CellSecInspector: Safeguarding Cellular Networks via Automated Security Analysis on Specifications

Ke Xie
ke.xie@usu.edu
Utah State University
Logan, USA

Xingyi Zhao
xingyi.zhao@usu.edu
Utah State University
Logan, USA

Min-Yue Chen
chenmi33@msu.edu
Michigan State University
East Lansing, USA

Yu-An Chen
cheny132@msu.edu
Michigan State University
East Lansing, USA

Yiwen Hu
huyiwen@umbc.edu
University of Maryland, Baltimore County
Baltimore, USA

Munshi Saifuzzaman
munshi.saifuzzaman@usu.edu
Utah State University
Logan, USA

Wen Li
awen.li@usu.edu
Utah State University
Logan, USA

Shuhan Yuan
shuhan.yuan@usu.edu
Utah State University
Logan, USA

Guan-Hua Tu
ghtu@msu.edu
Michigan State University
East Lansing, USA

Tian Xie
tian.xie@usu.edu
Utah State University
Logan, USA

Abstract

The complexity, interdependence, and rapid evolution of 3GPP specifications present fundamental challenges for ensuring the security of modern cellular networks. Manual reviews and existing automated approaches, which often depend on rule-based parsing or small sets of manually crafted security requirements, fail to capture deep semantic dependencies, cross-sentence/clause relationships, and evolving specification behaviors. In this work, we present *CellSecInspector*, an automated framework for security analysis of 3GPP specifications. *CellSecInspector* extracts structured state–condition–action (SCA) representations, models mobile network procedures with comprehensive function chains, systematically validates them against 9 foundational security properties under 4 adversarial scenarios, and automatically generates test cases. This end-to-end approach enables the automated discovery of vulnerabilities without relying on manually predefined security requirements or rules. Applying *CellSecInspector* to the well-studied 5G and 4G NAS and RRC specifications and selected sections of TS 23.501 and TS 24.229, it discovers 43 vulnerabilities, 7 of which are previously unreported. Our findings show that *CellSecInspector* is a scalable, adaptive, and effective solution to assess 3GPP specifications for safeguarding operational and next-generation cellular networks.

1 Introduction

Cellular networks are critical for modern global connectivity, with 5G and 4G forming the backbone of modern mobile communication systems. Technical specifications and architectures are standardized by the 3rd Generation Partnership Project (3GPP), an association of seven Organizational Partners, including ATIS in the U.S., ETSI in Europe, and CCSA in China. Thousands of specifications are published by 3GPP to ensure global interoperability across mobile networks and devices. Consequently, mobile equipment vendors and operators are required to ensure compliance with 3GPP specifications. However, this standardization introduces systemic risks: *design flaws in 3GPP specifications can propagate into all compliant networks and devices, creating widespread threats with global negative impact.* Moreover, the increasing complexity, frequent revisions, and evolving new generations of mobile networks continue to expand attack surfaces.

To mitigate those risks, researchers spend increasing effort to scrutinize cellular network specifications and services documented in the 3GPP specifications to identify design issues. However, the traditional manual analysis by experts is slow, error-prone, and infeasible at scale. State-of-the-art automated approaches primarily rely on rule-based approaches (e.g., dependency parsing [11, 20], and keyword-driven scanning (e.g., hazard indicators [20]), failing to capture the

deeper and implicit knowledge in 3GPP specifications. As a result, vulnerabilities in critical mobile network procedures continue to be exploited, enabling attacks such as downgrade attacks [12, 18, 20, 32, 34, 35, 37, 39, 40, 49]. These methods suffer from three fundamental limitations: (1) they lack deep insights into the logic, processes, and context of specification texts, particularly when dealing with cross-sentence/clause dependencies; (2) their security requirements are manually developed for specific procedures, which severely limits scalability across the thousands of 3GPP specifications; (3) their analysis pipelines require costly manual updates to remain synchronized with evolving 3GPP releases. These limitations highlight the urgent need for an intelligent, automated system capable of understanding 3GPP specifications and systematically assessing them.

In this study, we develop *CellSecInspector* (Cellular Network Security Inspector), an automated framework capable of understanding 3GPP specifications and systematically assessing them via foundational security properties. *CellSecInspector* models mobile networks in structured SCA (state-condition–action) representations from 3GPP specifications, and examines them with foundational security properties. By integrating specification analysis with automated test case generation, *CellSecInspector* delivers a scalable, adaptive, and future-proof solution for 3GPP specification security assessment. In this study, *CellSecInspector* analyzes the critical NAS and RRC layers for current operational 5G and 4G mobile networks as well as the selected sections of TS 23.501 and TS 24.229. *CellSecInspector* not only discovers 36 previously reported vulnerabilities but also reports 7 new vulnerabilities. We summarize our contributions as follows.

⋄**New Framework:** We developed *CellSecInspector*, a novel approach for automated security assessment of 3GPP specifications, advancing toward fully automated and scalable specification analysis for safeguarding not only operational 5G/4G cellular networks but also Next-G mobile networks.

⋄**New findings:** Applying *CellSecInspector* to 5G and 4G NAS and RRC specifications, as well as the selected sections of TS 23.501 and TS 24.229, we discovered 43 vulnerabilities, 7 of which were previously unreported. They are shown to have serious security consequences, demonstrating the urgent need for automated 3GPP specification analysis and the effectiveness of *CellSecInspector* in identifying known and previously undetected vulnerabilities.

2 Background

Non-Access Stratum (NAS) and Radio Resource Control (RRC): The NAS is a key component of the control plane in modern 5G and 4G cellular networks, managing mobility, session control, and security between user equipment (UE) and the core network. It ensures seamless session establishment, authentication, and mobility across different networks. The RRC operates within the Access Stratum (AS) of both 5G and 4G networks and manages signaling between the UE and the radio access network (RAN). It handles connection setup, maintenance, release, handovers, paging, measurement reporting, and security configuration.

Large Language Models (LLMs): LLMs have emerged as powerful systems trained on massive text corpora to perform a wide range of natural language processing tasks, including comprehension, reasoning, and generation. Modern LLMs, including ChatGPT [17], DeepSeek [13], have demonstrated impressive emergent abilities across diverse tasks, showing that LLMs can go beyond text processing to support domain-specific reasoning. This capability makes LLMs a promising foundation for analyzing complex 3GPP specifications, where precision, contextual understanding, and reasoning across specifications are critical.

3 Related Work

Given the scale and complexity of 3GPP specifications, NLP techniques have increasingly been applied for cellular network security analysis. *Atomic* [20] develops adversarial test cases from the 4G NAS specification. It relies heavily on keyword seeds (e.g., conditional words, risky operations) to identify relevant sentences to generate the test cases. *ConTester* [19] automates LTE NAS conformance test generation by constructing an Event Dependency Graph (EDG). Their scope is limited to generating test cases for assessing commercial mobile devices rather than analyzing design issues within 3GPP specifications. *CellularLint* [47] leverages a domain-adapted BERT [28] model to identify inconsistencies in 5G and 4G NAS and security-related specifications. Its emphasis is on sentence-level semantic conflicts, similar to natural language inference (NLI) tasks [16, 25], rather than systematic security analysis. *Hermes* [11] parses the specification via handcrafted grammars using constituency parsing and dependency parse trees to develop FSMs, which are further manually translated into SMV language [11, 23, 32, 34] for formal security analysis. It remains rule-intensive and lacks flexibility to adapt to evolving specifications. *ARCANE* [51] leverages large language models to assist specification understanding and combines it with passive model learning to automatically construct protocol models for model-based fuzzing. The generated models are then used to produce fuzzing inputs for testing 5G O-RAN implementations. However, ARCANE focuses on implementation bug discovery rather than systematic analysis of design issues in 3GPP specifications. In contrast, *CellSecInspector* provides a security analysis framework that aims to comprehensively comprehend 3GPP specifications and conduct systematic analysis on cellular network specifications. It can

be easily adapted to conduct security analysis on large-scale and evolving 3GPP specifications, providing an efficient and effective approach to complement traditional manual efforts. Appendix B compares these works.

4 Overview of *CellSecInspector*

4.1 Approach Skeleton

Figure 1 shows the overview of *CellSecInspector*, which can automatically comprehend complicated 3GPP specifications, model the mobile network functions using structured state-condition-action (SCA) representations, and examine whether the 3GPP-designed procedures can violate foundational security properties. Five primary modules of *CellSecInspector* are developed: *SpecAdaptation* (Specification Adaptation), *SCA Representation Extractor*, *Function Chain Builder*, *SecOracle* (Security Oracle), and *VulnTestGenerator* (Vulnerability Test Generator). *SpecAdaptation* adapts the core model *CellSecInspector-Core* to the cellular network domain knowledge using the collected and curated 3GPP specifications. *CellSecInspector-Core* is further used in later modules. Two components are involved to comprehensively model the complicated mobile network functions. *SCA Representation Extractor* parses and extracts the SCA nodes from 3GPP specifications and the internal domain knowledge learned via *SpecAdaptation*. *Function Chain Builder* connects scattered SCA nodes into function chains for comprehensively modeling the cellular network services specified in 3GPP specifications. Next, *SecOracle* conducts the security checking using the foundational security properties. Lastly, *VulnTestGenerator* automatically generates test cases for validating vulnerabilities in the real world.

4.2 Challenges and Insights

3GPP specifications are inherently complex and have a vast scope, complicating the development of a framework for comprehending the cellular networks governed by 3GPP specifications. For instance, NAS and RRC specifications span thousands of pages, with intricate interdependencies and frequent updates across releases. A single 5G NAS specification TS 24.501 contains over 981 pages [7]. Reading and understanding 3GPP specifications is a time-consuming challenge even for experts. We design and develop *CellSecInspector* based on a pretrained LLM. To enable *CellSecInspector* for discovering vulnerabilities from 3GPP specifications, we solve three main challenges.

C1: Lack of annotated resources for mobile network security research tasks. Different from common NLP (natural language processing) research tasks, such as text classification, that come with the thorough and well-annotated datasets, there are no well-known and easily accessible resources in the mobile network research community. Specifically, we need to model the complex mobile network procedures first to conduct the security analysis on mobile network functions and services. Unfortunately, there are no established and public annotation specifications or training resources in this domain-specific task.

⋄ **Insight1:** To address C1, we propose the *SCA Representation Extractor*, which is designed to extract information for modeling the mobile network functions and services from 3GPP specifications. Two approaches are developed. First, instead of constructing the traditional FSM (finite state machine) models, *SCA Representation Extractor* captures the self-contained transition of mobile network procedures in our proposed *SCA (State-Condition-Action)* structure that can compose the complex and completed function event in a single presentation. Second, we adopt the efficient few-shot In-Context Learning (ICL) to enable our system to extract SCA nodes without large-scale annotated datasets.

C2: Computational cost for linking SCA nodes at scale. After deriving scattered SCA nodes, these nodes must be linked to construct complete mobile network procedures and capture the comprehensive network status (e.g., whether the security context is established). However, linking SCA nodes at scale is challenging because different procedures described in SCA nodes may or may not share meaningful relationships. Moreover, to avoid redundancy, 3GPP specifications use cross-references between clauses and specifications, further complicating the linkage process. With thousands or more SCA nodes that can create millions of candidate connections in a quadratic manner, it imposes substantial computational overhead for linking SCA nodes at scale.

