of cases.

Conclusion on LLM Precision The statistical analysis validates our framework and provides crucial insights into model selection. The results prove that by integrating formal methods with a high-performing LLM, it is possible to achieve the precision required for safety-critical applications. For the primary goal of TAPAssure, which is ensuring user safety, Meta Llama 3.3's demonstrated ability to efficiently, consistently, and reliably minimize safety violations makes it the more precise and trustworthy choice for this framework.

Case Study: Original vs. Minimized NuSMV Models To evaluate the iterative violation minimization process in TAPAssure, we present a detailed case study of Scenario-Safety Property set 3-3. This includes the original NuSMV model, the minimized version generated after refinement, a comparison of their semantics, and the trends observed over 10 regeneration trials.

Violation Trend Analysis Llama3.3 Table 3 presents the number of iterations and violation counts across 10 independent trials for Scenario 3-3. In every trial, the maximum number of violations observed was 3, indicating consistent initial conflicts in the TAP rules prior to refinement. Notably, three trials (Trials 5, 7, and 10) successfully reduced the number of safety violations to a minimum of 1, demonstrating TAPAssure's ability to synthesize near-optimal rules in 30% of cases. Among these, Trial 7 was the fastest to converge, requiring only 7 iterations to reach the minimized violation count. In comparison, Trial 5 took 24 iterations and Trial 10 required 18 iterations, reflecting variation in convergence efficiency depending on the scenario structure and LLM responses.

The majority of the remaining trials (7 out of 10) converged to a minimum of 2 violations, with several of them doing so in just 3 to 7 iterations. This shows that TAPAssure typically achieves significant violation reduction with modest computational effort, even if not all violations are fully eliminated. The longest convergence time was observed in Trial 5, which required 24 iterations to reach its minimum, highlighting that certain input conditions may lead to more challenging optimization paths. Overall, the results demonstrate that TAPAssure consistently reduces safety violations within a reasonable number of iterations, though the extent and speed of convergence may vary across trials.

Table 3: Violation Trends Across Trials for Scenarios-Safety Properties 3-3 (Llama3.3)

Trial	**Iterations**	**Max Violations**	**Min Violations**
1	8	3	3
2	7	3	2
3	5	3	2
4	3	3	2
5	24	3	1
6	3	3	2
7	7	3	1
8	12	3	2
9	3	3	2
10	18	3	1

Violation Trend Analysis (Qwen2.5) Table 4 presents the number of iterations and violation counts across 10 independent trials for Scenario 3-3 using the Qwen2.5-14B model. Compared to LLama3.3, Qwen2.5 consistently achieved lower maximum violation counts, with no trial exceeding two safety violations. Notably, most trials (9 out of 10) reached a minimum violation count of 2, indicating a stable but less flexible convergence pattern. Only one trial (Trial 9)

successfully minimized the violations to a count of 1 after 19 iterations, which was the highest convergence time among all Qwen2.5 trials.

Trials converged quickly in general, with the fastest run (Trial 3) requiring just 2 iterations to reach its final violation state. Despite the absence of more aggressive optimization behavior observed in Llama3.3 (where 30% of trials reached only one violation), Qwen2.5 demonstrated more consistent and predictable refinement with reduced maximum violations. However, Trial 10 showed a slight regression, ending with a minimum violation count of 3—likely due to a compilation error or model inconsistency. Overall, Qwen2.5 shows promise in reliably reducing violations within a limited range but exhibits less variability in optimization depth compared to Llama3.3.

Table 4: Violation Trends Across Trials for Scenarios-Safety Properties 3-3 (Qwen2.5-14B)

Trial	Iterations	Max Violations	Min Violations
1	5	2	2
2	4	2	2
3	2	2	2
4	4	2	2
5	3	2	2
6	3	2	2
7	8	2	2
8	10	2	2
9	19	2	1
10	15	2	3

Comparison Between Original and Minimized Scenario-Safety Property Sets The minimized scenario-safety property set represents the outcome of TAPAssure's iterative refinement process guided by the LLM-generated NuSMV model. The primary objective of this minimization is to reduce safety violations and simplify rule logic rather than to replicate every detail from the original scenario explicitly. Thus, certain scenarios or conditions were intentionally simplified or omitted because they were not directly relevant to improving safety property compliance. To assess the effectiveness of TAPAssure's iterative model regeneration, we compared the original scenario-safety property set (Scenario - Safety Property 3-3) against the final minimized set derived through LLM-guided refinement. Table 5 summarizes this comparison, highlighting whether each item in the original set was preserved, simplified, or omitted in the minimized version.

