



# SAGE: An Adaptive IoT Honeypot with FSM-Driven Protocol Emulation and GraphRAG-Powered Response Generation

SAURABH CHAMOTRA\*, Computer Science & Engineering, IIIT Guwahati, Guwahati, India

FERDOUS BARBHUIYA\*, Computer Science & Engineering, Indian Institute of Information Technology Guwahati, Guwahati, India

Increasing security threats in Internet of Things (IoT) ecosystems necessitate advanced deception mechanisms that can engage adversaries and generate realistic interactions. Traditional IoT honeypots suffer from limited protocol emulation, static response mechanisms, and susceptibility to fingerprinting, making them ineffective against sophisticated attacks. To address these challenges, this paper introduces SAGE (State-Aware Graph-Enhanced Honeypot), an adaptive IoT honeypot that integrates Finite-State Machine (FSM)-driven protocol emulation with GraphRAG-enhanced retrieval. Unlike conventional honeypots, SAGE dynamically models stateful IoT protocols, ensuring structured and realistic request-response interactions. For undefined requests, GraphRAG retrieval selects relevant protocol knowledge from knowledge graphs and integrates it with FSM-derived contextual information, enabling the Large Language Model (LLM) to generate coherent and deception-resilient responses. Additionally, a fact-checking engine and feedback mechanism refine responses, mitigating hallucinations and ensuring protocol fidelity. A key feature of SAGE is its ability to create an IoT honeypot with limited protocol knowledge while progressively evolving its emulation depth by dynamically expanding its state-response mappings based on observed adversarial interactions. Experimental evaluation demonstrates that SAGE enhances deception robustness, improves adversary engagement, and mitigates honeypot fingerprinting risks. By combining FSM-based protocol modeling, GraphRAG-enhanced retrieval, and adaptive LLM-generated responses, SAGE establishes a scalable, intelligence-driven security framework that significantly advances IoT honeypot technology.

CCS Concepts: • Computing methodologies → Artificial intelligence.

Additional Key Words and Phrases: IoT Security, Honeypots, Large Language Models (LLM), LLM-Powered IoT Honeypots, RAGs, GraphRAG, Finite State Machines (FSMs)

## 1 Introduction

The rapid expansion of the Internet of Things (IoT) has transformed industries such as smart cities, healthcare, manufacturing, and energy management by integrating edge computing, connected sensors, and autonomous decision-making [13, 35]. While these advancements enhance operational efficiency, they also introduce significant cybersecurity risks. IoT infrastructures have become prime targets for cyberattacks due to cost-effective and rapid deployments that often result in security trade-offs [8, 11, 39]. The widespread accessibility of IoT devices further exacerbates these threats, enabling attackers to exploit platforms such as Shodan [53], Censys [9], and Zoomeye [69] to identify and compromise unsecured endpoints. Millions of connected devices—including surveillance systems, industrial controllers, and medical equipment—are still vulnerable to cyber threats [36]. Unlike conventional IT breaches, IoT security compromises can lead to severe real-world consequences, such as

\*Both authors contributed equally to this research.

Authors' Contact Information: Saurabh Chamotra, Computer Science & Engineering, IIIT Guwahati, Guwahati, Assam, India; e-mail: saurabh.chamotra83@gmail.com; Ferdous Barbhuiya, Computer Science & Engineering, Indian Institute of Information Technology Guwahati, Guwahati, Assam, India; e-mail: ferdous@iiitg.ac.in.



This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

© 2025 Copyright held by the owner/author(s).

ACM 2577-6207/2025/9-ART

<https://doi.org/10.1145/3769011>

physical damage, operational disruptions, and financial losses. Several high-profile incidents underscore these risks, including the Oldsmar Water Treatment Attack (2021) [22] and cyber-attacks on critical infrastructure in Ukraine (2022) [32] and Denmark (2023) [51].

Despite widespread IoT adoption and the escalating security threats, security remains an afterthought, leaving systems highly vulnerable to exploitation [47]. Additionally, many IoT devices operate with constrained computational resources, making traditional security solutions—such as antivirus software, Endpoint Detection and Response (EDR), and Intrusion Detection Systems (IDS)—impractical due to their high processing overhead [46]. Conventional defenses primarily rely on static signatures, rendering them ineffective against zero-day exploits and sophisticated attack techniques [8, 66]. These limitations highlight the need for adaptive strategies that dynamically respond to evolving threats while addressing the resource constraints inherent in IoT environments. As traditional security measures struggle to keep pace, deception-based security mechanisms such as IoT honeypots have emerged as proactive defense solutions that engage attackers, facilitate early threat detection, and enhance threat intelligence gathering [5, 61]. By simulating real-world environments, honeypots deceive adversaries, redirect their actions, and enhance situational awareness, thereby strengthening IoT security postures [29].

However, despite their potential, developing realistic IoT honeypots presents significant challenges [17, 31]. The diversity of IoT protocols and the lack of publicly available documentation hinder accurate emulation. Without standardized specifications, researchers must conduct extensive reverse engineering to replicate device behaviors, leading to imperfect deception and an increased risk of detection by attackers [18]. Additionally, most existing IoT honeypots [15, 25, 43] lack the capability to dynamically expand protocol emulation, making them ineffective against rapidly evolving attack techniques. Traditional IoT honeypots [7, 63] often rely on passive log analysis, which prevents real-time attack detection and classification. These systems fail to actively track an attacker's progression through protocol states, making real-time labeling of attack activities difficult. Furthermore, the absence of comprehensive protocol state tracking limits honeypots' ability to generate contextually accurate responses, particularly for stateful protocols [34]. As a result, most honeypots fail to maintain realistic multi-step interactions, which reduces their ability to engage adversaries effectively [41, 60]. These shortcomings render IoT honeypots vulnerable to protocol fingerprinting, allowing attackers to easily identify and evade them. Adversaries exploit protocol inconsistencies and missing state transitions to fingerprint honeypots [58, 68]. Addressing these challenges requires a new generation of IoT honeypots that can dynamically adapt to evolving threats, maintain realistic interactions, and generate accurate protocol responses to enhance deception.