⋄ **Insight2:** To address C2, we introduce the *Function Chain Builder*, which integrates two approaches: Type I: Node-Informed Exhaustive Connection and Type II: Reference-Guided Connection. Type I compares all node pairs to identify three types of valid transitions: temporal connections that link nodes with identical end and start states, semantic connections that capture equivalent but textually different states, and causal connections where one node's state, condition, or action enables another. While Type I provides comprehensive coverage, its $O(n^2)$ complexity is costly as the number of nodes grows. To reduce this burden, Type II leverages the rich cross-references embedded in 3GPP specifications (e.g., '*as specified in subclause X.Y.Z*') to restrict the search space to relevant sections. Together, these strategies enable efficient and automated construction of function chains across 3GPP specifications.

C3: Manually crafted security requirements are impractical for security assessment on large-scale mobile network functions. State-of-the-art methods, such as *Hermes* [11] and *ConTester* [19], rely on a small set of manually

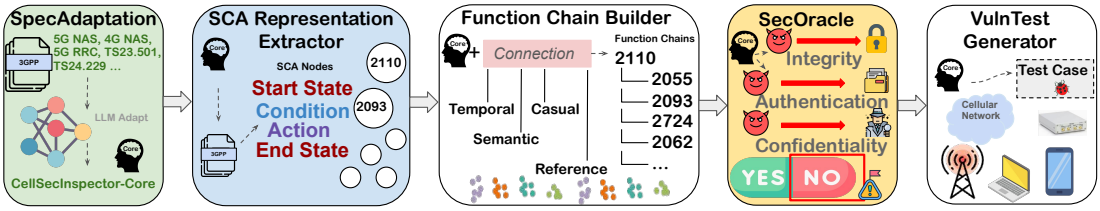

Figure 1: *CellSecInspector* Overview

crafted security requirements for specific procedures. For example, ConTester's security requirements S6 mandates that '*Except the messages listed below, no NAS signaling messages shall be processed by the receiving EMM entity in the UE forwarded to the ESM entity, ...*', which is the sentence directly from TS 24.301 subclause 4.4.4.2. When new NAS procedures are introduced in subsequent 3GPP releases, if these sentences are updated, their security requirements must be manually revised, limiting the scalability for the security assessments on large-scale mobile network functions. Similarly, approaches that build FSM models also need to manually translate FSM models into SMV language [11, 23, 32, 34] to enable formal security analysis, requiring significant expert efforts that are not practical given the complexity and breadth of modern mobile networks.

◇ **Insight3:** We develop the *SecOracle* that introduces 9 foundational security properties. *Unlike procedure-specific requirements, these properties are derived from classic and foundational security principles applicable across the entire mobile network stack.* Therefore, *SecOracle* can operate at large-scale cellular network procedures with broader security checkpoints. This equips *CellSecInspector* with the foundational ability for detecting previously unreported vulnerabilities.

5 Detailed Design

We next present *CellSecInspector* in detail.

5.1 *SpecAdaptation*

We develop the core reasoning engine, *CellSecInspector-Core*, on top of open-source LLMs, in *SpecAdaptation* to capture the cross-sentence and cross-clause relationships from the complex 3GPP specifications. To enable it to generalize and reason in 3GPP specifications, we expose *CellSecInspector-Core* to broad information to obtain cellular specification domain knowledge in the following two steps.

Cross-sentence/clause relation preserving data sanitation. To ensure *CellSecInspector-Core* can be exposed to the broad cellular network domain knowledge, we begin by collecting and curating 5G and 4G technical specifications from the 3GPP portal [5] as the corpus for training. To ensure consistency and clarity, we sanitize the specifications by (a) removing redundant spaces, extraneous line breaks, and leading symbols (e.g., dots, bullets, hyphens), (b) trimming trailing punctuation, (c) discarding empty parentheses and special unicode characters, and (d) removing tables and figures, which are usually explained or demonstrated in texts. Note that, references (e.g., 'as specified in subclause X.Y.Z') are preserved in the datasets, ensuring *CellSecInspector-Core* can learn the cross-sentence and cross-clause relations via explicit references that are neglected by state-of-the-art works [47].

Domain knowledge adaptation. A straightforward approach to expose standards knowledge to *CellSecInspector-Core* is to continually train (i.e., pretrain) the model on 3GPP specifications. While feasible in principle, this approach has two key limitations. First, pretraining a model is computationally expensive and difficult to maintain for the fast-evolving standards. 3GPP standards evolve frequently across different releases and versions. A continuously trained model can become out-of-date quickly unless training is performed repeatedly. Second, training an LLM on a narrow domain corpus (i.e., specifications) under limited compute and data diversity can degrade the model's general reasoning and instruction-following abilities [31, 36, 43]. For example, a continually pretrained model may exhibit catastrophic forgetting and spec-confusion behaviors, such as conflating procedures across different releases, hallucinating missing steps, or recurring boilerplate in specifications. These failures are harmful for standards analysis, where correctness hinges on precise conditions, message ordering, and release-specific normative requirements.

In our design, we leverage Retrieval-Augmented Generation (RAG) [41] to adapt *CellSecInspector-Core* to domain knowledge without modifying the base model. Specifically, *CellSecInspector-Core* retrieves relevant specification excerpts and conditions its generation on them, using the retrieved text as grounded evidence to complement its parametric knowledge. Concretely, given a query (e.g., message type, timer, or a security requirement), it searches a versioned corpus of 3GPP documents, which is prepared in the aforementioned step, and attaches the top-matched paragraphs (including section and release/version identifiers) to the model prompt. This retrieval step provides two technical benefits: (i) it ensures that *CellSecInspector-Core* uses the explicitly selected release/version text, and (ii) it reduces hallucination by requiring outputs to be justified by retrieved normative statements rather than relying solely on the model's memory.

Finally, *CellSecInspector-Core* obtains the 3GPP specification domain knowledge, and it will be further used in all modules.

5.2 *SCA Representation Extractor*

To address C1 and comprehensively model the complicated cellular network functions, we develop the *SCA (State-Condition-Action) Representation Extractor* that extracts the structured function representations in SCA nodes from 3GPP specifications. Each SCA node contains four fields: the start state, representing the initial state before the transition; the condition, denoting the triggering clause or prerequisite specified in the text; the action, describing the operation mandated by the specification upon satisfaction of the condition; and the end State, indicating the resulting state after the action is executed.

SCA node vs. FSM state. Different from the traditional finite state machine (FSM) that separates the states and transitions and stores the transition logic on edges between states, the SCA representation structure (i.e., node) aims to directly capture the self-contained transition, binding the logic of condition and action. This structure introduces two main advancements to maximize the utilization of the text processing capability of *CellSecInspector-Core*. First, it presents explicit transition semantics, including where the transition starts, when it is valid (condition), what happens (action), and where it ends. Each SCA node is one self-contained transition. The explicit viewpoint of a transition saves the step of merging all involved edges in the FSM when transiting from one state to another. That is, a FSM state may have several edges with its connected states, requiring extra efforts to identify how to transit to the next state. Second, it allows the complex conditions and actions, combining internal variables, timers, and status, can be described in a single node, which is valuable for modeling cellular network functions where the transitions may depend on complicated conditions and actions. *Note that SCA nodes can be easily transformed into the FSM states and edges. However, it is challenging to transform reversely.*

Few-shot Training with minimal dataset development overhead. To extract SCA nodes from 3GPP specifications, instead of developing a large dataset of many SCA nodes training cases with heavy manual engineering overhead, we leverage the efficient in-context learning (ICL) [17] to educate *CellSecInspector* without time-consuming fine-tuning to update model parameters. ICL works by embedding carefully selected demonstration examples and task-specific instructions directly into the model's input context, allowing the model to infer the desired output format. This method enables *CellSecInspector* to generalize to the SCA extraction task with a limited number of examples.

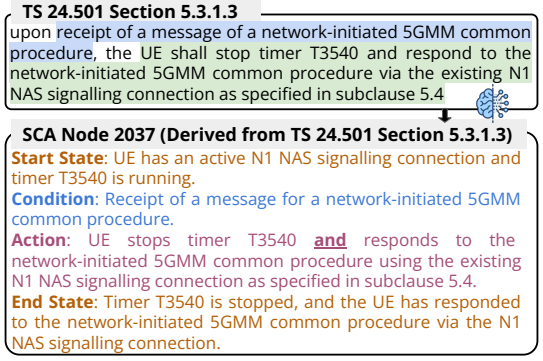

Figure 2: SCA representation for Node 2037

Specifically, we instruct *CellSecInspector* to extract structured specification events from 5G and 4G specifications by mapping each relevant sentence into the SCA node. Along with the instruction, a few manually developed SCA nodes are given for *CellSecInspector* to infer. From each specification, we develop two examples to demonstrate common specification patterns. Sentences that accurately describe function events are expected to populate all four fields.

If a sentence contains cross-references (e.g., 'as specified in subclause X.Y.Z'), the referenced content is retained verbatim in the most relevant field. For ambiguous or underspecified sentences, we employ best-effort inference. If no definition is found, the corresponding field is marked as 'Not explicitly defined' to ensure completeness of the extracted structure. Figure 6 in Appendix C shows an example ICL instruction.

SCA node example. Figure 2 presents an extracted SCA node from a short description in 5G NAS specification TS 24.501 by the following steps. *CellSecInspector* first parses the sentence structure using domain knowledge learned in *SpecAdaptation*. There was no explicit start state in this given example. *CellSecInspector* identifies the triggering condition by detecting the phrase upon receipt of a message, recognizing that the transition is activated when the UE receives a message for a network-initiated 5GMM common procedure. Next, *CellSecInspector* extracts the actions from the phrases '*shall stop timer T3540*' and '*respond to the procedure*,' which are organized as the structured expression that covers these two operations: stop T3540 **and** respond via the existing N1 NAS signalling connection as specified in subclause 5.4. From the derived condition and action, including '*via the existing N1 NAS signalling connection*' and '*timer T3540*', it is inferred that the UE already maintains an active N1 NAS signalling connection and T3540 is running before the event occurs, which are captured as the start state. Last, *CellSecInspector* derives the end state according to the start state, condition, and actions: after stopping T3540 and responding, the UE remains connected via the N1 NAS signalling connection, but T3540 is no longer running. In traditional FSM, it is impractical to infer the start and end states, and two operations

in the action may bring unnecessary difficulties in determining which edges shall be included within complicated FSM transitions when there are thousands of connected states.