Interpretation of Minimization Results. The minimized scenario-safety property set successfully resolves many original safety violations (reducing from the initial 4 violations). It strategically simplifies non-safety-critical elements, clarifies previously ambiguous wording, and explicitly enhances critical safety conditions. Although some original scenario details are intentionally omitted or adjusted, this is justified explicitly by their irrelevance to safety constraints. Thus, the minimized set effectively demonstrates TAPAssure's optimization ca-

Table 5: Comparison of Original and Minimized Scenario-Safety Property Sets

Type	Original Set (Scenario 3 / SP3)	Minimized Set (Inferred)
Scenario 1	Use Motion Sensor to trigger Virtual Light and Virtual A/C 1 when entering the bedroom.	Motion sensor in bedroom triggers `Virtual_Light_bedroom`. A/C 1 logic preserved but not triggered explicitly.
Scenario 2	At sunrise, turn off Virtual Fan 1 and Virtual Light in the kitchen.	`Virtual_Fan1` turns off at sunrise. Kitchen light omitted.
Scenario 3	Turn on Virtual TV 2 and set Virtual Fan 5 to low when 'movie mode' is activated.	Movie mode activates `Virtual_TV2` and sets `Virtual_Fan5` to `low`; fan automatically turns off after 60 minutes.
Scenario 4	Turn off Virtual A/C 3 when all doors are closed and no motion is detected in the house.	`Virtual_AC3` turns off if `doors_closed` and `no_motion` are true.
Scenario 5	Turn on Virtual Fridge 1 and Virtual Light when motion is detected in the dining room.	Motion in dining room triggers `Virtual_Fridge1`. Dining room light omitted.
Scenario 6	If Motion Sensor detects temperature above 80°F, activate Virtual A/C 2 and Virtual Fan 4.	`Virtual_AC2` is deactivated if temp $>$ 78°F. `Virtual_Fan4` turns `low` for temp $>$ 80°F, `medium` for $>$ 85°F.
Scenario 7	Turn on Virtual TV 1 and mute it if no movement is detected in the living room for 5 minutes.	If `no_motion` and `time > 5`, `Virtual_TV1` is turned on and muted conditionally.
Scenario 8	Turn off Virtual Fan 2 at 10 PM every night to conserve energy.	`Virtual_Fan2` turns off when `time = 22`.
SP 1	Virtual A/C 3 must deactivate if temperature drops below 70°F.	Same property enforced via `room_temperature_below_70`.
SP 2	Virtual A/C 2 must not exceed 78°F.	Same logic enforced using `room_temperature_above_78`.
SP 3	Virtual Fan 5 must not remain on for more than 1 hour without activity detection.	Fan 5 auto-off after 60 minutes. Activity detection condition abstracted.
SP 4	Virtual Fan 4 must operate at medium if temperature exceeds 85°F.	Same rule enforced directly. Additional case added for temp $>$ 80°F as `low`.

pability, prioritizing safety compliance and simplified formal verification over exact replication of original scenario logic.

Table 5 presents a comparison between the original and minimized scenario-safety property sets. The changes introduced during the minimization process reflect several recurring themes, primarily aimed at simplifying the automation rules while retaining or enhancing safety-relevant logic.

In several cases (e.g., Scenario 1 and Scenario 5), actions unrelated to the core safety requirement were intentionally omitted. For instance, the minimization removed the activation of Virtual A/C 1 and the dining room light, focusing only on the essential device (Virtual Light or Virtual Fridge 1) to streamline the automation and reduce unnecessary device usage. Similarly, in Scenario 2, the kitchen light was excluded, as it did not directly impact the safety property under consideration.