To address the limitations of traditional IoT honeypots, this paper introduces SAGE (State-Aware Graph-Enhanced Honeypot), a hybrid framework that integrates finite state machine (FSM)-based protocol modeling, large language model (LLM)-driven response generation to create a scalable, adaptive, and intelligent honeypot. Unlike conventional honeypots, SAGE emulates proprietary IoT protocols even with limited datasets and dynamically evolves its emulation depth based on attacker interactions, thereby enhancing deception fidelity and engagement depth over time. At its core, a Mealy machine [33] based FSM enables stateful and structured protocol interactions, generating deterministic responses for known request types. For requests that fall outside predefined FSM transitions, SAGE employs LLM-driven response generation, where the LLM retrieves contextual information and synthesizes state-aware responses. To mitigate hallucinations and enhance contextual accuracy, SAGE integrates GraphRAG[45] with fact-checking and a continuous feedback loop. By integrating GraphRAG with LLMs, SAGE effectively addresses the challenge of limited dataset availability, a persistent issue in developing honeypots for proprietary protocols. By integrating FSM-driven structured protocol modeling, LLM-enhanced adaptability, GraphRAG-based retrieval, SAGE achieves scalable, protocol-aware, and dynamically evolving deception mechanisms. The framework introduces several key innovations that distinguish it from existing honeypot solutions:

- **FSM-Based Protocol Modeling with LLM Augmentation:** A Mealy machine-based FSM enables structured, stateful protocol interactions, ensuring realistic responses for known request types. For requests outside FSM-defined transitions, the GraphRAG-augmented LLM synthesizes context-aware responses, improving protocol coverage and deception fidelity.
- **GraphRAG-Enhanced Retrieval for Contextual Awareness:** Unlike conventional RAG approaches, GraphRAG structures knowledge hierarchically, reinforcing protocol fidelity, enabling multi-step reasoning, and significantly reducing latency and response hallucination. This ensures greater response accuracy while maintaining coherence with protocol states.
- **Real-Time Attack Labeling with State-Specific Regular Expressions:** SAGE employs state-specific regular expressions to track adversarial activities within protocol states, enabling real-time attack labeling, classification, and behavioral analysis. This enhances situational awareness and optimizes deception strategies based on attacker interactions.

The remainder of this paper is structured as follows: Section II reviews related work, Section III outlines the problem statement and research gap, Section IV details the SAGE framework, Section V describes the prototype implementation, and Section VI presents the experimental evaluation, concluding with key findings and future research directions.

## 2 Literature Survey

The increasing scale and complexity of Internet of Things (IoT) ecosystems have exposed critical shortcomings in conventional security mechanisms such as firewalls, intrusion detection systems (IDS), and access controls [8, 46]. Firewalls, which rely on static rule sets, struggle to defend against emerging and dynamic attack vectors [39], while signature-based IDS are often ineffective in detecting previously unseen exploits [62]. These limitations are further compounded by the constrained computational and storage capacities of IoT devices, which hinder the implementation of real-time anomaly detection and robust encryption techniques [47]. As a result, traditional security solutions fall short in dynamically adapting to advanced threats, highlighting the urgent need for more proactive and intelligent defense mechanisms.

Honeypots have emerged as a proactive defense mechanism designed to overcome the limitations of traditional, passive IoT security solutions. Unlike conventional systems that focus solely on detection or prevention, honeypots are deliberately deployed as decoy targets to attract and engage attackers, capturing valuable intelligence on their behavior, tools, and techniques [18]. By simulating realistic services or devices, honeypots enable defenders to study adversary tactics in controlled environments, uncover novel threats, and bridge gaps left by static signature-based defenses. Given their diverse designs and deployment strategies, honeypots can be classified along several dimensions; however, one of the most widely recognized and practically relevant criteria is the *level of interaction* they provide. This dimension defines the degree of freedom offered to the attacker and the extent of system functionality exposed by the honeypot [17]. Based on this criterion, IoT honeypots are typically categorized into the following three types:

- **Low-Interaction Honeypots**, such as Compot [26], ThingPot [63], and Honeyd [48], provide lightweight emulation but offer limited engagement and behavioral insights.
- **High-Interaction Honeypots**, such as IoTpot [43], MimePot [7], and Honware [62], allow deeper attacker engagement, enabling the capture of rich interaction data at the expense of increased resource requirements and operational risks.
- **Medium-Interaction Honeypots**, including U-PoT [25] and HoneyThing [15], aim to balance realism and scalability but often require manual updates to accommodate new protocols and behaviors.

While these existing classes of IoT honeypots have demonstrated varying degrees of success in engaging attackers and collecting threat intelligence, several critical challenges remain unresolved [18]. One of the most

significant limitations is the inability of many honeypots to accurately emulate proprietary or undocumented protocols with high fidelity [54]. Most rely on static rule sets or manually crafted response templates, which restrict their capacity to produce contextually accurate and realistic interactions [60]. Additionally, traditional honeypots often exhibit deterministic behavior patterns that attackers can easily fingerprint and bypass [52, 58].

To address these limitations, researchers have explored finite automata-based approaches, which provide structured protocol modeling and logically consistent response generation [20, 28]. However, these methods face scalability challenges due to the well-known state explosion problem, where increasing protocol complexity results in an exponential growth in the number of required states [10]. In addition to finite automata approaches, machine learning (ML)-based techniques have been proposed to further improve honeypot adaptability and response realism. For instance, HoneyIoT [24] leverages reinforcement learning (RL) to dynamically generate high-fidelity responses based on observed attacker behavior, while IoTCandyJar [30] applies Markov Decision Processes (MDP) to optimize engagement strategies in real-time. Deep learning models, such as those implemented in NeuPot [52] and NeuralPot [54], further enhance response contextualization by learning from historical attack datasets. Despite these advancements, ML-based honeypots typically rely on large, labeled datasets and static emulation states, which limit their scalability and reduce their effectiveness.

Recent advancements in natural language processing (NLP) have introduced compelling opportunities to enhance honeypot design, particularly by addressing the dynamic and context-sensitive nature of adversarial behavior [27]. The emergence of large language models (LLMs)—including OpenAI’s GPT series, Anthropic’s Claude, and Google’s Bard [4, 21, 42]—alongside open-source releases such as Meta’s LLaMA and LLaMA2 [2], has significantly accelerated interest in leveraging LLMs for interactive deception. These models, with both cloud-based and on-premises deployment capabilities, provide a flexible foundation for cybersecurity research and applications. Recent studies have begun exploring the use of LLMs as dynamic deception engines within honeypot systems. For instance, HuntGPT [3] demonstrated that LLMs can generate real-time, protocol-agnostic responses to terminal commands across heterogeneous operating systems, enabling more realistic and flexible interaction with attackers. Building upon this foundation, several advanced honeypot architectures have emerged. AIIPot [34] integrates transformer-based models with reinforcement learning to adapt response strategies based on attacker behavior, thereby increasing engagement effectiveness. LLMPot [60] employs fine-tuned LLMs to emulate industrial protocols and device control logic, reducing dependency on static scripting. Similarly, the system proposed by Ragsdale et al. [49] utilizes LLMs to support adaptive interaction and maintain contextual coherence during adversarial engagements. Collectively, these approaches represent a paradigm shift from traditional rule-based deception mechanisms toward more autonomous, context-aware, and intelligent honeypot frameworks capable of sustaining realistic and engaging interactions with sophisticated attackers.