5.3 *Function Chain Builder*

After extracting scattered SCA nodes, each representing a self-contained state transition, we assemble them into coherent function chains, which complete the transitions for representing the complex mobile network functions specified in 3GPP specifications. Notably, this task is non-trivial due to the challenge outlined in **C2** (Section 4.2), where thousands or more SCA nodes can be developed from thousands of pages from a single 3GPP specification. To construct complete function chains from these scattered SCA nodes, two types of connection strategies are designed:

5.3.1 Type I: Node-Informed Exhaustive Connection. This category uses exhaustive pairwise comparisons between SCA nodes to discover all valid transitions. Three connection methodologies are defined:

(a) Temporal Connection. A temporal connection captures the most basic dependency between SCA nodes. It identifies whether the end state of a SCA node is identical to the start state of another. Formally, let the set of extracted SCA nodes be $N = \{n_1, n_2, \ldots, n_m\}$, where each node n_i is defined as

$$n_i = (\text{start}(n_i), \text{condition}(n_i), \text{action}(n_i), \text{end}(n_i)).$$

Each node $n_i \in N$ is associated with a start state and an end state that $\text{start}(n_i),\ \text{end}(n_i) \in Q$, where Q denotes the set of all specification states extracted from the 5G and 4G NAS and RRC specifications. The Temporal-Connection relation T is defined as

$T = \{(n_i, n_j) \in E \times N \mid \text{end}(n_i) = \text{start}(n_j)\}$. SCA node n_j is a valid temporal successor of n_i if the start state of n_j is identical to the end state of n_i. The formalized algorithm is shown in Algorithm 1 in Appendix C. As shown in Figure 3, the SCA Node 2110 extracted from the 5G NAS specification presents a transition from 5GMM-IDLE mode with suspend indication to 5GMM-CONNECTED mode upon receiving an indication from lower layers that the RRC connection has resumed. Another Node 2055 represents a transition starting exactly in 5GMM-CONNECTED mode, describing the transition from 5GMM-CONNECTED to 5GMM-CONNECTED with RRC inactive. Because the end state of Node 2110 is identical to the start state of Node 2055, they meet the requirement for a valid temporal connection.

(b) Semantic Connection. Temporal Connection requires that two SCA nodes must have exactly the same end state and start state. However, 3GPP specifications are written by different experts. There may be slight differences in textual descriptions for the identical concepts or functions in 3GPP specifications, which we define as the *textual descriptions different but semantically equivalent*. To connect SCA nodes

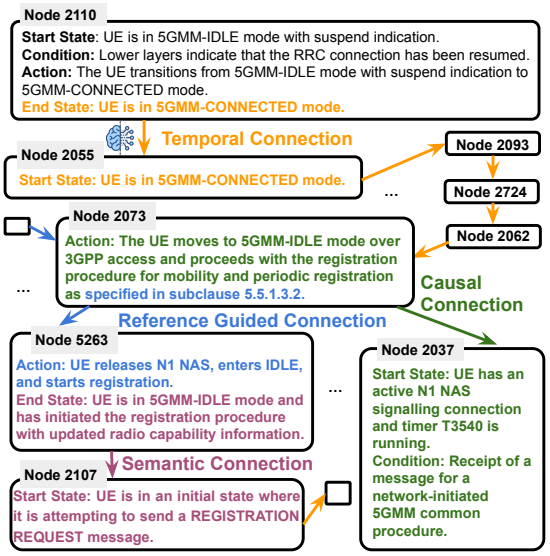

Figure 3: 5G mobility management function chain

with such a relation, we introduce Semantic Connection, which identifies if the end state $\text{end}(n_i)$ of Node n_i and the start state $\text{start}(n_j)$ of Node n_j are semantically equivalent or strongly related. If so, Node n_j is allowed to be the successor of Node n_i, even when the textual descriptions of $\text{end}(n_i)$ and $\text{start}(n_j)$ differ. Formally, given two SCA nodes, we check their semantics whether $sem(\text{end}(n_i),\ \text{start}(n_j)) = \text{Yes}$

to associate the SCA nodes with *textual descriptions different but semantically equivalent* for their end states and start states. In our approach, we leverage *CellSecInspector-Core*, which has been trained with profuse domain knowledge, to compare their semantic meaning. It takes 3 steps: (i) the SCA nodes' start and end states are tokenized; (ii) the transformer layers are used to build contextual embeddings based on tokens; (iii) the embeddings are further compared via the cosine similarity and the enhanced verification via ICL to distinguish deeper logic relations, such as entailment, contradictions, and overlaps. This relation S is defined as

$$S = \left\{(n_i, n_j) \in N \times N \;\middle|\; sem(\text{end}(n_i),\ \text{start}(n_j)) = \text{Yes}\right\}$$

The formalized algorithm is illustrated in Algorithm 2 in Appendix C. As shown in Figure 3, Node 5263 ends in a state where the UE is in 5GMM-IDLE mode and has initiated the registration procedure due to a change in its radio capability. Node 2107 begins in a state where the UE is attempting to send a REGISTRATION REQUEST message. It proceeds to enter 5GMM-IDLE mode and continues the registration procedure. Although the end and start states are described differently, they both refer to the same specification state, the UE initiating a registration procedure triggered by updated radio capability. This represents a valid semantic connection.

(c) Causal Connection. Beyond the direct matching of end and start states, function chains can be causally constructed through causal relationships, where the states, conditions, or actions of one SCA node directly enable, influence, or trigger the states, conditions, or actions of another. That is, the states, conditions, or actions in n_i serve as a plausible prerequisite or cause for the states, conditions, or actions in n_j, suggesting a likely sequential or dependent relationship. Identifying causal connections between nodes is essential to thoroughly construct complete and coherent function chains. We define this Causal-Connection relation C such that for any pair of SCA nodes (n_i, n_j), a connection exists if $C = \{(n_i, n_j) \in N \times N \mid \text{causal}(n_i, n_j) = \text{Yes}\}$.

CellSecInspector-Core is employed to infer such causal relations. It analyzes SCA nodes n_i and n_j, and determines whether the n_i precedes n_j. For example, if the end state of n_i transits the UE into a mode that is required as the start state of n_j, or if the condition in n_i (e.g., radio capability change) leads to logically subsequent condition in n_j (e.g., registration request trigger), or if the action in n_i (e.g., releasing NAS signaling connection) sets up the environment necessary for the action in n_j (e.g., initiating a new signaling procedure), a causal relationship is inferred.

As shown in Figure 3, Node 2073 models the UE transitioning from 5GMM-CONNECTED to 5GMM-IDLE mode and initiating a registration procedure. As part of this process, the UE establishes an N1 NAS signalling connection, which is necessary for subsequent communication with the network. Node 2037 begins in a state where such a signalling connection is already active and the UE is ready to handle network-initiated 5GMM common procedures. Its condition, receiving a message from the network, relies on the existence of an active signaling context. This requirement is fulfilled by the action on Node 2073, which creates the signaling connection through registration. With this context, Node 2037 can further proceed to receive or respond to the network-initiated procedure. Therefore, the action in Node 2073 is a prerequisite for the state and condition of Node 2037. We identify it as a valid causal connection $(n_{2073}, n_{2037}) \in C$.

5.3.2 Type II: Reference Guided Connection. While the connection strategies in Type I can, in principle, be applied to all SCA nodes, this approach is computationally expensive as discussed in **C3** (see Section 4.2). A single 3GPP specification can yield thousands of nodes, and across the more than 2,000 specifications that define operational 5G and 4G mobile networks [3], exhaustive pairwise validation is infeasible. To reduce the computational cost, we constrain the search space so that each SCA node is only checked with other nodes within the same subclause. However, this restriction risks constructing incomplete function chains. We introduce the Referenced Guided Connection to address it. To ensure consistency and maintainability, 3GPP specifications are written in a highly modular style, relying heavily on cross-references to avoid redundancy and to coordinate contributions across multiple working groups. These references are the key of Reference Guided Connection, which works as follows: **1. Reference Detection**: Identify whether a SCA node n_i contains references to specific clauses (e.g., '*as specified in subclause X.Y.Z*'). **2. Space Expansion**: Extract the referenced clause and retrieve all candidate nodes n_j within that clause. **3. Connection Validation**: Apply Type I connections to determine if a valid connection exists between n_i and n_j.

As illustrated in Figure 3, we apply the Reference Guided Connection strategy to link Node 2073 and Node 5263. First, Node 2073 explicitly cites subclause 5.5.1.3.2 in its action field, which specifies the registration procedure triggered by radio capability updates. Based on this reference, all SCA nodes derived from subclause 5.5 are considered candidate targets, including Node 5263. We then apply Type I mechanisms to validate whether a true dependency exists. In this instance, a Causal Connection from Node 2073 to Node 5263 is detected. Specifically, Node 2073 initiates a registration procedure. This behavior corresponds to the more detailed process described in Node 5263, where the registration is triggered by a radio capability change. The action in Node 2073 logically precedes and leads to the action in Node 5263, whose execution realizes the registration behavior initiated in Node 2073. Although their start and end states differ, the procedural flow from Node 2073 to Node 5263 is a causal dependency. Thus, it is a valid Reference Guided Connection.

5.3.3 Complete Function Chain. Applying all proposed connection strategies to analyze the relations among SCA nodes, the *Function Chain Builder* can efficiently and comprehensively construct the function chain.

These function chains serve as structured models of complex mobile network services. Figure 3 depicts the constructed function chains that model the 5G mobility management.

6 SecOracle

SecOracle (Security Oracle) is an adaptive security validation module to assess the security of procedures and functions defined in 3GPP specifications. Current methods, such as exhaustive character-altered testing (i.e., random fuzzing [30]), manually developing mobile network procedure-specific requirements [11, 19], or manually transpiling FSM models into SMV language [11, 23, 32, 34], are inefficient and unscaled for assessing large-scale function chains (**C3** in Section 4.2).

To address it, we propose a two-step approach: First, we define a set of security properties, which are different from the security requirements in prior works that are specific to each

procedure or FSM model [11, 19]. These properties encapsulate the fundamental requirements for entire mobile networks, encompassing both mobile users and network infrastructure operators. They thus can cover a wider range of scenarios and conditions. Derived from security standards (e.g., 3GPP Security Assurance Methodology (SECAM) [2], ITU-T X.805 [1]) and classic security models (e.g., Dolev-Yao [29], Bell-LaPadula [38], and Biba Models [14]), we build 9 core security properties in total (see Appendix E) to cover key security dimensions. The followings are a partial of our defined security properties: `authentication`, ensuring only legitimate users or entities can initiate procedures; `authorization`, enforcing that state transitions, such as NAS transitions, are exclusively triggered by legitimate, authenticated entities; `service integrity and confidentiality`, requiring all communications to be properly secured, with mobile entities verifying message integrity and confidentiality to prevent adversaries from launching attacks such as MitM.

Second, motivated by that cellular network devices, including UEs, base stations, and function modules in Core Network, do message exchanges all the time, 4 fundamental attack methods, including dropping, modifying, rejecting, and replaying, denoted as A are applied to each node and node transition. *SecOracle* leverages reasoning capability to systematically analyze whether function chains still meet all security properties under these 4 attacks. Formally, given $\forall R_i \in$Security Properties, $\forall n_j \in$ C for each function chain, $\forall a_k \in$ A, the security property verification is expressed as, CheckViolation(R_i, n_j, a_k). A violation is flagged and recorded as $C, a_k, n_j \models \neg R_i$ for further analysis. Figure 7 in Appendix C shows how *CellSecInspector-Core* is instructed in this module.

