Grammar-Constrained Refinement of Safety Operational Rules Using Language in the Loop: What Could Go Wrong

Anonymous Author(s)

Abstract

Safety specifications in cyber-physical systems (CPS) capture the operational conditions that the system shall satisfy to operate safely within its intended environment. As environmental conditions evolve, operational rules that express acceptance criteria must be continuously refined to preserve alignment with observed system behavior. These rules, often expressed as logical or quantitative constraints on system behavior, are validated through simulation-based testing but may become inconsistent as new operational conditions or boundary behaviors emerge. We ask: Can grammar-constrained refinement loop using large language models (LLMs) automatically refine inconsistent operational rules while preserving syntactic correctness and semantic validity, and what safety concerns emerge from language-in-the-loop refinement? We introduce a framework that combines counterfactual reasoning with a grammar-constrained refinement loop to refine operational rules, aligning them with the observed system behavior. Applied to an autonomous driving control system, our grammar-constrained refinement loop, supported by counterfactual evidence, successfully resolved the inconsistencies in an operational rule inferred by a conventional baseline while remaining grammar compliant. An empirical LLM variant study further revealed model dependent refinement quality and safety lessons, which motivate parser-based grammar enforcement, stronger automated validation and broader evaluation in future work.

CCS Concepts

• **Do Not Use This Code → Generate the Correct Terms for Your Paper**; *Generate the Correct Terms for Your Paper*; Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper.

Keywords

Safety Operational Rule Refinement, Large Language Models, Cyber-Physical Systems

ACM Reference Format:

Anonymous Author(s). 2018. Grammar-Constrained Refinement of Safety Operational Rules Using Language in the Loop: What Could Go Wrong. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

Conference acronym 'XX, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Safety specifications for CPS rely on operational rules that translate safety requirements into testable conditions over system execution used for verification and validation. In autonomous driving, these rules are scoped by the Operational Design Domain (ODD) [5, 7] and guided by ISO 26262 [4] and ISO/PAS 21448 (SOTIF) [12]. They are typically expressed as logical or quantitative constraints over context and system variables, and evaluated through simulation-based testing [19] to ensure safe operation within the declared operational boundaries. However, as the operating context evolves and the system behavior shifts, keeping operational rules valid and aligned with observed executions becomes a recurring challenge.

Mining operational rules from observed traces has been commonly performed by learning linear- (LTL) and signal-temporal logic (STL) properties [17, 18], using machine learning and specification mining methods [9, 14], typically constrained by templates rather than a domain specific grammar. Other works use surrogate models [10, 11, 22] and genetic programming [6] as interpretable methods to infer and correct STL properties, or apply parameter mining and falsification [2, 8, 13] to tune bounds that make observed behaviors satisfy a fixed specification. No prior work is designed to automatically refine an existing inconsistent operational rule under a domain specific grammar. Recent work has explored using LLMs to support the construction and maintenance of safety specification artifacts. Nouri et al. [16] propose a prompt based pipeline for an automotive SafetyOps workflow that generates safety requirements and then checks the resulting rule set for redundancy, contradictions, and other quality issues. Li et al. [15] use LLMs to generate LTL specifications under safety restrictions, then iteratively refine candidate formulas using language inclusion checks and counterexamples. Both works commonly combine LLM generation with automated validation to reduce model variability and improve rule validity. However, to our knowledge, no prior work uses grammar-constrained LLMs to refine an existing inconsistent specification to align with observed system behavior. Moreover, existing validation in LLM based specification synthesis typically targets syntactic or format compliance, rather than consistency with runtime outcomes.