However, despite these advancements, existing LLM-based honeypots face several critical limitations. AIIPot relies heavily on pre-collected request-response datasets and lacks explicit protocol state management, making it vulnerable to data poisoning and ineffective against novel or stateful attacks [34]. LLMPot, while improving protocol emulation, requires task-specific retraining for each new configuration and treats protocol knowledge as flat text, limiting its ability to manage multi-step interactions or adapt dynamically to evolving attack strategies [60]. These shortcomings undermine the reliability and scalability of current LLM-based honeypots, particularly in dynamic IoT environments.

In response to these limitations, Retrieval-Augmented Generation (RAG) has emerged as a promising technique to improve LLM outputs by retrieving relevant external knowledge—such as protocol documentation, attack logs, or configuration data—during inference [12, 19]. This retrieval mechanism grounds LLM-generated responses in factual context, improving accuracy and adaptability without requiring extensive retraining [23]. However, conventional RAG implementations treat external knowledge as flat, unstructured text, limiting their ability to manage protocol state transitions or multi-step interactions. Graph-Based RAG (GraphRAG) addresses these gaps by structuring knowledge as graphs [45, 67], enabling multi-hop reasoning, protocol state management, and

enhanced context tracking. This makes GraphRAG particularly well-suited for IoT deception scenarios where maintaining stateful and coherent interactions is critical for sustaining attacker engagement.

Motivated by these insights, this paper proposes **SAGE**—a novel honeypot framework that integrates Finite State Machine (FSM)-based protocol modeling with GraphRAG-powered response generation. By combining structured protocol emulation with adaptive, context-aware response synthesis, SAGE significantly enhances the realism, consistency, and adaptability of honeypot interactions. The framework incorporates fact-checking mechanisms and iterative feedback loops [44] to minimize inaccuracies and ensure compliance with protocol specifications, while customized prompt engineering techniques [50] further refine LLM outputs to align with domain-specific constraints. Leveraging GraphRAG’s graph-based retrieval structure, SAGE supports multi-hop reasoning and protocol state management, enabling contextually consistent and realistic response generation even in adversarial scenarios where attackers actively probe for inconsistencies. To contextualize SAGE’s contribution, **Table 1** presents a comparative analysis of existing honeypot designs, categorizing them into three primary types: (i) *traditional and protocol-specific honeypots*, which rely on static rules or handcrafted emulation (e.g., Conpot, HoneyPLC, S7commTrace); (ii) *machine learning-based honeypots*, which utilize reinforcement learning (RL) or sequence models for adaptive response generation (e.g., IoTCandyJar, NeuPot); and (iii) *LLM-driven adaptive honeypots*, which leverage large language models and advanced prompting techniques for intelligent, context-aware deception (e.g., AIIPot, LLMPot, and SAGE). This classification highlights the unique positioning of SAGE as a scalable, adaptive, and effective solution for next-generation IoT deception.

Table 1. Comparison of Traditional, ML-based, and LLM-based Honeypots in IIoT Contexts

Feature	Traditional / Protocol-Specific			ML-based		LLM-based	
	Conpot	S7comTrace	HoneyPLC	IoTCandyJar	NeuPot	LLMPot	SAGE
Adaptive Learning & Protocol Expansion	✗	✗	✗	✗	✗	✗	✓
Adaptive Response Generation	✗	✗	✗	✓	✗	✓	✓
Machine Learning Integration	None	None	None	RL+Markov	Seq2Seq	LLM+FT	LLM+GraphRAG
Protocol Coverage	ICS	S7	S7+Modbus	IoT	Modbus	IoT	IIoT
Physical Process Emulation	✓	✗	✓	✗	✗	✗	✓
Attack Detection & Classification	✗	✗	✗	✗	✓	✓	✓
Real-Time Detection & Labeling	✗	✗	✗	✗	✗	✓	✓

### 3 Proposed Framework: SAGE (State-Aware Graph-Enhanced Honeypot)

This section introduces **SAGE** (State-Aware Graph-Enhanced Honeypot), a hybrid IoT honeypot framework that integrates Finite State Machine (FSM)-based protocol modeling with a GraphRAG-enhanced, large language model (LLM)-based response generation engine to support adaptive and realistic deception. The architecture consists of two principal modules: an **FSM-based Protocol Emulation Engine** for deterministic protocol execution, and an **LLM-based Response Generation Module** that leverages GraphRAG retrieval and prompt-based synthesis.

The FSM component provides structured handling of standard request-response sequences, ensuring protocol compliance for well-defined interactions. In parallel, the GraphRAG-enhanced LLM module retrieves semantically relevant information from a hierarchical knowledge graph to generate coherent responses for undefined or unexpected inputs. Operating in tandem, these modules allow SAGE to dynamically adapt its behavior and generate intelligent, protocol-consistent responses. This dual mechanism significantly reduces reliance on static response templates, enhancing adaptability and resilience against evasion strategies. By synthesizing contextually grounded and semantically accurate outputs, the system fosters deeper adversarial engagement and improves deception realism. In addition, this section presents a complexity analysis of the hierarchical knowledge graph used in GraphRAG, demonstrating its efficiency and expressiveness compared to conventional flat retrieval-augmented generation (RAG) methods. The following sections detail the design and implementation of each core component.