Consider the function chains shown in Figure 3. For each node, *SecOracle* applies 4 attack methods to assess possible violations against security properties. In Node 2073, the UE sends a REGISTRATION REQUEST to transit from 5GMM-CONNECTED to 5GMM-IDLE mode. As a detected violation shown in Figure 4, under a replay attack, if this REGISTRATION REQUEST is intercepted and replayed, the network may process it again. According to 3GPP 5G NAS specification TS 24.501 subclause 4.4.4.3 [7], the AMF must accept such a message even if it is unauthenticated and lacks integrity protection. As a result, the replayed message is processed as valid, creating duplicate registration contexts and inconsistent session states. These inconsistencies can lead to billing errors, misrouted downlink traffic, or even session management failures. In severe cases, adversaries can repeatedly force the UE to re-register, thereby disrupting ongoing sessions and achieving a denial-of-service (DoS) attack using forged NAS messages [34]. This is a specification design vulnerability, violating the property `service integrity`, as

```
| 2062 | 2073 | Replay | BEGIN_JSON {
  "vulnerability_detected": "Yes",
  "violated_property": [5],
  "explanation": "A Replay attack could exploit the transition
      from 5GMM-CONNECTED mode with RRC inactive indication to
      5GMM-IDLE mode by capturing and retransmitting the
      REGISTRATION REQUEST message. This violates the Replay
      Attack Protection requirement.",
  "attack_case": "An attacker captures the REGISTRATION REQUEST
      message during the transition and retransmits it, causing
      the UE to process the same message twice, leading to
      potential registration failures or security breaches.",
  "vulnerability_source": "event2073"
} END_JSON |
```

Figure 4: A violation flagged by *SecOracle* for Nodes 2062 and 2073 transition

the 3GPP specification explicitly permits the AMF to accept unauthenticated REGISTRATION REQUEST.

7 VulnTestGenerator

After detecting security violations by *SecOracle*, *VulnTestGenerator* generates corresponding testing procedures for validating suspected vulnerabilities. It aims to narrow the gap between specification-level findings and real-world verifications. As shown in Figure 8 in Appendix C, *VulnTestGenerator* is instructed via ICL to develop structured test cases that specify network states, security context status, UE and core network configurations, operation sequences, and expected outputs, from flagged violations. Those test cases can guide us how to validate and observe the violated security properties. Appendix F presents an example test case for validating the security property `service integrity` violation in Node 2073.

Validating vulnerabilities with test cases. Executing the generated test cases on either a controlled testbed (e.g., open5Gs [24], srsRAN [50]) or an operational mobile network allows us to validate the vulnerability and find the root cause. Notably, conducting real-world experiments requires substantial human effort, including building experimental platforms, instrumenting the network/UE to collect experimental data, and analyzing results. The current scope of *VulnTestGenerator* focuses on generating test cases to guide these validation experiments. Executing test cases in an automated manner is a non-trivial activity. We leave automated execution as future work.

8 Evaluation

This section reports the evaluation of *CellSecInspector*. Implementations and 3GPP specifications used for evaluations are introduced in Appendix D.

8.1 Research Question

To evaluate the performance of *CellSecInspector*, we answer the following research questions.

◇ **RQ1:** Can *CellSecInspector* identify new vulnerabilities in 3GPP specifications?

◇ **RQ2:** How does *CellSecInspector* compare with the state-of-the-art approach in vulnerability discovery effectiveness?
◇ **RQ3:** Do SCA nodes outperform FSMs used in prior works?
◇ **RQ4:** What is the accuracy of *SecOracle*?
◇ **RQ5:** What is *CellSecInspector*'s runtime performance?

8.2 RQ1: New Vulnerabilities

Among the well-studied TS 38.331, TS 24.501, and TS 24.301, *CellSecInspector* uncovers 5 new vulnerabilities. Beyond these three specifications, which are also used in prior work and therefore enable a fair comparison, we further apply *CellSecInspector* to TS 23.501 (System architecture for the 5G System (5GS)) and TS 24.229 (IP multimedia call control protocol based on Session Initiation Protocol (SIP) and Session Description Protocol (SDP); Stage 3). Considering validating each finding requires substantial human effort, *CellSecInspector* analyzes only one selected section from TS 23.501 and TS 24.229, respectively; *CellSecInspector* identifies two additional vulnerabilities in these sections. Details of new vulnerabilities will be reported later.

8.3 RQ2: Vulnerability Discovery Comparison

Beyond discovering new vulnerabilities, it is important to understand how *CellSecInspector* performs relative to prior art. Accordingly, we compare *CellSecInspector* with the state-of-the-art approach in terms of vulnerability discovery effectiveness. This comparison is intended to assess whether *CellSecInspector* can have a better coverage of security flaw identification in 3GPP specifications. As discussed in Section 3, some existing approaches [19, 20, 51] target implementation-level issues instead of detecting design flaws in 3GPP specifications. *CellularLint* [47] aims to detect sentence-level semantic conflicts in 3GPP specifications rather than finding the design vulnerabilities. To the best of our knowledge, *Hermes* [11] is the only comparable system that can also discover vulnerabilities from standards. To enable a fair comparison, we collect and develop a benchmark set of known vulnerabilities [11, 12, 15, 18, 21, 22, 32–35, 37, 39, 40, 42, 44, 49, 52, 53] that are design defects from the same specifications (TS 38.311, TS 24.501, and TS 24.301) used by *Hermes*.

As shown in Table 1, *Hermes* can detect 22 of all 36 known vulnerabilities in 5G and 4G RRC specifications. After we manually verify the flagged violations by *CellSecInspector*, we confirm that *CellSecInspector* is capable of discovering all of these known vulnerabilities. The vulnerability discovery results show that *CellSecInspector* is capable of effectively modeling mobile network procedures, conducting security analysis, and identifying security threats in 3GPP specifications. Combining the results in RQ1, *CellSecInspector* not only has better vulnerability discovery coverage, but also can discover new vulnerabilities.

ID	Attack	H*	C**
1	Downgrade to non-LTE network services [49]	✓	✓
2	Denying all network services [49]	✓	✓
3	Denying selected service [49]	×	✓
4	Signaling DoS [12, 35, 39, 40]	✓	✓
5	S-TMSI catching [37]	✓	✓
6	IMSI catching[52]	✓	✓
7	EMM Information [44]	✓	✓
8	Impersonation attack [21]	×	✓
9	Synchronization Failure attack [53]	×	✓
10	Malformed Identity Request [42]	×	✓
11	Neutralizing TMSI refreshment [34]	×	✓
12	NAS Counter Reset [34]	✓	✓
13	Uplink NAS Counter Desynchronization [34]	✓	✓
14	Exposing NAS Sequence Number [34]	✓	✓
15	Cutting off the Device [34]	×	✓
16	Exposure of SQN [15]	✓	✓
17	5G AKA DoS Attack[18]	✓	✓
18	SUCI catching [22]	×	✓
19	IMSI cracking [33]	×	✓
20	NAS COUNT update attack [11]	✓	✓
21	Deletion of allowed CAG list [11]	✓	✓
22	Downgrade using ATTACH/REGISTRATION REJECT [49]	✓	✓
23	AUTHENTICATION REJECT attack [53]	✓	✓
24	DETACH/DEREGISTRATION REQUEST attack [32]	✓	✓
25	SERVICE REJECT attack [49]	✓	✓
26	Denial-of-Service with RRC SETUP REQUEST attack [34]	×	✓
27	Installing Null Cipher and Null Integrity [34]	✓	✓
28	Lullaby Attack [34]	✓	✓
29	Incarceration with RRC REJECT/RELEASE [34]	✓	✓
30	Measurement report [49]	×	✓
31	RLF report [49]	✓	✓
32	Blind DoS attack [37]	×	✓
33	AKA bypass [37]	×	✓
34	Paging channel hijacking [32]	×	✓
35	Energy Depletion with RRC SETUP [11]	✓	✓
36	V2X Message Spoofing over PC5 [48]	×	✓
	Detected Ratio	**22/36**	**36/36**

H*: *Hermes*, C**: *CellSecInspector*

Table 1: Comparison of KNOWN vulnerabilities detected by Hermes and *CellSecInspector*

8.4 RQ3: SCA Nodes vs. FSMs

While *CellSecInspector* can identify new vulnerabilities and its discovery effectiveness is evaluated through comparisons with *Hermes*, these results alone do not explain why *CellSecInspector* has superior performance. Most prior works [11, 51] follow the traditional approach to construct the FSMs and then apply model checking for formal verification. We therefore next investigate the source of *CellSecInspector* 's advantage by evaluating the SCA nodes, which form the core representation used by *CellSecInspector* for standards reasoning and vulnerability analysis.

We compare SCA nodes with the conventional FSMs synthesized by prior works to model the mobile network functions. *Hermes* and *ARCANE* are applied on the same specifications for 5G/4G NAS and RRC to construct FSMs for comparison. Note that a conventional FSM may only have edges and nodes without explicit condition and action states. To compare FSMs to SCA nodes, we convert FSMs constructed

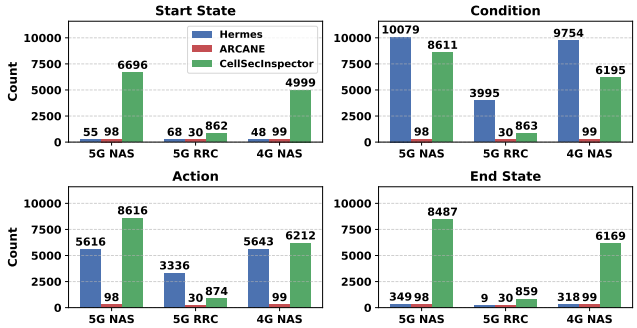

Figure 5: Quantity Comparison of SCA nodes and FSMs

by *Hermes* and *ARCANE* in the form of the same four fields in SCA nodes with additional procedures as follows: *Hermes* uses the constituency parse tree to split the condition and action states from edges; *ARCANE* relies on manual labeling to determine the condition and action states. Following the same procedure, we manually split its edges into two fields. Using these four fields, including start/end states, conditions, and actions, can precisely describe state transitions of mobile network functions. We next assess the comparisons using the metrics of the quantity, completeness, and accuracy of state transitions.

Quantity. Figure 5 presents the comparison among *CellSecInspector*, *Hermes*, and *ARCANE* in modeling 5G/4G NAS and RRC specifications using SCA representations and FSM states.