To address these limitations, we propose an approach for refining operational rules to resolve inconsistencies between rule verdicts and observed system behavior. Rather than constructing a formal proof of compliance, our approach is guided by observed counterfactual evidence. The core novelty is combining counterfactual analysis with a domain-specific grammar-constrained LLM to synthesize minimal refinements of operational rules while preserving syntactic correctness and semantic validity. This paper provides two main contributions: (i) We introduce a rule refinement framework that combines counterfactual reasoning to localize inconsistency boundaries between operational rules and observed behavior, and grammar-constrained LLMs to generate interpretable, syntactically

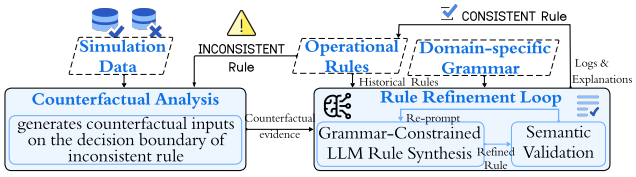

Figure 1: Operational Rule Refinement Approach Overview.

valid refinements. The refinement loop validates candidate rules through semantic validation; (ii) We present an initial empirical study on an autonomous driving subsystem (ADS) that evaluates the effectiveness of our approach in resolving inconsistencies, motivated by a conventional GP-based inference method. We further assess the model-dependent language-in-the-loop refinement quality across multiple LLM variants and retrieve safety lessons.

2 Operational Rule Refinement Framework

To address inconsistencies between operational rules and observed system behavior, we introduce a grammar-constrained rule refinement framework that leverages counterfactual reasoning combined with grammar-constrained LLM to produce interpretable, syntactically and semantically valid refinements of operational rules. The framework can be applied as a corrective mechanism for outdated operational rules within safety specifications that have become inconsistent with evolving system behavior. As a running example, we consider an autonomous driving controller with three scenario features; the ego vehicle speed (`ego_speed`), the distance to the front vehicle (`dist_front`), and the lateral offset within the lane (`lane_offset`).

Operational Rules and Semantics. Let $X \subseteq \mathbb{R}^d$ denote the input domain of the system under test (SUT), where each input vector $x = [i_1, \ldots, i_d]$ represents input variables. The observed outcome (the system execution label) of x is $y \in \{\text{Pass}, \text{Fail}\}$. An *operational rule* is a predicate $r : X \rightarrow \{\text{true}, \text{false}\}$ that constrains the operating region of the system. Here, true and false denote the logical evaluation of the rule on a given input. Specifically, $r(x) = \text{true}$ means that the rule holds for x, i.e., the input is considered valid within the admissible operating region, while $r(x) = \text{false}$ indicates that the rule is violated, i.e., the input lies outside that region. For example, an operational rule r_1 : `(dist_front < 5.0) ∧ (ego_speed > 0)` specifies a logical predicate over `ego_speed` and `dist_front` features. Given an input vector x_1 = `(ego_speed=8.0, dist_front=4.2, lane_offset=0.1)`, $r_1(x_1) = \text{true}$.

We consider two rule sets, R_{Pass} and R_{Fail}, encoding conditions that imply *Pass* or *Fail*, respectively. A rule is denoted *consistent* if it evaluates to true and matches the observed outcome y of x, *inconsistent* if it evaluates to true and contradicts y, and *inconclusive* if it evaluates to false (no definitive outcome can be derived). In this paper, we refine inconsistent rules, since they expose discrepancies between operational rules and observed system behavior. In our running example, consider $r_1 \in R_{\text{Pass}}$ as a pass rule and the observed outcome of x_1 is $y_1 = \textit{Fail}$, which makes r_1 inconsistent.

Each operational rule is expressed using a domain-specific grammar that defines the syntactic space of valid predicates over the system inputs. In general, the structure of the grammar is guided by domain knowledge, which determines the form and semantics of admissible expressions. In the automotive domain, operational rules for cyber-physical systems are typically expressed as arithmetic and relational constraints over configuration parameters and input variables. While it may vary across domains, in this paper we adopt the grammar G introduced in prior work [6] to express environmental assumptions for cyber-physical systems, as it is particularly suited to the automotive context studied here. Each operational rule is a hierarchical logical formula built from disjunctions ($Disj$), conjunctions ($Conj$), and relational predicates (Rel) over arithmetic expressions (Exp). A rule is composed of one or more disjunctive clauses, each representing an alternative valid operating condition. Within each clause, conjunctions combine relational predicates that must hold simultaneously. Each predicate Rel compares two expressions Exp using operators $\{<, \leq, >, \geq, =, \neq\}$, where expressions are formed from arithmetic operations over constants ($const$) and input variables (var).

```
Rule ::= Disj
Disj ::= Disj ∨ Conj | Conj
Conj ::= Conj ∧ Rel | Rel
Rel ::= Exp rop Exp
rop ::=<|≤|>|≥|=|≠
Exp ::= Exp aop Exp | const | var
aop ::= + | − | ∗ | /
```