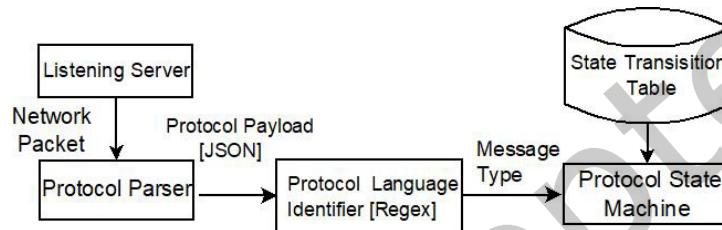


Fig. 1. FSM-Based Protocol Emulation Architecture

### 3.1 IoT Protocol Emulation Using FSM

This subsection presents a Finite State Machine (FSM)-based methodology for modeling IoT protocols, offering a structured and extensible approach to protocol emulation that addresses the limitations of traditional static request-response mappings. Static models often fail to capture the inherently stateful and complex nature of Industrial IoT (IIoT) communications [6], exhibiting poor scalability and brittleness when confronted with protocol deviations, malformed requests, or unexpected inputs [60]. In contrast, the FSM-based approach enables dynamic and accurate emulation by explicitly defining protocol states, transition logic, and deterministic outputs. It is lightweight—requiring minimal initial training data—and allows new state transitions to be added at runtime without recompilation. This flexibility enables the emulation logic to evolve based on observed interactions, supporting incremental learning and continuous refinement of protocol behavior. By expanding the state transition table modularly, the framework accommodates both known and previously unseen interactions, thereby enhancing deception realism and deepening adversarial engagement. The FSM structure further ensures robustness and scalability, even in scenarios with incomplete or evolving protocol knowledge. Figure 1 illustrates the FSM-based architecture, where incoming network packets are processed sequentially through defined state transitions to generate protocol-compliant responses. The overall workflow comprises the following core components:

- **Protocol Parser:** The Protocol Parser is responsible for extracting the payload from incoming network packets and converting it into a structured, machine-readable format such as JSON. It identifies protocol-specific headers, parses critical fields—such as function codes, subfunction identifiers, and control bytes—and ensures syntactic validity before forwarding the parsed structure to the subsequent modules. This structured output enables downstream components to operate with semantic clarity and consistency. Specifically, the output of the Protocol Parser serves as the direct input to the Protocol Language Identifier (PLI), which performs rule-based classification of messages. *Example:* Given a sample S7comm packet header: 32 01

00 00 00 08 00 00 00 01, the parser recognizes this as a Job request (ROSCTR = 0x01), extracts key fields, and transforms the raw payload into the following structured representation:

```
{
    "ProtocolID": "0x32",
    "ROSCTR": "0x01",
    "Function": "0x04",
    "Length": "0x0008"
}
```

- **Regular Expression-Based Protocol Language Identifier (PLI):** The Protocol Language Identifier (PLI) module performs early-stage message classification by applying regular expression-based pattern matching on the structured JSON output generated by the Protocol Parser. Instead of operating on raw hexadecimal streams, the PLI processes parsed protocol fields—such as "Function", "ROSCTR", and "ProtocolID"—to categorize messages according to their operational role. This classification ensures that only well-formed and semantically interpretable messages are passed to the FSM for state evaluation, thereby improving parsing robustness and minimizing erroneous transitions. Formally, the PLI is defined as:

$$PLI : \Sigma^* \rightarrow C \quad (1)$$

where  $\Sigma^*$  denotes the space of all protocol message representations (in structured JSON format), and  $C$  is the set of protocol-defined message categories. Given a structured message  $m$ , the function  $PLI(m) = c$  assigns it to a category  $c$  using predefined regular expression patterns.

**Example:** Consider the following structured output produced by the Protocol Parser:

```
{
    "ProtocolID": "0x32",
    "ROSCTR": "0x01",
    "Function": "0x04",
    "Length": "0x0008"
}
```

To classify this message, the PLI applies a rule such as:

"Function":\s\*" $0x04$ "

which maps the message to the *Read Variable* category. Similarly, to detect a System Status List (S7) request associated with reconnaissance or fingerprinting, a pattern like:

"Function":\s\*" $0x04$ ".\*"S7LID":\s\*" $0x0011$ "

can be used. These rules allow the system to tag and route messages appropriately for downstream FSM handling. A more comprehensive breakdown of S7comm message classification is provided in Section 4.

- **Protocol State Machine (PSM) Processing:** After a message is classified by the PLI, it is passed to the Protocol State Machine (PSM) for state-based evaluation. The PSM governs honeypot behavior by modeling protocol interactions using a Mealy Machine [33], where the output depends on both the current state and the input message. This structure enables precise emulation of protocol workflows and enforcement of stateful communication logic. As the PSM only processes messages that have been parsed and semantically categorized, it operates with high reliability and avoids ambiguity. The system maintains internal state throughout the interaction, ensuring protocol-compliant outputs and supporting multi-stage exchanges. Formally, the Mealy machine is defined as:

$$M = (Q, \Sigma, \Lambda, \delta, \omega, q_0) \quad (2)$$

where:

- $Q$ : Set of protocol states
- $\Sigma$ : Set of incoming classified messages
- $\Lambda$ : Set of output responses

- $\delta : Q \times \Sigma \rightarrow Q$ : State transition function
- $\omega : Q \times \Sigma \rightarrow \Lambda$ : Output function
- $q_0$ : Initial state

Transitions and responses are computed as:

$$\delta(S_i, \Sigma_x) = S_j, \quad \omega(S_i, \Sigma_x) = \Lambda_y \quad (3)$$

This formalism provides deterministic, state-aware protocol emulation, laying the foundation for modular expansion and integration with adaptive response systems in later stages.

- **State Transition Processing:** The FSM employs a predefined State Transition Table to model protocol state progression and response generation, ensuring structured and state-aware interactions. Table 2 presents a sample transition table for Siemens PLCs operating under the S7 protocol [1]. Initially, the FSM resides in the idle state ( $S_1$ ), awaiting a request. Upon receiving "ProtocolId": "0x32", it transitions to Communication Setup ( $S_2$ ) to process initialization parameters. Detecting "ROSCTR": "Userdata (7)" advances it to Authenticated ( $S_3$ ), establishing a session. In this state, memory access requests trigger an access control check, generating a permission error response for unauthorized attempts. To enhance adaptability, the Function Handler Module integrates dynamically with the FSM, enabling flexible response generation and enhancing protocol emulation accuracy.

Current State	Input Pattern (Regex)	Next State	Output Response
$S_1$	"ProtocolId": "0x32"	$S_2$	fun_handler1()
$S_2$	"ROSCTR": "Userdata\s*(7)"	$S_3$	fun_handler2()
$S_3$	"Read\s*WRITE\s*Memory"	$S_3$	fun_handler4()

Table 2. State Transition Table for S7comm Protocol with Regular Expressions

- **Protocol state handler functions** The state handler function is integral to the state transition table lookup process, dynamically generating variables and detecting state-specific attack signatures. Since vulnerabilities depend on protocol states, attackers may exploit weaknesses upon reaching specific states. To counter this, the function maps states to vulnerabilities in real time, enabling proactive threat detection and adaptive responses. Integrated with the finite state machine via the function handler module, it ensures realistic protocol emulation while distinguishing legitimate queries from malicious attempts. Upon receiving a request, it invokes security functions tailored to the detected state, enhancing system resilience. By cross-referencing states with known exploits, the state handler detects attack signatures and dynamically adjusts responses to strengthen protocol authenticity and intrusion detection.