Other changes were made to enhance clarity and ensure explicit enforcement of safety conditions. For example, Scenario 3 introduces an explicit timer for automatically turning off Virtual TV 2 and Virtual Fan 5 after 60 minutes of movie mode activation, providing a clear safety mechanism for energy management. Scenario 6 further refines environmental controls by specifying distinct activation and deactivation thresholds for Virtual A/C 2 and defining different operating levels for Virtual Fan 4 based on temperature.

In some scenarios and safety properties (e.g., Scenario 4, Scenario 7, Scenario 8, and Safety 1), the logic was preserved without modification, indicating that the original rules were already optimal or sufficiently precise with respect to

the safety objectives. Finally, certain simplifications involved replacing complex activity-based triggers with straightforward timers, as seen in Safety 3, where the fan is now turned off after a fixed duration, omitting the need for activity detection. Overall, these minimization decisions collectively reduce complexity, improve the interpretability of the rules, and enhance the enforcement of safety properties in the automated smart home environment.

This comparison confirms that TAPAssure's model refinement does not compromise safety logic and successfully eliminates redundant or unused transitions for streamlined verification.

6 Related Work and Comparison

TAPAssure sits at the intersection of two research directions: the use of Large Language Models (LLMs) to simplify smart home automation and the application of formal methods to ensure the safety of IoT systems. Prior work has generally emphasized one of these aspects at the expense of the other. TAPAssure is, to our knowledge, the first system to unify natural language–based rule generation with proactive, design-time verification and refinement.

6.1 LLMs for Smart Home Automation

Recent efforts have shown how LLMs can improve usability by translating high-level goals into Trigger-Action Programming (TAP) rules. Sasha [9] and Aware Auto [15] exemplify this trend, converting goals such as "get ready for a party" [8] into coordinated device automations. These approaches lower the barrier for non-technical users to create sophisticated routines. However, they provide no mechanisms for detecting conflicts, unsafe conditions, or policy violations, creating what we term the *usability-correctness gap*. As non-experts author increasingly complex automations, the absence of verification increases the likelihood of unsafe or inconsistent behaviors.

6.2 Formal Verification for IoT Safety

In parallel, research on IoT safety has focused on verification but typically assumes structured, expert-authored code. Soteria [10], for example, statically analyzes IoT apps by translating them into an intermediate representation and applying model checking to detect violations such as "the door must remain locked when the user is away." While effective in this setting, Soteria presupposes formalized source code and therefore cannot be applied to natural-language specifications.

Other approaches enforce safety reactively at runtime. IoTGuard [4] instruments applications to block unsafe actions during execution, while IOTMEDIATOR [18] resolves conflicting commands in multi-resident environments. These systems follow a "fail-safe" philosophy, but blocking or altering actions at runtime can result in degraded or unpredictable user experiences. TAPAssure instead adopts a proactive, *correct-by-construction* paradigm: by integrating model

checking into the generation process, it ensures that unsafe rules are removed before deployment.

6.3 Comparative Analysis

In summary, existing work either emphasizes user-friendly generation without guarantees (Sasha, Aware Auto) or strong verification detached from usability (Soteria, IoTGuard, IOTMEDIATOR). TAPAssure brings these strands together by refining natural-language–generated rules through model checking at design time, closing the usability-correctness gap. Table 6 positions TAPAssure relative to these approaches.

Table 6: Methodological Comparison of Automation and Verification Approaches

Approach	**Core Task**	**Verification**	**Stage**	**Feedback**
Sasha [9]	LLM Generation	None	Design-Time	None
Aware Auto [15]	LLM Generation	None	Design-Time	None
Soteria [10]	Conflict Detection	Model Checking	Pre-Deployment	Static Reports
IoTGuard [4]	Safety Enforcement	Dynamic Monitoring	Runtime	Blocking
IOTMEDIATOR [18]	Conflict Resolution	Mediation	Runtime	Patching
TAPAssure (Ours)	**LLM Gen + Refinement**	**Model Checking (LTL)**	**Design-Time**	**Counterexamples**

7 Conclusion

In this paper, we introduced TAPAssure, a framework integrating LLMs with formal verification to improve the safety and reliability of TAP rules in smart homes. Our comprehensive evaluation using Meta Llama 3.3 and Qwen2.5 demonstrated that TAPAssure effectively identifies and resolves both syntax errors and safety violations in an autonomous fashion.