There are three key observations. (1) *Hermes* and *ARCANE* both capture significantly fewer start and end states, indicating that both of them do not derive as many mobile network functions from 3GPP specifications as *CellSecInspector*. Table 2 compares an SCA node derived by *CellSecInspector* and a FSM transition annotated by *Hermes* for a sentence '*A UE enters the state 5GMM-SERVICE-REQUEST-INITIATED after it has started the service request procedure and is waiting for a response from the network.*' from TS 24.501. While the sentence does not explicitly specify the start state, *CellSecInspector* reasonably infers that the UE is likely in 5GMM-REGISTERED state and correctly recognizes two sequential conditions, the action, and the end state. In contrast, *Hermes* fails to extract and infer the start state, the action, and the end state in the FSM transition due to limited contextual reasoning, resulting in incomplete FSM transitions and limited coverage. (2) *Hermes* reports a larger number of extracted conditions and actions in 5G and 4G NAS and RRC specifications. However, it is primarily due to splitting closely connected conditions and actions within a single sentence into multiple independent entries, as shown in Table 2. This fragmentation inflates raw counts of extracted elements but fails to preserve the internal structure and logical interdependencies among conditions and actions. (3) *ARCANE* constructs fewer condition and action states as well, while their counts are consistent

Element	*CellSecInspector* (SCA Node)	*Hermes* (FSM Transition)
Start State	UE is likely in 5GMM-REGISTERED.	<start_state> A UE enters the state 5GMM-SERVICE-REQUEST-INITIATED </start_state>
Condition	After the UE has started the service request procedure AND is waiting for a response from the network.	<condition> it has started the service request procedure </condition> <condition> is waiting for a response from the network </condition>
Action	UE enters the 5GMM-SERVICE-REQUEST-INITIATED state.	N/A
End State	UE is in the 5GMM-SERVICE-REQUEST-INITIATED state, waiting for a response from the network.	N/A

Table 2: Comparison between a SCA node from *CellSecInspector* and a FSM transition from *Hermes*

Spec#	Method	0 fields	1 field	2 fields	3 fields	4 fields
24.501	Hermes	1122 (20.1%)	1055 (19.0%)	2177 (39.1%)	207 (3.7%)	5 (0.1%)
	CellSec	1778 (17.1%)	36 (0.3%)	132 (1.3%)	1920 (18.4%)	6549 (62.9%)
	ARCANE	0 (0%)	0 (0%)	0 (0%)	0 (0%)	98 (100%)
38.331	Hermes	2948 (46.2%)	2675 (41.9%)	753 (11.8%)	1 (0.0%)	0 (0.0%)
	CellSec	321 (26.9%)	11 (0.9%)	1 (0.1%)	7 (0.6%)	855 (71.5%)
	ARCANE	0 (0%)	0 (0%)	0 (0%)	0 (0%)	30 (100%)
24.301	Hermes	3026 (50.6%)	463 (7.7%)	2244 (37.5%)	237 (4.0%)	6 (0.1%)
	CellSec	1771 (22.2%)	30 (0.4%)	13 (0.2%)	1207 (15.1%)	4974 (62.2%)
	ARCANE	0 (0%)	0 (0%)	0 (0%)	0 (0%)	99 (100%)

Table 3: SCA completeness comparison of Hermes, ARCANE, and *CellSecInspector*

with its derived start and end states. Different from *CellSecInspector* and *Hermes*, *ARCANE* has a strict requirement to construct FSMs, which doesn't allow start/end states or condition/action to have a missing field. However, since it is not capable of capturing the comprehensive functions, *ARCANE* does not capture many important state transitions, such as the example shown in Table 2. In summary, *CellSecInspector* can identify and infer more implicit states using contextual information, while *Hermes* and *ARCANE* can not achieve it.

Completeness. We evaluate the completeness of state transitions. A state transition is defined to be complete if all 4 corresponding start, condition, action, and end fields can be derived. A field is considered valid if it is non-empty and not a placeholder (e.g., "Not specified" or "Not explicitly defined"). Importantly, an incomplete state transition doesn't mean it is not accurate or it can't be used for further security analysis. However, more fields indicate that more information is likely derived from 3GPP specifications. Table 3 shows the completeness comparison of *Hermes*, *ARCANE*, and *CellSecInspector*.

There are three key observations. (1) *Hermes* has much less complete state transitions compared to *ARCANE* and *CellSecInspector* across all specifications. A large portion of state transitions contains only partial fields. Only 0.1%, 0%, and 0.1% state transitions derived by *Hermes* have all 4 fields. Such many incomplete state transitions can bring potentially negative impacts for further security analysis

Method	Total	Accuracy	$\{a, b, c, d\}$	Raw Agr.	κ
CellSecInspector	3147	98.92%	{3113, 13, 8, 13}	99.33%	0.80
Hermes	3634	25.23%	{917, 32, 28, 2657}	98.35%	0.96
ARCANE	162	0.00%	{0, 0, 0, 162}	100.00%	N/A

Table 4: Expert judgment of SCA nodes and FSMs

Model	Expert1	Expert2	Avg. Acc.	Cohen's κ
DeepSeek-V3.2	1.00	1.00	1.00	1.00
Qwen-Plus	1.00	1.00	1.00	1.00
Qwen3-4B	0.90	0.90	0.90	0.92
DeepSeek-R1-8B	0.90	1.00	0.95	0.94

Table 5: Expert validation of SCA node extraction across 4 models

for missing critical specification context. (2) *ARCANE* exhibits the most restricted structure extraction as mentioned above. Its intermediate representation models specification behavior primarily as message-level transitions with IE validity constraints. If the specification description does not explicitly provide all required semantic elements, *ARCANE* cannot reconstruct a complete representation. As a result, only a small number of state transitions is derived and captured. (3) *CellSecInspector* achieves the best balance between constructing more and complete state transitions. Across all specifications, the majority of extracted events contain three or four fields, with the proportion of fully specified transitions reaching 62.2% for TS 24.301, 62.9% for TS 24.501, and 71.5% for TS 38.331. While the ratios are less than those of *ARCANE*, *CellSecInspector* constructs significantly more complete state transitions.

Accuracy. We assess the accuracy of extracted SCA nodes and FSMs by judging whether an SCA node or a FSM is accurate based on two criteria: (1) evidence-grounded correctness, whether each extracted representation accurately reflects the protocol behavior described in the specification and is supported by explicit textual evidence, and (2) semantic consistency, whether the extracted representation preserve the intended meaning of the mobile network function behavior described in 3GPP specifications. Because this judgment requires domain expertise in cellular network standards, it cannot be reliably outsourced to crowdworkers. We thus ask two domain experts to independently judge whether an SCA node/a FSM is accurate. Formally, each SCA node and FSM is labeled as $v_i \in \{Pass, Fail\}$. We define that an SCA node or FSM is accurate only if both experts label it as Pass. Based on the annotations from two experts, we construct a contingency table $\{a, b, c, d\}$, where a denotes Pass/Pass, b denotes Pass/Fail, c denotes Fail/Pass, and d denotes Fail/Fail. Due to the large amount of SCA nodes and FSMs, two experts are asked to assess the SCA nodes and FSMs extracted from the selected sections, including Clause 4 of 4G NAS specification, Clause 4 of 5G NAS specification, Clauses 4 and 5 of 5G RRC specification. We report the acceptance rate (a/N), raw agreement ($(a+d)/N$), and Cohen's κ in Table 4.

SCA nodes achieve the highest accuracy compared to FSMs constructed by *Hermes* and *ARCANE* for two reasons. First, a large portion of FSMs extracted by *Hermes* fail to capture the intended meaning from 3GPP specifications. The missing fields, such as actions or conditions, in FSMs do not meet the experts' expectation for precisely modeling a state transition, such as the example shown in Table 2. Second, in *ARCANE*, FSM edges are modeled as message-level transitions with field validity constraints. Specifically, for passive learning and fuzzing, *ARCANE* introduces a simplified intermediate representation (IR) in which each message retains only the message type and a list of information elements (IEs). The values of these IEs are abstracted into three states: valid, invalid, or missing, rather than preserving their full semantic meaning. For example, the protocol behavior following an ATTACH REQUEST is reduced to a representation of the form "a message of a certain type appears, with several IEs marked as valid." This abstraction compresses the fine-grained semantics of specification execution, such as why a response occurs and what actions the specification should perform, into a coarse transition defined solely by message type and IE validity. This approach may be suitable for further fuzzing operations. However, none of the experts agree that its FSMs successfully model the mobile network functions.

In addition, we evaluate whether the accuracy of extracted SCA nodes can be affected if *CellSecInspector-Core* is built on top of different vanilla models. Four models, **Qwen-Plus-2025-12-01**, **Qwen3-4B-Instruct-2507**, **DeepSeek-V3.2 Release 2025/12/01**, and **DeepSeek-R1-Distill-Llama-8B**, are used to adapt cellular domain knowledge and further used to extract SCA nodes. We next conduct expert validation on a representative subset of extracted nodes from these models. Two domain experts independently evaluate whether each SCA node is correct with respect to the specification text. Table 5 reports the validation results, including the accuracy of each expert, the average accuracy, and the inter-annotator agreement measured by Cohen's κ. Both DeepSeek-V3.2 and Qwen-Plus achieve the top validation accuracy in our sampled evaluation, while the smaller local models exhibit lower accuracy. These results indicate that larger models are preferable choices.

8.5 RQ4: Violation Detection Accuracy

After aggregating violations, we manually examine each one to determine whether the issue identified by *SecOracle* is correct. As mentioned in RQ2, not every violation can be assessed and confirmed; we count those violations as incorrect. Under this criterion, *SecOracle* reports 90 violations in

TS 24.501, of which 18 are confirmed as valid design defects; 62 violations in TS 24.301, of which 15 are verified as genuine standard defects; 161 violations in TS 38.331, of which 12 are valid; 36 violations in the analyzed clause of TS 23.501, of which 1 is valid; and 8 violations in the analyzed clause of TS 24.229, of which 1 is valid. Note that the known vulnerabilities 22, 23, 24, 25 in Table 1 appear in both 5G and 4G specifications. We count them as separate valid violations because they arise from distinct specifications.

Thus, the per-specification accuracy of violation detection is 20.0% (TS 24.501), 24.2% (TS 24.301), 7.5% (TS 38.311), 2.8% (TS 23.501), and 12.5% (TS 24.229). The overall accuracy is 13.2%. They indicate that *CellSecInspector* is effective in identifying potential design defects in 3GPP specifications, although there remains clear room to improve the precision of violation detection. Two observations are worth noting. First, a substantial portion of flagged violations originates from the same vulnerabilities. For example, the large number of violations in TS 38.331 is mainly due to identical RRC messages reused across multiple procedures, leading to duplication reports. Second, not all violations can be assessed now, as discussed in RQ2. Counting such cases as incorrect lowers the measured overall accuracy.

8.6 RQ5: Runtime Performance

We measure the end-to-end processing time of key components, including *SCA Representation Extractor*, *Function Chain Builder*, *SecOracle*, and *VulnTestGenerator*, .

Specifically, on the full specifications of TS 24.501 (5G NAS), TS 23.301 (4G NAS), TS 38.331 (5G RRC), and selected clauses of TS 23.501 (5G system architecture), and TS 24.229 (IMS signaling), *SCA Representation Extractor* derives 10,415 SCA nodes in 17.36 hours, 7,995 SCA nodes in 13.33 hours, 1,195 SCA nodes in 1.99 hours, 1,253 SCA nodes in 2.09 hours, and 320 SCA nodes in 0.53 hours. *Function Chain Builder* takes the most time, approximately 60 days, 30 days, 4 days, 0.44 days, and 0.03 days to construct 11,273, 6,943, 2,22, 83, and 6 function chains with an average of 5.46 seconds per SCA pair analysis. Notably, 99% of the time was spent identifying Semantic Connection and Causal Connection. Note that this step demands substantial computational resources, and the introduction of Reference Guided Connection significantly reduces the searching space for less computational resources. Without Reference Guided Connection, analyzing the relations of all SCA pairs for constructing function chains becomes infeasible with the computational resources in our experiment setting, requiring an estimated 18.78 years ($10,415^2 \times \frac{5.46 \text{ sec}}{3600 \times 24 \times 365} = 18.78$ years), 11.06 years, 0.25 years, 0.27 years, and 0.018 years. *SecOracle* and *VulnTestGenerator* together complete the vulnerability analysis and develop corresponding test procedures in 17.54 hours, 10.80 hours, 0.35 hours, 2.10 hours, and 0.54 hours for these specifications.