Problem Statement. Given the rule sets R_{Pass} and R_{Fail}, and a labeled input set $T = \{(x_i, y_i)\}$, the objective is to derive refined rule sets as changes in the components of existing rules, such as constants, input variables, relational operators, or logical connectors, so that they satisfy the following conditions: (i) Maintain semantic consistency with the grammar; (ii) Reduce the number of inconsistent rules; (iii) Preserve previously consistent rules; (iv) Eliminate contradictory rules.

Approach Overview. The grammar-constrained rule refinement framework operates on simulation or testing data and existing operational rules to resolve inconsistencies between operational rules and observed system behavior. In this work, we assume the presence of a safety operational rule set evaluated for inconsistencies using a manual review process or an automated consistency checking mechanism. We also assume that the rules are expressed in temporal logic according to a predefined grammar that constrains the admissible structure of rules (e.g., grammar G). Figure 1 shows the overview of our approach. The framework takes as input (1) an operational rule set $\mathcal{R}$ containing at least one inconsistent rule r, (2) a grammar specification G (3) a labeled execution dataset $\mathcal{D}$ of simulation or test cases. Each test case contains an input vector x of input values and an observed outcome $y \in \textit{Pass}, \textit{Fail}$ with respect to a given safety specification. For example, the input vector x_1, r_1 assigns the x_1 a *Pass* verdict. However, the observed system outcome for x_1 is $y_1 = \textit{Fail}$, which makes r_1 inconsistent. The approach then proceeds through the following steps:

1) Counterfactual Analysis: This step takes as input an inconsistent operational rule r and a labeled simulation dataset $\mathcal{D}$ containing test inputs and observed outcomes. It produces a counterfactual evidence file E by generating counterfactual inputs for inputs that expose inconsistencies in r. Concretely, for each inconsistent case $(x, y) \in \mathcal{D}$, we search for a minimally perturbed input x' such that the outcome flips, yielding evidence of a local decision boundary for the predicates in r. For our running example, the

Counterfactual Analysis step generates a counterfactual input x'_1 = `(ego_speed=8.0, dist_front=4.0, lane_offset=0.1)` with label y' = *Pass*, indicating that a small change to `dist_front` is sufficient to restore the *Pass* verdict. For each counterfactual, we calculate the feature-wise perturbation as $\Delta = x' - x$. In our example, the perturbation is -0.2. The counterfactual x' is obtained through an L_1 minimal-change search [20] over the input space, which identifies the smallest modification that restores agreement between the rule verdict and the observed system behavior. Starting from x, the search incrementally expands the L_1 radius and evaluates modified feature assignments until it finds the first configuration x' that flips the outcome. The resulting evidence file E stores the dataset $\mathcal{D}$, the rule r, the paired inputs (x, x') together with their labels (y, y') and perturbation Δ.

Prompt template. *Input:* G; inconsistent rule r; historical rules; evidence. *Task:* Return a refined rule r' in the syntax of G and a short explanation. *Loop:* If r' uses out-of-vocabulary tokens or conflicts with historical rules, re-prompt with the failure summary and regenerate. *Format exemplar:* $(0 < ARG2 < 5) \wedge (ARG1 > 0) \ \vee \ (8 < ARG2 < 12)$