Our evaluation highlighted distinct strengths of each LLM: Qwen2.5 showed superior efficiency in resolving syntax errors, while Meta Llama 3.3 demonstrated greater robustness in minimizing safety property violations, making it particularly suitable for safety-critical scenarios. By successfully bridging the intuitive capabilities of LLMs with the rigor of formal verification, TAPAssure offers a robust pathway toward creating smart home automation that is not only powerful but also verifiably safe.

References

1. Alam, M., Reaz, M.B.I., Ali, M.A.R.: Automation in smart homes. Handbook of green information and communication systems pp. 329–348 (2012)
2. Bommasani, R., Hudson, D.A., Adeli, E., Altman, R., Arora, S., von Arx, S., et al.: On the opportunities and risks of foundation models. arXiv preprint arXiv:2108.07258 (2021)
3. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. arXiv preprint arXiv:2005.14165 (2020)

4. Celik, Z.B., Tan, G., McDaniel, P.: Iotguard: Dynamic enforcement of security and safety policy in commodity iot. In: Proceedings of the Network and Distributed System Security Symposium (NDSS) (2019)
5. Cimatti, A., Clarke, E., Giunchiglia, E., Roveri, M., Sebastiani, R., Tacchella, A.: Nusmv 2: An opensource tool for symbolic model checking. International Conference on Computer Aided Verification pp. 359–364 (2002)
6. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (1999)
7. Huang, J., Cakmak, S., Miller, R.: Supporting trigger-action programming in the smart home. In: Proceedings of the ACM CHI Conference on Human Factors in Computing Systems. pp. 803–812 (2013)
8. King, E., Yu, H., Lee, S., Julien, C.: " get ready for a party": Exploring smarter smart spaces with help from large language models. arXiv preprint arXiv:2303.14143 (2023)
9. King, E., Yu, H., Lee, S., Julien, C.: Sasha: creative goal-oriented reasoning in smart homes with large language models. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies **8**(1), 1–38 (2024)
10. Lee, Y., Aafer, Y., Zhi, W., Huang, J., Zhang, Y., Zhang, X., Lee, W.: Soteria: Automated iot safety and security analysis. In: USENIX Annual Technical Conference (USENIX ATC). pp. 147–158 (2018)
11. Mendoza, P., Hodges, S., DeJonghe, D., Cakmak, S.: Flexible conflict detection and resolution for smart home systems. IEEE Transactions on Smart Grid **6**(1), 107–116 (2015)
12. Perera, C., Zaslavsky, A., Christen, P., Georgakopoulos, D.: Emerging trends, challenges, and opportunities in internet of things: Applications. arXiv preprint arXiv:1401.4977 (2014)
13. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners. OpenAI Blog **1**(8), 9 (2019)
14. Rivkin, D., Hogan, F., Feriani, A., Konar, A., Sigal, A., Liu, S., Dudek, G.: Sage: Smart home agent with grounded execution. arXiv preprint arXiv:2311.00772 (2023)
15. Shi, Y., Liu, X., Yu, C., Yang, T., Gao, C., Liang, C., Shi, Y.: Bridging the gap between natural user expression with complex automation programming in smart homes (2024), https://arxiv.org/abs/2408.12687
16. Stojkoska, B.L.R., Trivodaliev, K.V.: A review of internet of things for smart home: Challenges and solutions. Journal of Cleaner Production **140**, 1454–1464 (2017)
17. Ur, B., McManus, E., Pak, K., Sciolla, J., Littman, M.L.: Practical trigger-action programming in the smart home. In: Proceedings of the ACM CHI Conference on Human Factors in Computing Systems. pp. 803–812 (2014)
18. Xiao, Q., Wang, Y., Li, R., Chen, W., Zhang, X.: Iotmediator: Resolving conflicts in smart home iot applications. In: Proceedings of the ACM International Conference on Information Processing in Sensor Networks (IPSN). pp. 250–261. ACM (2020). https://doi.org/10.1145/3372224.3380894
19. Zellers, R., Holtzman, A., Rashkin, H., Bisk, Y., Farhadi, A., Roesner, F., et al.: Defending against neural fake news. In: Advances in Neural Information Processing Systems. vol. 32, pp. 9054–9065 (2019)