Importantly, *CellSecInspector* can be executed in parallel, which can reduce the overall time required to process individual specifications and enable efficient concurrent analysis across multiple specifications. More computational resources can also increase the runtime performance. Compared with the extensive manual effort traditionally required by domain experts to extract, correlate, and validate these procedures, *CellSecInspector* achieves substantial efficiency gains, demonstrating the feasibility of automated security analysis for 3GPP specifications at scale.

9 Conclusion

Security assurance in cellular networks relies on rigorous validation of conformance to 3GPP specifications, yet automating this process remains challenging due to the complex semantics and dependencies within evolving standards. In this work, we introduce *CellSecInspector*, an automated framework that extracts structured SCA representations, builds function chains through multi-linking strategies, and systematically validates them against 9 security properties under 4 adversarial scenarios, automatically generating test cases for practical verification. Applying it to selected 3GPP specifications, it can uncover 7 new vulnerabilities and also discover 36 known vulnerabilities. These results demonstrate that *CellSecInspector* can significantly improve scalability, accelerate security assessments, and reveal latent security vulnerabilities overlooked.

References

[1] 2003. *Security architecture for systems providing end-to-end communications.* Technical Report X.805. ITU-T.

[2] 2020. *Security Assurance Methodology (SECAM) for 3GPP network products.* Technical Report TR 33.916, v16.0.0. ETSI / 3GPP. https://www.etsi.org/deliver/etsi_tr/133900_133999/133916/16.00.00_60/tr_133916v160000p.pdf Technical report, Release 16.

[3] 2025. 3GPP Specifications. https://www.3gpp.org/specifications. Accessed: 2025-08-31.

[4] 2025. Jetstream2 launches large language model (LLM) inference service. Online; Indiana University Jetstream2 News & Events. https://jetstream-cloud.org/news-events/news/4-11-25_llm-inference-service.html Accessed on 28 August 2025.

[5] 3GPP. 2025. Specifications. Online. [Online]. Available: https://www.3gpp.org/ftp/Specs.

[6] 3rd Generation Partnership Project (3GPP) 2020. *Universal Mobile Telecommunications System (UMTS); LTE; 5G; Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS); Stage 3 (3GPP TS 24.301 version 16.8.0 Release 16).* 3rd Generation Partnership Project (3GPP).

[7] 3rd Generation Partnership Project (3GPP) 2022. *5G; Non-Access-Stratum (NAS) protocol for 5G System (5GS); Stage 3 (3GPP TS 24.501 version 17.8.0 Release 17).* 3rd Generation Partnership Project (3GPP).

[8] 3rd Generation Partnership Project (3GPP) 2022. *TS 38.331: NR; Radio Resource Control (RRC); Protocol specification.* 3rd Generation Partnership Project (3GPP).

[9] 3rd Generation Partnership Project (3GPP) 2025. *System architecture for the 5G System (5GS).* 3rd Generation Partnership Project (3GPP).

[10] 3rd Generation Partnership Project (3GPP) 2026. *Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE; IP Multimedia Call Control Protocol based on Session Initiation Protocol (SIP) and Session Description Protocol (SDP) (3GPP TS 24.229 version 19.5.0 Release 19).* 3rd Generation Partnership Project (3GPP).

[11] Abdullah Al Ishtiaq, Sarkar Snigdha Sarathi Das, Syed Md Mukit Rashid, Ali Ranjbar, Kai Tu, Tianwei Wu, Zhezheng Song, Weixuan Wang, Mujtahid Akon, Rui Zhang, et al. 2024. Hermes: Unlocking security analysis of cellular network protocols by synthesizing finite state machines from natural language specifications. In *33rd USENIX Security Symposium (USENIX Security 24).* 4445–4462.

[12] Ramzi Bassil, Imad H. Elhajj, Ali Chehab, and Ayman Kayssi. 2013. Effects of Signaling Attacks on LTE Networks. In *2013 27th International Conference on Advanced Information Networking and Applications Workshops.* 499–504. doi:10.1109/WAINA.2013.136

[13] Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. 2024. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954* (2024).

[14] K. J. Biba. 1977. *Integrity Considerations for Secure Computer Systems* (esd-tr-76-372 ed.). Technical Report. MITRE Corp. 66 pages.

[15] Ravishankar Borgaonkar, Lucca Hirschi, Shinjo Park, and Altaf Shaik. 2018. New privacy threat on 3G, 4G, and upcoming 5G AKA protocols. *Cryptology ePrint Archive* (2018).

[16] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. arXiv:1508.05326 [cs.CL] https://arxiv.org/abs/1508.05326

[17] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[18] Jin Cao, Maode Ma, Hui Li, Ruhui Ma, Yunqing Sun, Pu Yu, and Lihui Xiong. 2020. A Survey on Security Aspects for 3GPP 5G Networks. *IEEE Communications Surveys & Tutorials* 22, 1 (2020), 170–195. doi:10.1109/COMST.2019.2951818

[19] Yi Chen, Di Tang, Yepeng Yao, Mingming Zha, XiaoFeng Wang, Xiaozhong Liu, Haixu Tang, and Baoxu Liu. 2023. Sherlock on Specs: Building {LTE} Conformance Tests through Automated Reasoning. In *32nd USENIX Security Symposium (USENIX Security 23).* 3529–3545.

[20] Yi Chen, Yepeng Yao, XiaoFeng Wang, Dandan Xu, Chang Yue, Xiaozhong Liu, Kai Chen, Haixu Tang, and Baoxu Liu. 2021. Bookworm game: Automatic discovery of lte vulnerabilities through documentation analysis. In *2021 IEEE Symposium on Security and Privacy (SP).* IEEE, 1197–1214.

[21] Merlin Chlosta, David Rupprecht, Thorsten Holz, and Christina Pöpper. 2019. LTE security disabled: misconfiguration in commercial networks. In *Proceedings of the 12th conference on security and privacy in wireless and mobile networks.* 261–266.

[22] Merlin Chlosta, David Rupprecht, Christina Pöpper, and Thorsten Holz. 2021. 5G SUCI-Catchers: Still catching them all?. In *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks.* 359–364.

[23] Edmund Clarke, Kenneth McMillan, Sérgio Campos, and Vasiliki Hartonas-Garmhausen. 1996. Symbolic model checking. In *International conference on computer aided verification.* Springer, 419–422.

[24] Open5GS Community. [n. d.]. Open5GS: Open Source Implementation of 4G/5G Core Network. https://open5gs.org/. Accessed: 2025-08-31.

[25] Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The pascal recognising textual entailment challenge. In *Machine learning challenges workshop.* Springer, 177–190.

[26] DeepSeek. 2025. DeepSeek-R1-Distill-Llama-8B. https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Llama-8B.

[27] DeepSeek. 2025. DeepSeek-V3.2 Release. https://api-docs.deepseek.com/news/news251201.

[28] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers).* 4171–4186.

[29] Danny Dolev and Andrew Yao. 2003. On the security of public key protocols. *IEEE Transactions on information theory* 29, 2 (2003), 198–208.

[30] Patrice Godefroid. 2020. Fuzzing: Hack, art, and science. *Commun. ACM* 63, 2 (2020), 70–76.

[31] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. 2020. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th annual meeting of the association for computational linguistics.* 8342–8360.

[32] Syed Hussain, Omar Chowdhury, Shagufta Mehnaz, and Elisa Bertino. 2018. LTEInspector: A systematic approach for adversarial testing of 4G LTE. In *Network and Distributed Systems Security (NDSS) Symposium 2018.*

[33] Syed Rafiul Hussain, Mitziu Echeverria, Omar Chowdhury, Ninghui Li, and Elisa Bertino. 2019. Privacy attacks to the 4G and 5G cellular paging protocols using side channel information. *Network and distributed systems security (NDSS) symposium2019* (2019).

[34] Syed Rafiul Hussain, Mitziu Echeverria, Imtiaz Karim, Omar Chowdhury, and Elisa Bertino. 2019. 5GReasoner: A property-directed security and privacy analysis framework for 5G cellular network protocol. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security.* 669–684.

[35] Georgios Kambourakis, Constantinos Kolias, Stefanos Gritzalis, and Jong Hyuk Park. 2011. DoS attacks exploiting signaling in UMTS and IMS. *Computer Communications* 34, 3 (2011), 226–235.

[36] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, and Christopher Kanan. 2018. Measuring catastrophic forgetting in neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[37] Hongil Kim, Jiho Lee, Eunkyu Lee, and Yongdae Kim. 2019. Touching the untouchables: Dynamic security analysis of the LTE control plane. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1153–1168.

[38] Leonard J LaPadula and D Elliot Bell. 1996. *Secure computer systems: A mathematical model.* Technical Report. Citeseer.

[39] Patrick PC Lee, Tian Bu, and Thomas Woo. 2009. On the detection of signaling DoS attacks on 3G/WiMax wireless networks. *Computer Networks* 53, 15 (2009), 2601–2616.

[40] Wai Kay Leong, Aditya Kulkarni, Yin Xu, and Ben Leong. 2014. Unveiling the hidden dangers of public ip addresses in 4g/lte cellular data networks. In *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications*. 1–6.

[41] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* 33 (2020), 9459–9474.

[42] Benoit Michau and Christophe Devine. 2016. How to not break LTE crypto. In *ANSSI Symposium sur la sécurité des technologies de l'information et des communications (SSTIC)*.

[43] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems* 35 (2022), 27730–27744.

[44] Shinjo Park, Altaf Shaik, Ravishankar Borgaonkar, and Jean-Pierre Seifert. 2016. White rabbit in mobile: Effect of unsecured clock source in smartphones. In *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*. 13–21.

[45] Qwen. 2025. Qwen3-4B-Instruct-2507. https://huggingface.co/Qwen/Qwen3-4B-Instruct-2507.

[46] Qwen. 2026. *Qwen-Plus Model Documentation.* https://modelstudio.console.alibabacloud.com/ap-southeast-1/?tab=doc#/doc/?type=model&url=2840914_2&modelId=qwen-plus

[47] Mirza Masfiqur Rahman, Imtiaz Karim, and Elisa Bertino. 2024. {CellularLint}: A systematic approach to identify inconsistent behavior in cellular network specifications. In *33rd USENIX Security Symposium (USENIX Security 24)*. 5215–5232.

[48] Roshan Sedar, Charalampos Kalalas, Francisco Vázquez-Gallego, Luis Alonso, and Jesus Alonso-Zarate. 2023. A comprehensive survey of V2X cybersecurity mechanisms and future research paths. *IEEE Open Journal of the Communications Society* 4 (2023), 325–391.

[49] Altaf Shaik, Ravishankar Borgaonkar, N Asokan, Valtteri Niemi, and Jean-Pierre Seifert. 2015. Practical attacks against privacy and availability in 4G/LTE mobile communication systems. *arXiv preprint arXiv:1510.07563* (2015).

[50] Software Radio Systems. [n. d.]. srsRAN: Open Source LTE and 5G RAN Implementation. https://www.srsran.com/. Accessed: 2025-08-31.