2) Rule Refinement Loop: This step takes the evidence E together with the grammar specification G, and historical consistent rules used for the semantic validation. First, a grammar-constrained LLM acts as a synthesis assistant to propose a candidate refinement of r. Through zero-shot instruction prompting with a reference format exemplar and constraint-based guidance, the prompt provides the paired boundary inputs (x, y) and (x', y') and the corresponding Δ from E and instructs the LLM to produce r', a refinement of r, with minimal, grammar-compliant predicate changes such as threshold adjustments, operator replacements, or selective addition or removal of conjuncts and disjuncts. The objective of the refinement is to restore consistency with the observed system behavior while preserving the rule's original semantics and interpretability and remaining within the grammar constraints. For our running example, the grammar-constrained refinement identifies the predicate in r_1 most responsible for the inconsistency. Since r_1 is a *Pass* rule that incorrectly labels x_1 as *Pass* while the observed outcome is *Fail*, the refinement makes the rule more restrictive to exclude the failing region. The LLM proposes tightening the `dist_front` condition and the candidate refinement yields $r_1^\star$: (`dist_front` < 4.1) $\wedge$ (`ego_speed` > 0). In addition to the objective, the prompt includes historical consistent rules and instructs the LLM to propose a candidate refinement that remains consistent with the existing rule set $\mathcal{R}$. If the candidate conflicts with historical rules or violates the allowed vocabulary, the prompt re-initializes the LLM refinement process. Before termination, each candidate refinement undergoes semantic validation on $\mathcal{D}$ to ensure that it restores consistency without introducing new inconsistencies. The refinement rule $r^\star$ is returned only if it passes this validation; otherwise, the loop continues. The grammar conformance is enforced through prompt-level constraints, including a whitelist of allowed tokens and operators (e.g., $\{\wedge, \vee, <, >, \leq, \geq\}$ and domain feature names. The full prompt with all examples is provided in our shared package [1]. The output of the framework is (i) the refinement rule $r^\star$ expressed in the syntax of the grammar, (ii) the change log summarizing the changes applied to r, and (iii) a short explanation of how the refinement addresses the observed inconsistencies.

3 Experimental Exploration

In this section, we conduct a first experimental exploration on our running case-study system, an autonomous driving control system [3] that implements autopilot control for both lateral and longitudinal guidance in lane-following scenarios. The input vector includes numeric signals from both the ego-vehicle and the environment such as speed, steering angle, road curvature, weather, and obstacle distance. The system processes the inputs to compute throttle and steering adjustments that ensure lane keeping. The safety requirement is that *the vehicle maintains its lane within admissible bounds*. We randomly generate 198 inputs, execute one run per input, and label each run *Pass* if the requirement holds, and *Fail* otherwise. The results are available in our shared package [1]. To evaluate our framework, we report five evaluation metrics:

(1) Decisiveness gain (DG) measures how consistently the rule's verdict matches the actual simulation outcomes. We compute $1 - \frac{N_{mismatch}}{N}$, where $N_{mismatch}$ denotes the number of runs for which the rule does not match the ground truth verdict, and total runs N.

(2) *Semantic validity (SV)* measures whether the refined rule stays grounded in the provided ODD and current operational rules. We use expert ratings to mark a predicate as *invalid* if it introduces an out-of-range bound, a variable not present in the input vector, or an unsupported operator that violates the grammar or data constraints. We compute $1 - \frac{N_{invalid}}{N_{pred}}$, where $N_{invalid}$ is the number of invalid predicates and N_{pred} is the total number of predicates.

(3) *Interpretability (I)* measures whether the LLM explanation is easy to follow and justifies all refinements. We use expert ratings: 1.0 (Excellent) if the explanation (i) identifies and isolates inconsistencies in the original rule, (ii) presents the refined rule, and (iii) clearly justifies each major change. We assign 0.7 to 0.8 (High) when the explanation may not cover every edit but provides specific, well matched justifications for the main refinements. We assign 0.5 (Low) when the explanation remains generic and does not justify the specific changes that were needed for most predicates.

(4) *Grammar compliance (GC)* measures the structural correctness of the refined rule, i.e., whether it preserves the grammar's disjunctive and conjunctive structure and follows the format exemplar in the prompt. We tokenize the rule (e.g., `'operator'`, `ARG`, `Value`, '(', ')') and count the structural violations as tokens that break the grammar. We then compute $1 - \frac{N_{viol}}{N_{tok}}$, where N_{viol} is the number of violating tokens and N_{tok} is the total number of tokens.