### 3.2 Response Generation Through LLM

When the Finite-State Machine (FSM) encounters an undefined request  $R \notin \Sigma$ , the system activates the *Dynamic Function Handler*, which forwards the request  $R$ , the current state  $S$ , and relevant contextual metadata to the large language model (LLM) for processing. This transition is formally expressed as:

$$R \notin \Sigma \Rightarrow \text{LLM Activation} \quad (4)$$

The LLM then synthesizes a context-aware response  $\Lambda_{\text{LLM}}$ , which is passed through a feedback-driven validation pipeline to ensure factual accuracy and protocol compliance. Upon successful validation, the FSM transition logic is updated as follows:

$$\delta(S, R) = S_{\text{new}}, \quad \omega(S, R) = \Lambda_{\text{LLM}}, \quad \Sigma \leftarrow \Sigma \cup \{R\} \quad (5)$$

This mechanism enables the FSM to evolve semi-autonomously by incorporating previously unseen interactions without requiring manual rule definition. While the initial FSM is curated by domain experts, subsequent expansions are informed by attacker interactions and validated LLM-generated responses, facilitating continuous adaptation with minimal human intervention.

To ensure accuracy and contextual fidelity, the response generation engine integrates GraphRAG-based retrieval, structured prompt engineering, and a knowledge graph (KG) constructed from protocol logs, datasheets, and network traces. The KG encodes hierarchical relationships and protocol-specific dependencies, offering structured, device-aware context to the LLM. Retrieved knowledge is combined with session metadata and embedded into customized prompts, enabling the LLM to generate responses that are both semantically coherent and protocol-consistent. Figure 2 illustrates the three core stages of the response generation pipeline: (1) **Knowledge graph creation**, (2) **Knowledge graph querying**, and (3) **Response processing & generation**. The architecture and complexity of these components, particularly the GraphRAG module, are elaborated in the following subsections.

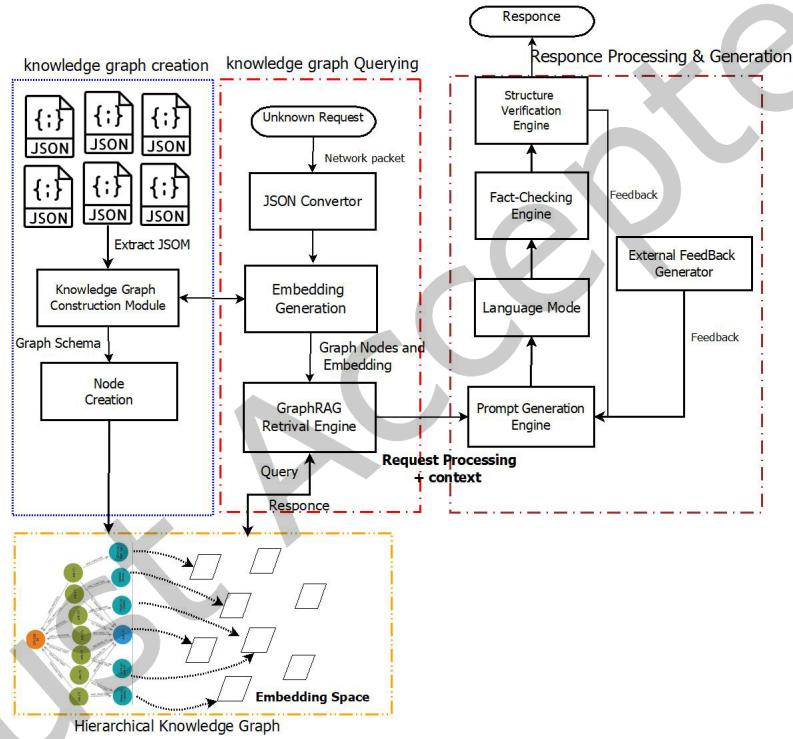


Fig. 2. LLM-Based Response Generation Framework Utilizing Knowledge Graphs with Retrieval-Augmented Generation.

**3.2.1 Knowledge Graph Creation.** The hierarchical Knowledge Graph (KG) is constructed using ground-truth data collected from both real-world ICS testbed traffic and publicly available PCAP datasets. Specifically, structured request-response pairs were extracted from a custom-deployed testbed featuring Siemens S7-300/400 PLCs and EXOR HMI devices. Further details on the data collection and testbed configuration are presented in Section 5. This diverse dataset ensures comprehensive semantic coverage of protocol behaviors and attack patterns.

The construction process begins by parsing structured logs to extract key protocol attributes—such as packet parameters, control fields, and payload contents. These attributes are encoded into vector embeddings to capture

semantic relationships between protocol entities. A tree-based hierarchical KG is then created by organizing the extracted requests as graph nodes, linking them to fields such as `PacketType`, `FunctionGroup`, and `SubFunction`. This hierarchical structure supports efficient semantic similarity matching, whereby incoming requests are compared against graph nodes using cosine similarity in embedding space to identify functionally similar protocol behaviors, even when exact matches are absent. To enhance semantic alignment, we adopt hierarchical embedding models such as HyperVec [40], which explicitly encode asymmetrical relationships between general and specific concepts. In this model, higher vector norms are assigned to generalized categories (e.g., protocol families), while lower norms represent specific instances (e.g., function codes). This norm-based encoding enables the KG to capture both functional depth and semantic proximity between protocol elements, improving the relevance of retrieved matches during query execution.

The formal implementation of the KG construction pipeline is detailed in Algorithm 1, which loads request packets from an SQLite source, generates semantic embeddings, and stores the structured knowledge graph in Neo4j [38], a graph database optimized for high-performance relational and hierarchical querying. The structured

---

**Algorithm 1** Processing Request Packets and Storing in Graph Database
 

---

```

1: Load request-response packets from SQLite.
2: data  $\leftarrow$  Execute Query(SELECT rid, req FROM req_resp)
3: Establish Neo4j database connection.
4: for each request in data do
5:   Parse JSON and extract protocol metadata.
6:   Classify packets based on protocol ID and control field.
7:   Generate embeddings from relevant parameters.
8:   Store structured data in Neo4j, linking nodes by function group and subfunction.
9: end for
```

---

KG not only supports high-fidelity response generation but also serves as the foundation for GraphRAG-based querying, which is detailed in the next subsection.