[51] Sixu Tan, Zeyu Li, Zhutian Liu, Harsh Patel, and Zhaowei Tan. 2025. Automated Model-Based Fuzzing for 5G O-RAN. In *Proceedings of the 31st Annual International Conference on Mobile Computing and Networking*. 201–215.

[52] Fabian Van Den Broek, Roel Verdult, and Joeri De Ruiter. 2015. Defeating IMSI catchers. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*. 340–351.

[53] Chuan Yu and Shuhui Chen. 2019. On Effects of Mobility Management Signalling Based DoS Attacks Against LTE Terminals. In *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*. 1–8. doi:10.1109/IPCCC47392.2019.8958725

[54] Miao Zhang, Runhan Feng, Hongbo Tang, Yu Zhao, Jie Yang, Hang Qiu, and Qi Liu. 2025. Automated Extraction of Protocol State Machines from 3GPP Specifications with Domain-Informed Prompts and LLM Ensembles. arXiv:2510.14348 [cs.NI] https://arxiv.org/abs/2510.14348

A Discussion

Can the defined security properties and four attacks guarantee discovery of all vulnerabilities? *CellSecInspector* aims to reduce reliance on slow, manual expert review by enabling systematic and scalable detection of standard-level defects in the extracted function chains. It does not guarantee the discovery of all vulnerabilities for two main reasons. First, security verification is inherently scoped. Any finite set of security properties may miss emerging or unmodeled failure modes. We can only claim security with respect to the predefined properties and assumptions of the threat model [11, 19]. Second, while not all SCA nodes correspond to explicit message exchanges, some nodes represent event or condition-driven transitions such as local policy decisions. In this case, *SecOracle* still applies the four attacks to such nodes by exploiting their triggered events or conditions. If a SCA node is purely deterministic without any inputs, such as entering one state after another state, this transition is non-attackable under our settings. Noteably, the majority of reported vulnerabilities in RRC and NAS layers included in Table 1 stem from message exchanges and thus fall within the scope of our attack model.

More security properties. Different from prior works [11, 19] that rely on manually crafted security requirements tied to specific procedures, *CellSecInspector* uses security properties that encapsulate fundamental security properties for entire mobile networks. When more critical properties are introduced, it can incorporate them into its framework, enabling automated security analysis with broader coverage in an efficient and scalable manner.

Cross Specification Analysis. Currently, *CellSecInspector* focuses on security analysis within individual specifications. Nevertheless, our proposed Reference Guided Connection provides a foundation for developing the end-to-end function chains that span multiple specifications, allowing the future work for detecting vulnerabilities rooted in cross-specification interactions that are invisible to single-specification analysis.

B Related Work

Table 6 summarizes and contrasts representative 3GPP specification analysis systems, highlighting their objectives, whether they build FSM models, security analysis scope, reproducibility, and comparability.

C *CellSecInspector*

This appendix describes the detailed processing pipelines of the four core components in *CellSecInspector*: the *SCA Representation Extractor* in Figure 6, the procedures of the *Function Chain Builder* are formalized through Algorithm 1, Algorithm 2, Algorithm 3, and Algorithm 4, which respectively define Temporal, Semantic, Causal, and Reference-Guided connections.*SecOracle* in Figure 7, and *VulnTestGenerator* in Figure 8.

Algorithm 1 Temporal Connection

Input: Set of SCA nodes $N = \{n_1, \ldots, n_m\}$
Output: Temporal-Connection relation T
$T \leftarrow \emptyset$
for each ordered pair $(n_i, n_j) \in N \times N$ **do**
  $e_i \leftarrow \text{end}(n_i)$
  $s_j \leftarrow \text{start}(n_j)$
  if $e_i = s_j$ **and** $e_i \notin$ {"not specified", "not explicitly"} **then**
    $T \leftarrow T \cup \{(n_i, n_j)\}$
  end if
end for
return T

Algorithm 2 Semantic Connection

Input: Set of SCA nodes $N = \{n_1, \ldots, n_m\}$, model *CellSecInspector-Core*
Output: Semantic-Connection relation S
$S \leftarrow \emptyset$
for each ordered pair $(n_i, n_j) \in N \times N$ **do**
  $e_i \leftarrow \text{end}(n_i)$
  $s_j \leftarrow \text{start}(n_j)$
  if e_i and s_j are valid **then**
    /* *Representation Encoding* */
    $\mathbf{t}_i \leftarrow$ Tokenize(e_i)
    $\mathbf{t}_j \leftarrow$ Tokenize(s_j)
    $\mathbf{H}_i \leftarrow$ *CellSecInspector-Core*.Encode$(\mathbf{t}_i)$
    $\mathbf{H}_j \leftarrow$ *CellSecInspector-Core*.Encode$(\mathbf{t}_j)$
    $\mathbf{z}_i \leftarrow$ MeanPool(LastLayer$(\mathbf{H}_i)$)
    $\mathbf{z}_j \leftarrow$ MeanPool(LastLayer$(\mathbf{H}_j)$)
    /* *Similarity Estimation* */
    $sim \leftarrow$ Cosine$(\mathbf{z}_i, \mathbf{z}_j)$
    /* *Semantic Verification* */
    Build using sim, e_i, and s_j
    $r \in$ {Yes, No} $\leftarrow$ *CellSecInspector-Core*
    if r = Yes **then**
      $S \leftarrow S \cup \{(n_i, n_j)\}$
    end if
  end if
end for
return S

Algorithm 3 Causal Connection

Input: Set of SCA nodes $N = \{n_1, \ldots, n_m\}$, model *CellSecInspector-Core*
Output: Causal-Connection relation C
$C \leftarrow \emptyset$
for each ordered pair $(n_i, n_j) \in N \times N$ **do**
  /* *Prepare structured reasoning input* */
  Build using:
    start(n_i), condition(n_i), action(n_i), end(n_i)
    start(n_j), condition(n_j), action(n_j), end(n_j)
    causal definition and reasoning rules
  /* *LLM reasoning without similarity metrics* */
  $r \in$ {Yes, No} $\leftarrow$ *CellSecInspector-Core*
  if r = Yes **then**
    $C \leftarrow C \cup \{(n_i, n_j)\}$
  end if
end for
return C

Method	Primary Objective	FSM Modeling	Security Scope	Remarks
Atomic [20]	Detecting hazard indicators via textual entailment	×	Risk-sentence detection only	Identifying hazard indicators via sentence-level textual entailment without modeling specification logic or performing systematic vulnerability analysis
ConTester [19]	Extracting security requirements	×	Developing conformance test cases	Only extracting security requirements for developing conformance test without uncovering design-level vulnerabilities
CellularLint [47]	Detecting semantic inconsistencies	×	Sentence-level contradiction detection	Detecting semantic inconsistencies (i.e., sentence-level contradictions) via sentence-pair inference instead of reasoning over complete specification logic
SpecGPT [54]	LLM-based FSM development via prompts and ensembles	✓	No security reasoning	Constructing FSMs via simple prompt-based LLM inferences without domain-specific efforts; leaving security-hazard discovery and security reasoning out of scope. Furthermore, the system implementation has not been publicly released, limiting reproducibility
Hermes [11]	Automatic FSM extraction via traditional NLP approaches	✓	Formal security analysis	Parsing specifications via handcrafted grammars using constituency parsing and dependency parse trees, remaining rule-intensive and lacking flexibility to adapt to evolving specifications
ARCANE [51]	Automated vulnerability discovery via model-based fuzzing	✓	Implementation vulnerability discovery	Automatically constructs protocol models and performs model-based fuzzing
CellSecInspector	End-to-end SCA nodes and systematic vulnerability reasoning	✓	Specification-level design-flaw detection	First system to enable scalable specification extraction and systematic security analysis

Table 6: Comparison of existing systems for 3GPP specification security analysis

Algorithm 4 Reference-Guided Connection

Input: Set of SCA nodes $N = \{n_1, \ldots, n_m\}$
Output: Reference-Guided relation R
$R \leftarrow \emptyset$
for each node $n_i \in N$ **do**
  /* Stage I: Reference Detection (Rule-Based) */
  $C \leftarrow$ ExtractReferencedClauses(n_i)
  if $C = \emptyset$ **then**
    continue
  end if
  $\mathcal{N}_c \leftarrow$ RetrieveCandidateNodes(C)
  /* Stage II: Type-I Connection Validation */
  for each candidate $n_j \in \mathcal{N}_c$ **do**
    if TemporalCheck(n_i, n_j) **or** SemanticCheck(n_i, n_j) **or** CausalCheck(n_i, n_j) **then**
      $R \leftarrow R \cup \{(n_i, n_j)\}$
    end if
  end for
end for
return R

SCA Representation Extractor

You are a **TECHNICAL SPECIFICATION** expert specializing in **3GPP TS 24.501.** Your task is to analyze the given sentence and extract a SCA node.

SCA node format:
Node ID: {node_id}{section_info}
Sentence: "{sentence}"
Start State: Representing the initial system state before the transition.
Condition: Denoting the triggering clause or prerequisite specified in the specification.
Action: Describing the operation mandated by the specification once the condition is satisfied.
End State: indicating the resulting system state after the action is executed.

Rules:
1. Each sentence corresponds to exactly ONE SCA node.
2. Fill in **all fields (Start State, Condition, Action, End State).**
 - If the sentence only serves as a structural heading, use **"Not specified"** for all fields.
 - If the fields is unclear, use **"Not explicitly defined"** instead of leaving it blank.
3. Retain all 3GPP references (e.g., **"as specified in subclause X.Y.Z"**).
4. Do NOT rephrase the sentence; keep it as provided.
5. Ensure consistency, always follow the event format exactly.

Example:
1. Input Sentence: The non-access stratum (NAS) described in the present document forms the highest stratum of the control plane between UE and AMF (reference point "N1" see 3GPPTS23.501[8]) for both 3GPP and non-3GPP access.

Expected Output:
Event ID: 5 (Derived from Section 4.1)
Sentence: "The non-access stratum (NAS) described in the present document forms the highest stratum of the control plane between UE and AMF (reference point "N1" see 3GPPTS23.501[8]) for both 3GPP and non-3GPP access"
Start State: UE and AMF are not explicitly connected at the NAS level.
Condition: UE needs to communicate with AMF using NAS procedures.
Action: Define NAS as the highest control plane layer and establish its role in UE-AMF interaction.
End State: NAS is identified as the highest control plane stratum for both 3GPP and non-3GPP access.

2. If required by operator policy, the AMF shall include the NSSAI inclusion mode IE in the REGISTRATION ACCEPT message (see table 4.6.2.3.1 of subclause 4.6.2.3). Upon receipt of the REGISTRATION ACCEPT message.

Expected Output:
Event ID: 5675 (Derived from Section 5.5.1.3.4)
Sentence: "If required by operator policy, the AMF shall include the NSSAI inclusion mode IE in the REGISTRATION ACCEPT message (see table 4.6.2.3.1 of subclause 4.6.2.3). Upon receipt of the REGISTRATION ACCEPT message"
Start State: The UE has sent a REGISTRATION REQUEST message to the AMF, and the AMF is processing the registration request.
Condition: Operator policy requires the inclusion of the NSSAI inclusion mode IE in the REGISTRATION ACCEPT message.
Action: The AMF includes the NSSAI inclusion mode IE in the REGISTRATION ACCEPT message as specified in table 4.6.2.3.1 of subclause 4.6.2.3.
End State: The REGISTRATION ACCEPT message, containing the NSSAI inclusion mode IE, is sent to the UE.