(5) *Change minimality (CM)* measures how conservatively the LLM refines the rule while preserving the original constraints. We use expert ratings: 1.0 (Optimal) for pruning to the logical core with minimal edits and the same variables (e.g., `ARG2 > 3 AND ARG2 > 5` $\rightarrow$ `ARG2 > 5`). We assign 0.7 to 0.8 (Conservative) for moderate cleanup or added complementary operators (e.g., adding an upper bound to `ARG2 > 5`) without changing variables. We assign 0.4 to 0.5 (Over constrained) when the rule adds unjustified bounds that narrow its scope (e.g., `ARG1 > 0` $\rightarrow$ `1 < ARG1 < 2`). We assign 0.0 to 0.3 (Low) for extensive rewrites where most predicates change and the rule logic shifts substantially.

Conventional baseline. As a motivating experiment, we consider a GP baseline. We use an operational pass rule inferred by GP from prior assertion inference work [6, 11], which reports inconsistencies quantified using accuracy and misprediction metrics across systems

Table 1: Evaluation metric scores per LLM variant.

LLM	GC	SV	I	CM	LLM	GC	SV	I	CM
GPT5 Thinking	**1.0**	**1.0**	0.5	**0.9**	Qwen3 Max	**1.0**	0.8	0.7	0.7
GPT5 Instant	**1.0**	0.7	0.7	0.4	DeepSeek DeepThinking	0.5	0.5	0.7	0.3
Gemini Flash 2.5	**1.0**	**1.0**	0.8	0.7	DeepSeek Normal	0.7	**1.0**	0.5	0.8
Gemini Pro 2.5	**1.0**	0.4	0.7	0.2	Claude Sonnet 4.5	**1.0**	0.2	**1.0**	0.0

and requirements. We evaluate this rule on the same dataset of $N = 198$ labeled runs and observe $N_{mismatch} = 27$ runs, yielding $DG = 0.86$. We apply our approach to the inconsistent rule using 8 LLM variants [21], GPT-5 (Thinking and Instant), Claude Sonnet 4.5, DeepSeek (DeepThinking and Normal), Qwen3 Max, and Gemini 1.5 (Pro and Flash), and record under the same grammar G, prompt template the refined rules with change log and explanation. We then compute decisiveness on the 198 labeled runs for all variants. All variants reduce mismatches to 0 ($DG = 1.0$), corresponding to a gain of +0.14 over the GP baseline. This indicates improved alignment between operational rule verdicts and observed system behavior, motivating broader baselines in future work.

LLM Variant Study. Given the promising preliminary observations, we assess how refinement quality varies with model choice in our language-in-the-loop setting and we retrieve lessons learned about LLM use in safety operational context. We analyze the refined rules, change logs, and explanations generated by our approach configured with the 8 LLM variants.

Table 1 reports four metric scores for each LLM. GPT5 Thinking mode shows the strongest combination of semantic validity and minimality while staying fully grammar compliant. Gemini Flash 2.5 and Qwen3 also remain grammar compliant with relatively strong semantic validity. In contrast, Gemini Pro 2.5 and Claude Sonnet 4.5 are grammar compliant and fairly interpretable. DeepSeek exhibits mixed behavior, with the Normal variant achieving high semantic validity and minimality but lower interpretability, while DeepThinking shows weaker grammar compliance and more extensive changes. We retrieve the following lessons:

Lesson 1: Even with the same grammar, prompt, and simulation data, different LLMs vary in outcome. Some rules look correct but include formatting that breaks the expected structure and variable naming. These should be treated as unsafe to apply. For example, DeepSeek DeepThinking returned the rule inside a markdown code block and wrapped the rule with an extra outer list , which violates the structure. The expected rule is `[('greater_than_func','ARG1','0')]`..., but the model returned `[[('greater_than_func','ARG1','0')]...]`.

Lesson 2: LLMs tend to increase apparent safety by tightening bounds and occasionally adding extra constraints, but this can over constrain the rule in a conservative way that is not correctly grounded in the provided ODD, yielding many unnecessary nominal restrictions. For example, a refined rule may turn a simple threshold into a tight range, changing `0 < ARG1` to `0 < ARG1 < 8`. This can look safer, yet it may be unsupported by the provided ODD and therefore unjustified. In safety critical use, validity checks against the ODD should be adopted to flag new or tightened bounds that are not semantically valid, and iterative feedback loop should be triggered whenever the model makes large threshold shifts, violates the grammar, or introduces new variables, even if the explanation appears convincing.