**3.2.2 Knowledge Graph Querying.** Traditional Retrieval-Augmented Generation (RAG) methods [16], which rely on flat vector similarity or key-value pair matching, often fall short when faced with syntactically diverse or previously unseen requests. These approaches lack structural awareness and are unable to model the hierarchical dependencies present in industrial control protocols.

To overcome these limitations, the proposed framework employs a Graph-based Retrieval-Augmented Generation (GraphRAG) mechanism. GraphRAG operates over a hierarchical knowledge graph (KG) that encodes protocol semantics across multiple abstraction levels. Each incoming request is parsed into structured attributes—such as `PacketType`, `FunctionGroup`, and `SubFunction`—and mapped to graph nodes using semantic embeddings. The retrieval engine initiates its search from the most specific leaf nodes and escalates to higher abstraction levels if no direct match is found, enabling graceful fallback and improved generalization. The complete retrieval pipeline is illustrated in Figure 2. Protocol logs are first converted into structured JSON and passed to the KG construction module, which organizes them into semantically annotated graph nodes. Upon receiving a new request, the system generates an embedding vector, traverses the KG semantically via the GraphRAG engine, and forwards relevant context to the LLM-based response generation module.

To concretely illustrate how GraphRAG enables structured retrieval, Figure 3 shows an example traversal using an S7comm protocol request. The system begins with a unique request identifier (`rid`) and resolves it by traversing semantic fields in hierarchical order: `PacketType`  $\rightarrow$  `FunctionGroup`  $\rightarrow$  `SubFunction`  $\rightarrow$  `SZL-ID`. This path determines whether the requested `SZL-ID` is recognized by the device, which in turn governs the response logic. If the `SZL-ID` is listed, the emulated device returns a realistic response (`Device = 1`); otherwise, it mimics a rejection (`Device = 0`). This structured decomposition ensures that even novel requests can be

interpreted through topological proximity in the graph, supporting generalization without loss of semantic fidelity.

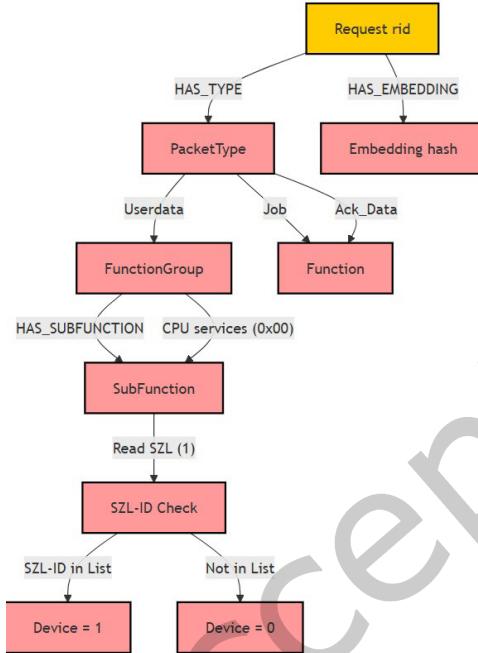


Fig. 3. Example of hierarchical traversal for an S7comm protocol request showing resolution from packet metadata to device-specific behavior.

Unlike flat RAG, which conducts global nearest-neighbor search over unstructured embeddings, GraphRAG constrains retrieval to semantically relevant subgraphs. The search begins at the most specific nodes (e.g., SubFunction, SZL-ID) and escalates only when necessary to broader categories (e.g., FunctionGroup, PacketType). This structured fallback mechanism improves retrieval precision, reduces false positives, and ensures consistent alignment with protocol behavior. The formal complexity and empirical efficiency of GraphRAG are analyzed in Section 3.2.4.

**3.2.3 Response Generation & Processing.** Figure 2 illustrates the architecture of the response generation and processing pipeline, which complements FSM-driven emulation and GraphRAG-based retrieval with adaptive, context-aware language modeling. This module is activated when incoming protocol requests fall outside the defined FSM transition table, indicating unknown or unmodeled interactions. The unresolved request, along with associated contextual metadata, is passed to the *Prompt Generation Engine*, which synthesizes structured, semantically rich prompts using candidate knowledge retrieved from the hierarchical knowledge graph (KG), including relevant graph nodes, embedding vectors, and session-specific context. These prompts are then processed by the *Language Model*, which generates candidate responses that align with the emulated device's behavior and protocol semantics. To ensure both syntactic validity and semantic accuracy, the LLM output undergoes a two-stage validation process. First, the *Structure Verification Engine* evaluates the response for protocol conformity and syntactic correctness. Next, the *Fact-Checking Engine* verifies factual consistency by cross-referencing the

generated output against trusted data sources and historical request-response logs. This dual-layered verification mitigates the risk of hallucinations and ensures high-fidelity, protocol-compliant responses. Once validated, the response is sent to the attacker, and the subsequent attacker interaction is continuously monitored by the *External Feedback Generator*. This module analyzes behavioral signals to refine prompt construction, adjust embeddings, and adapt LLM behavior in real time. Validated responses are archived in the *Request-Response Database* and may be incorporated into the FSM through a semi-supervised update mechanism.

**3.2.4 Complexity Analysis of Hierarchical Knowledge Graph in GraphRAG.** The hierarchical tree-based Knowledge Graph (KG) approach in GraphRAG improves retrieval efficiency by organizing request-response embeddings within a multi-level structure. This subsection analyzes the space and time complexity of the proposed method in comparison to conventional flat vector-based Retrieval-Augmented Generation (RAG) systems. In flat RAG, retrieval is performed over an unstructured vector space using Approximate Nearest Neighbor (ANN) algorithms such as FAISS, HNSW, or KD-Trees. The time complexity of querying is given by:

$$T_{\text{flat}}(q) = O(N^{\rho} \log N) \quad (6)$$

where  $N$  is the number of stored embeddings and  $\rho \in [0.2, 0.5]$  is an empirical factor dependent on the ANN method.

The space complexity in flat RAG is:

$$S_{\text{flat}} = O(Nd) \quad (7)$$

where  $d$  is the embedding dimension. As  $N$  grows, flat RAG suffers from increased search cost due to exhaustive similarity comparisons across the high-dimensional space.