Figure 6: ICL instruction for *SCA Representation Extractor*

SecOracle

You are a senior 4G/5G NAS security analyst.
Analyze the following state transition and determine whether a(n) {attack_type} attack
could introduce a realistic vulnerability in the protocol.

From Event (ID: {from_id})
 Start State: {from_event['start_state']}
 Condition: {from_event['condition']}
 Action: {from_event['action']}
 End State: {from_event['end_state']}

To Event (ID: {to_id})
 Start State: {to_event['start_state']}
 Condition: {to_event['condition']}
 Action: {to_event['action']}
 End State: {to_event['end_state']}

The attacker may perform a(n) {attack_type} attack at some point between these two events.

Relevant 5G security requirements include (but are not limited to):
{security_requirements_list}

Evaluation rules:
Respond **"Yes"** ONLY IF the attack leads to a realistic, meaningful vulnerability such as:
 • Authentication bypass
 • Integrity protection failure
 • State inconsistency between UE and network
 • Session hijacking
 • Denial of service
 • Replay-induced state divergence
 • Unauthorized state transition
 • Privacy leakage
Do NOT respond "Yes" only because a message is optional or dropped.
Only respond "Yes" if the protocol behavior causes one of the above security consequences.

Return your answer strictly in the following JSON format:
BEGIN_JSON
{
 "vulnerability_detected": "Yes" or "No",
 "violated_requirements": [list of integers],
 "explanation": "short explanation",
 "issue_classification": "Protocol Design Issue"
 or "Implementation Issue"
 or "Both"
 or "N/A",
 "test_case": "short string or 'N/A'",
 "vulnerability_source": "event{from_id}"
 or "event{to_id}"
 or "event{from_id} and event{to_id}"
 or "unclear"
}
END_JSON

Figure 7: *SecOracle*

VulnTestGenerator

```
You are a mobile network security test engineer specializing in
5G NAS, LTE NAS, and 5G RRC.

Your job is to design a concrete test procedure for an attack scenario
and output it as a Markdown table with the following columns:

| Step | Procedure | U–M | Message | Parameter | Verdict |
|---:|---|:---:|---|---|---|

Definitions:

- Step:
  Integer starting from 1, strictly increasing.

- Procedure:
  Short sentence describing what happens in this step.

- U–M:
  "→" if the main signalling goes from UE to network
  "←" if it goes from network to UE
  "-" if no message is exchanged

- Message:
  Main NAS / RRC message name(s) in this step, or "-" if none.

- Parameter:
  Important parameters / IEs / flags in this step, or "-" if none.

- Verdict:
  "-" for intermediate steps.
  Only the FINAL step should have "Pass" or "Fail" (or a short phrase).

---------------------------------------------------------------------

Attack type: {attack}

node_i (from event):
- Start State: {node_i.Start State}
- Condition:   {node_i.Condition}
- Action:      {node_i.Action}
- End State:   {node_i.End State}

node_j (to event):
- Start State: {node_j.Start State}
- Condition:   {node_j.Condition}
- Action:      {node_j.Action}
- End State:   {node_j.End State}

Extra JSON describing the vulnerability and test idea:

```json
{json_blob}
```
```

Figure 8: *VulnTestGenerator*

D Implementation and Datasets

We use a workstation running Ubuntu 24.04 LTS with an NVIDIA GeForce RTX 4090 GPU (CUDA 12.6; NVIDIA driver 560.35.03) as the primary client environment to access remote compute resources and run supporting experiments. 4 LLMs, DeepSeek-V3.2 [27], Qwen-Plus [46], DeepSeek-R1-8B [26], and Qwen3-4B [45] are used as the base models in the evaluation. We access the first two models through their public APIs. The later two are self-hosted on Jetstream2 [4] using two H100s. DeepSeek-V3.2-Release is primarily used as the base model to build *CellSecInspector-Core*. The implementations for vulnerability validation differ across vulnerabilities; we describe the corresponding setups and procedures in the relevant vulnerability sections.

Specification Dataset. To ensure comparability with prior works, we construct a dataset from the same 3GPP specifications that govern control-plane signaling for NAS and RRC, which are critical for the security and mobility of operational 5G and 4G networks. In addition, to showcase that *CellSecInspector* can be applied to other 3GPP standards, specifications for 5G system architecture and SIP/SDP protocols are also included. Thus, the dataset includes: ***TS 38.331*** version 17.0.0 Release 17 [8], ***TS 24.501*** version 16.8.0 Release 16 [7], ***TS 24.301*** version 16.8.0 Release 16 [6], ***TS 23.501*** version 20.0.0 Release 20 [9], and ***TS 24.229*** version 19.5.0 Release 19 [10].

E Security Properties

F.1 Authentication

Ensures that only legitimate users, devices, and network entities can access services.

- **UE Authentication:** Ensures that only authorized users can access the network.
- **Network Entity Authentication:** Verifies the legitimacy of network nodes (eNodeB, gNodeB, AMF, SMF, etc.).
- **Mutual Authentication:** Guarantees two-way authentication between the UE and the network to prevent man-in-the-middle attacks.
- **SBA Authentication Binding:** Enforces token-based authentication between Service-Based Architecture (SBA) functions.
- **Secure PLMN Selection:** Protects against forced redirection to rogue or unauthorized Public Land Mobile Networks (PLMNs).
- **Roaming & Inter-PLMN Security:** Ensures consistent authentication policies across operators during roaming.

F.2 Authorization

Controls permissions and restricts operations to authorized entities only.

- **API & Service Access Control:** Enforces strict authorization for APIs and service-based interfaces.
- **Policy & QoS Rule Protection:** Prevents unauthorized modification of policies, Quality of Service (QoS) rules, or charging rules.
- **Edge Application Security:** Ensures strong isolation between Multi-access Edge Computing (MEC) applications.
- **SBA API Security:** Requires strong authorization and input validation for Service-Based Architecture (SBA) REST APIs.
- **Slice Isolation:** Ensures strict separation between 5G network slices (eMBB, URLLC, mMTC, etc.).
- **Multi-Tenancy Isolation:** Guarantees tenant isolation in SBA and MEC environments.
- **Network Function Isolation:** Enforces logical separation between core network functions (AMF, SMF, UPF, etc.).

F.3 Service Integrity

Guarantees that services, signaling, and data remain correct, unaltered, and trustworthy.

- **Message Integrity:** Ensures signaling and data messages are not altered during transmission.
- **Security Context Validation:** Ensures NAS/RRC security contexts are properly established before any sensitive exchanges.
- **Secure Paging Mechanisms:** Validates paging messages and prevents spoofed triggers.
- **Secure Emergency Services:** Ensures integrity-protected emergency registration and deregistration procedures.

F.4 Service Confidentiality

Protects sensitive signaling and user data from unauthorized access or disclosure.

- **Data Encryption:** Protects user data and signaling from interception.
- **Service Confidentiality:** Ensures sensitive information remains protected at all times.
- **Integrity & Confidentiality Binding:** Ensures both encryption and integrity protections are activated together.
- **Secure Algorithm Negotiation:** Prevents downgrades to weak ciphering or integrity algorithms.
- **Session Key Update:** Secures key re-derivation and renewal during mobility or context switches.
- **Key Lifecycle Management:** Ensures secure generation, distribution, update, and revocation of cryptographic keys.

F.5 Privacy Protection

Prevents leakage of sensitive identifiers, location information, and other personal data.

• **Identifier Protection:** Ensures SUPI, SUCI, GUTI, and other permanent identifiers are never exposed in plaintext.
• **Location Privacy:** Protects against user tracking through paging, tracking area (TA) updates, or broadcast messages.
• **Secure Paging Privacy:** Prevents identity leakage during paging in multi-network environments.
• **Privacy Protection Mechanisms:** Ensures slice, subscription, and policy data remain confidential.

F.6 Network Availability & Signaling Security

Ensures the continuous availability and reliability of network services by protecting against attacks that disrupt connectivity or signaling procedures.

• **Denial-of-Service Resistance:** Protects against resource exhaustion attacks.
• **Signaling Flood Protection:** Detects and mitigates signaling flood attacks.
• **Rogue Base Station Detection:** Identifies and blocks connections to fake eNodeBs/gNodeBs.
• **Physical Layer Attack Protection:** Defends against RF jamming, replay, and signal spoofing attacks.

F.7 Interworking Security

Ensures secure and consistent behavior when UEs operate across different access technologies, network generations, and specification layers.

• **Specification Downgrade Protection:** Blocks fallback to insecure or legacy specifications.
• **Cross-Specification Interaction Security:** Prevents vulnerabilities caused by NAS, RRC, SBA, and IP interactions.
• **Specification Compatibility:** Ensures secure, consistent behavior across 4G, 5G, NSA, and SA deployments.

F.8 Threat Detection & Logging

Enables real-time anomaly detection and comprehensive monitoring to ensure system integrity.

• **Cross-Layer Threat Detection:** Monitors interactions across NAS, RRC, SBA, and user-plane layers.
• **Logging & Auditing:** Maintains secure, tamper-proof logs for incident analysis and forensic investigations.
• **Timer Behavior Verification:** Validates specification timers to prevent potential exploitation.
• **Counter Behavior Verification:** Ensures NAS, RRC, and PDCP counters are strictly monotonic, unique, and correctly synchronized between UE and network. Detects and prevents counter wrap-around, reuse, or desynchronization.

F.9 Regulatory Compliance

Ensures adherence to legal, regulatory, and industry security requirements while protecting against fraudulent activities.

• **Regulatory Compliance:** Enforces lawful intercept, data protection, and privacy regulations.
• **Billing & Fraud Protection:** Prevents manipulation of charging data and unauthorized resource usage.

F Test Case Example

Table 7 presents the example test case, which is designed to validate the `service integrity` violation identified in Node 2073.

Step	Procedure	U–M	Message	Parameter	Verdict
1	The UE is switched on.	-	-	-	-
2	The UE initiates the registration procedure.	→	REGISTRATION REQUEST	initial registration	-
3	The network accepts the registration and UE enters 5GMM-CONNECTED.	←	REGISTRATION ACCEPT	-	-
4	The lower layer sends RRC suspend indication to UE.	←	RRC Connection Suspend	-	-
5	The UE transitions to 5GMM-CONNECTED with RRC inactive indication.	-	-	-	-
6	A trigger occurs to update UE radio capability; the UE prepares a new registration.	→	REGISTRATION REQUEST and NG-RAN-RCU = "UE capability update needed"	-	-
7	The UE transitions to 5GMM-IDLE and initiates the capability update registration.	-	-	-	-
8	The attacker captures the REGISTRATION REQUEST in transit.	←	(Captured) REGISTRATION REQUEST	-	-
9	The attacker replays the REGISTRATION REQUEST to the network.	→	REGISTRATION REQUEST	-	-
10	The UE processes both messages.	-	-	-	**Fail**

Table 7: Test case for validating `service integrity` violation in Node 2073