Lesson 3: There is a link between how much a model changes the rule and how easy its output is to interpret. When changes are broader, the model explicitly critiques the original inconsistencies and justifies each major change. For example GPT5 Thinking mode makes small, targeted refinements, reducing the rule to `ARG1 > 0` or `ARG2 > 3`, and its justification only focuses on redundancy removal and fixing a malformed predicate. In contrast, Gemini Pro 2.5 introduces new conjunctive constraints and new variables (`ARG3`) alongside operator changes, and it provides a structured rationale for each addition, for example explaining the shift to >= at the boundary and motivating the new < caps.

This initial study shows that grammar-constrained, counterfactual-guided refinement can eliminate baseline inconsistencies, while the LLM variant study reveals model dependent quality and safety trade-offs that illustrate *"what can go wrong"*. Although grammar guidance supports syntactic correctness, safety guarantees require additional verification feedback loops, motivating stronger semantic validation in future work. The future work is organized as follows. We will strengthen grammar enforcement by adopting a strongly-typed rule generator and a parser-based acceptance mechanism that rejects any output violating the grammar or structural format. We will also reinforce semantic validation by moving beyond a static regression test suite to simulation-based falsification and robustness testing, to ensure that refined rules are consistent over the broader input space and the stated ODD. This mechanism will flag unjustified or tightened constraints and expose unsafe overfitting. To counter overly conservative refinements, we will incorporate change minimality into an independent selection mechanism so edits are penalized even when decisiveness is high. For evaluation, we will broaden baselines and study subjects to assess whether grammar constrained LLM refinement offers clear benefits over established interpretable rule learning and specification mining. Beyond the GP baseline [6], we will compare against decision trees and decision rules [10, 11, 22], and against temporal specification mining methods such as Texada [14] and TeLEx [9], contrasting grammar guided refinement with template and structure guided approaches. We will then scale experiments across multiple ADS subsystems, requirements, and ODDs, and study how expanded grammars, prompting strategies, and validation mechanisms affect refinement quality and safety risk in operational settings.

4 Conclusion

Safety operational rules can lose alignment with observed system behavior as systems and operating contexts evolve. This paper introduced a rule refinement framework that uses counterfactual reasoning and a grammar-constrained LLM refinement loop to produce interpretable refinements that are syntactically correct and semantically valid. An initial study on an autonomous driving subsystem showed that our loop eliminates inconsistencies produced by the selected conventional method, with +0.14 decisiveness. An LLM variant study further exposed model dependent quality and safety trade offs including syntactic violations and over constraining operational rule refinements that risk overfitting to the test data. These findings highlight what can go wrong even under grammar guidance, motivating stronger grammar enforcement, more rigorous semantic validation, and broader evaluation in future work.

References

[1] Anonymous. 2026. REVISED-Shared Package for LLM-Guided Rule Refinement Framework. https://figshare.com/s/ddf2a7040719e019b52b. Anonymous dataset link provided for double-blind review.

[2] Eugene Asarin, Alexandre Donzé, Oded Maler, and Dejan Nickovic. 2011. Parametric identification of temporal properties. In *International Conference on Runtime Verification*. Springer, 147–160.

[3] Matteo Biagiola and Stefan Klikovits. 2024. SBFT Tool Competition 2024 - Cyber-Physical Systems Track. In *Proceedings of the 17th ACM/IEEE International Workshop on Search-Based and Fuzz Testing, SBFT 2024, Lisbon, Portugal, 14 April 2024*. ACM, 33–36. doi:10.1145/3643659.3643932

[4] Rami Debouk. 2019. Overview of the second edition of ISO 26262: Functional safety—Road vehicles. *Journal of System Safety* 55, 1 (2019), 13–21.

[5] Laura Fraade-Blanar, Marjory S Blumenthal, James M Anderson, and Nidhi Kalra. 2018. *Measuring automated vehicle safety: Forging a framework*.

[6] Khouloud Gaaloul, Claudio Menghi, Shiva Nejati, Lionel C Briand, and Yago Isasi Parache. 2021. Combining genetic programming and model checking to generate environment assumptions. *IEEE Transactions on Software Engineering* 48, 9 (2021), 3664–3685.

[7] Alfredo García, David Llopis-Castelló, and Francisco Javier Camacho-Torregrosa. 2022. From the vehicle-based concept of operational design domain to the road-based concept of operational road section. *Frontiers in Built Environment* 8 (2022), 901840.

[8] Bardh Hoxha, Adel Dokhanchi, and Georgios Fainekos. 2018. Mining parametric temporal logic properties in model-based design for cyber-physical systems. *International Journal on Software Tools for Technology Transfer* 20, 1 (2018), 79–93.

[9] Susmit Jha, Ashish Tiwari, Sanjit A Seshia, Tuhin Sahai, and Natarajan Shankar. 2019. TeLEx: learning signal temporal logic from positive examples using tightness metric. *Formal Methods in System Design* 54, 3 (2019), 364–387.

[10] Baharin A Jodat, Abhishek Chandar, Shiva Nejati, and Mehrdad Sabetzadeh. 2024. Test generation strategies for building failure models and explaining spurious failures. *ACM Transactions on Software Engineering and Methodology* 33, 4 (2024), 1–32.

[11] Baharin A Jodat, Khouloud Gaaloul, Mehrdad Sabetzadeh, and Shiva Nejati. 2025. Automated Test Oracles for Flaky Cyber-Physical System Simulators: Approach and Evaluation. *arXiv preprint arXiv:2508.20902* (2025).

[12] OM Kirovskii and VA Gorelov. 2019. Driver assistance systems: analysis, tests and the safety case. ISO 26262 and ISO PAS 21448. In *IOP Conference Series: Materials Science and Engineering*, Vol. 534. IOP Publishing, 012019.

[13] Panagiotis Kyriakis, Jyotirmoy V Deshmukh, and Paul Bogdan. 2019. Specification mining and robust design under uncertainty: A stochastic temporal logic approach. *ACM Transactions on Embedded Computing Systems (TECS)* 18, 5s (2019), 1–21.

[14] Caroline Lemieux, Dennis Park, and Ivan Beschastnikh. 2015. General LTL specification mining (T). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 81–92.

[15] Junle Li, Meiqi Tian, and Bingzhuo Zhong. 2025. Automatic Generation of Safety-compliant Linear Temporal Logic via Large Language Model: A Self-supervised Framework. *arXiv preprint arXiv:2503.15840* (2025).

[16] Ali Nouri, Beatriz Cabrero-Daniel, Fredrik Törner, Håkan Sivencrona, and Christian Berger. 2024. Engineering safety requirements for autonomous driving with large language models. In *2024 IEEE 32nd International Requirements Engineering Conference (RE)*. IEEE, 218–228.

[17] Amir Pnueli. 1977. The temporal logic of programs. In *18th annual symposium on foundations of computer science (sfcs 1977)*. ieee, 46–57.

[18] Nicholas Rescher and Alasdair Urquhart. 2012. *Temporal logic*. Vol. 3. Springer Science & Business Media.

[19] Cumhur Erkan Tuncali, Georgios Fainekos, Hisahiro Ito, and James Kapinski. 2018. Simulation-based adversarial test generation for autonomous vehicles with machine learning components. In *2018 IEEE intelligent vehicles symposium (IV)*. IEEE, 1555–1562.

[20] Sandra Wachter, Brent Mittelstadt, and Chris Russell. 2017. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harv. JL & Tech.* 31 (2017), 841.

[21] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223* 1, 2 (2023).

[22] Xiubin Zhu, Dan Wang, Witold Pedrycz, and Zhiwu Li. 2022. Fuzzy rule-based local surrogate models for black-box model explanation. *IEEE Transactions on Fuzzy Systems* 31, 6 (2022), 2056–2064.