In contrast, the hierarchical KG organizes embeddings using protocol metadata (e.g., Protocol ID, Function Group, ROSCTR), resulting in a tree of expected height:

$$h = O(\log N) \quad (8)$$

Querying the hierarchical KG involves traversing the tree to relevant subspaces, followed by local ANN-based search at the leaf nodes:

$$T_{\text{tree}}(q) = O(\log N + k \log k) \quad (9)$$

where  $k$  is the number of candidate nodes at the leaf level.

The space complexity for storing the hierarchical KG is:

$$S_{\text{tree}} = O(Nd + N \log N) \quad (10)$$

The  $O(N \log N)$  term accounts for indexing and relational metadata stored in the graph database.

The key advantage of the hierarchical approach lies in its ability to prune irrelevant branches early, reducing the effective search space and improving scalability. Table 3 summarizes the comparative complexities:

Approach	Space Complexity $S$	Retrieval Complexity $T(q)$
Flat RAG (ANN-based)	$O(Nd)$	$O(N^{\rho} \log N)$
Hierarchical KG (Tree-based)	$O(Nd + N \log N)$	$O(\log N + k \log k)$

Table 3. Comparison of Space and Time Complexity between Flat RAG and Hierarchical KG

Given that  $\log N \ll N^{\rho}$  for large  $N$ , the hierarchical KG offers superior scalability and query performance. Its effectiveness, however, depends on balanced tree construction; imbalanced structures may degrade efficiency. Additionally, graph databases incur indexing overhead, which should be accounted for in deployment settings. By combining structured traversal with localized ANN search, the hierarchical KG reduces retrieval cost while maintaining contextual and protocol-level coherence.

## 4 System Implementation

This section presents the practical implementation of the proposed adaptive honeypot framework, emphasizing its protocol emulation core, dynamic response generation capabilities, and integrated system architecture. The framework emulates Siemens Programmable Logic Controllers (PLCs) using the S7comm protocol [1], a widely adopted communication standard in the Industrial Internet of Things (IIoT) domain. Given that Siemens PLCs account for approximately 31% of the global industrial automation market [65], they represent a high-value target and are frequently subjected to cyberattacks. The proprietary nature of the S7comm protocol, coupled with limited publicly available documentation, further reinforces its suitability as a case study for demonstrating the capabilities of the proposed emulation approach. Although S7comm serves as the reference protocol in this implementation, the framework is designed to be extensible and adaptable to other proprietary or sparsely documented industrial protocols.

Figure 4 illustrates the overall workflow of the implemented system, which comprises two tightly integrated components: an **FSM emulation engine** and a **GraphRAG-powered, LLM-based response generation system**. Incoming packets are processed by a regular-expression-based Protocol Language Identifier (PLI), which extracts key fields and assigns a semantic category. If the request corresponds to a known transition in the Finite State Machine (FSM), a deterministic response is generated by invoking the associated function handler, and the response is returned to the attacker. If the request is unrecognized or malformed, control is delegated to a fallback mechanism powered by a GraphRAG-enhanced Large Language Model (LLM). The request is first logged in a request-response database and enriched with metadata such as function group, subfunction code, and contextual state information. This metadata is used to query a hierarchical knowledge graph containing protocol documentation, behavioral insights, and historical interactions. The GraphRAG retrieval engine identifies semantically relevant prior instances, which are embedded into a structured prompt by the Prompt Generation Engine. This prompt is submitted to the LLM to synthesize a contextually appropriate response. The generated response is then passed through a Fact Checker to ensure syntactic correctness and adherence to S7comm protocol semantics. If the response is validated, it is stored in the request-response database and sent to the attacker. If deemed invalid, a Feedback Engine triggers a refinement loop, updating the prompt using observed attacker behavior and re-engaging the LLM for response regeneration. To ensure clarity and a structured presentation, this section is organized into three subsections. The first subsection provides a detailed analysis of the S7comm protocol, including its architecture and communication model. The second subsection describes the modeling of S7comm interactions using a Finite State Machine (FSM)-based emulation engine. The third subsection introduces the Response Generation Engine, which leverages a GraphRAG-augmented Large Language Model (LLM) pipeline to synthesize S7comm-compliant responses for novel or unclassified attacker requests. The subsections that follow elaborate on each of these components in detail.

### 4.1 Siemens S7comm Protocol Overview

The S7comm protocol is a proprietary industrial communication protocol used in Siemens PLCs, including the S7-200 (via extensions), S7-300, S7-400, S7-1200, and S7-1500 series. It enables programming, monitoring, and control over TCP/IP networks. The protocol operates over ISO-on-TCP (RFC 1006), encapsulated within TPKT (ISO 8073 Class 0) and COTP (ISO 8073 Class 4), facilitating read/write operations, system diagnostics, and PLC mode control. S7comm is primarily utilized by engineering tools such as Siemens Step7, TIA Portal, and third-party SCADA/HMI systems for seamless integration and remote interaction with PLCs. However, its widespread adoption introduces significant security vulnerabilities, making it a prime target for adversarial exploitation [56, 59]. Operating over TCP port 102, it facilitates seamless interaction between clients and PLCs by enabling command execution, data retrieval, and system diagnostics. A connection with an S7 PLC is established through a structured sequence:

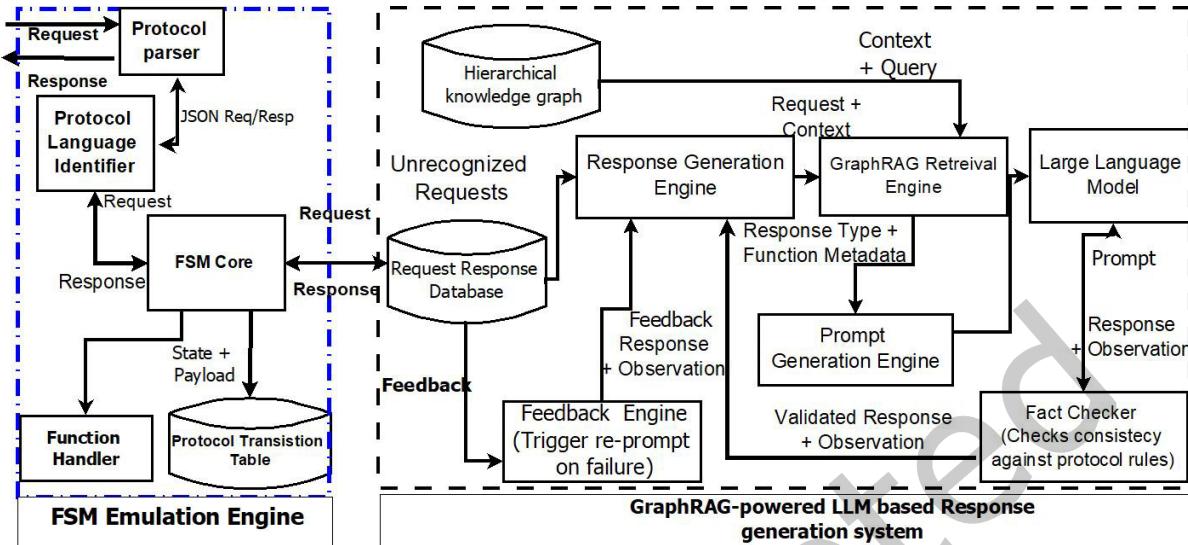


Fig. 4. End-to-End Workflow of the Proposed Adaptive Honeypot Framework

- (1) **TCP Handshake** – standard three-way handshake (SYN, SYN-ACK, ACK) establishes a TCP connection.
- (2) **PDU Size Negotiation** – The *Connection-Oriented Transport Protocol (COTP)* negotiates the *Protocol Data Unit (PDU)* size, defining the maximum allowable message length.
- (3) **S7 Communication Setup** – The client initiates the *S7comm session* (`s7comm.param.func = 0xf0`, setup communication) by exchanging necessary configuration parameters.
- (4) **Data Exchange** – Once communication is established, the client performs *read/write operations* using predefined S7comm function codes to access or modify PLC memory and retrieve system diagnostics.

Figure 5 illustrates the packet exchange sequence for S7comm connection establishment. Once the COTP connection is established, S7comm packets are transmitted within COTP frames, where the COTP PDU type is set to `0x0F` Data. An S7 PDU (Protocol Data Unit) consists of three key components:

- (1) **Header** – 12 bytes in length, containing metadata such as PDU length, PDU reference, and message type.
- (2) **Parameters** – Structure varies based on message type and PDU function.
- (3) **Data** – Payload relevant to the specific operation.

Figure 6a presents the structure of the S7comm header, while the parameter section, shown in Figure 6b, governs command execution through fields such as Function Code, MAX AMQ Caller, MAX AMQ Callee, and PDU Length. These fields define connection limits, packet segmentation, and function execution. The Function Code plays a critical role in structuring communication by classifying requests into:

- **Read and Write Operations** – Retrieve or modify PLC memory.
- **PLC Control** – Start/stop execution modes.
- **Diagnostics and System Information** – Gather system status and hardware details.
- **Block Operations** – Manage program and data blocks.

The hierarchical representation in Figure 7 illustrates key values and their placement within the protocol structure. These values serve as the blueprint for designing the *Knowledge Graph (KG)* used to enhance Large Language Model (LLM)-driven honeypot responses. The GraphRAG retrieval model developed in this study leverages these function codes and key protocol parameters to determine appropriate processing paths. These

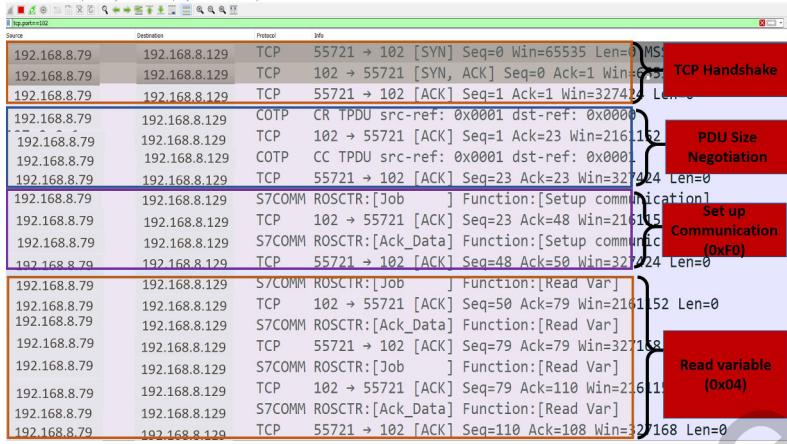


Fig. 5. S7comm Connection Sequence

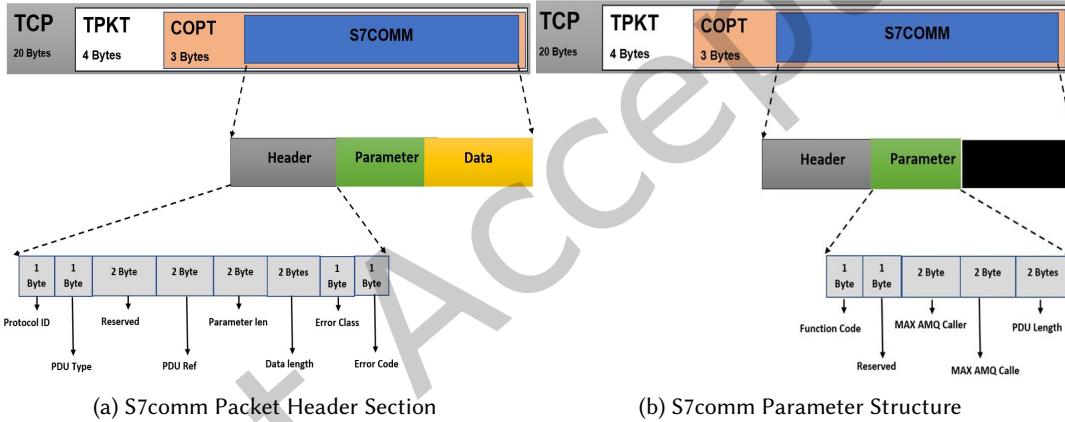


Fig. 6. S7comm Packet Structure

values, extracted from incoming packets, define request categories that guide knowledge retrieval from the graph database. Simultaneously, the Finite-State Machine (FSM) tracks state transitions based on these values, ensuring that generated responses align with expected protocol behaviors.

#### 4.2 S7comm Protocol Emulation Using FSM

The FSM Emulation Engine enables structured, stateful interaction with attacker requests by modeling Siemens S7comm protocol behavior using a Mealy Machine abstraction. As shown in Figure 4, parsed input from the Protocol Parser is first classified by the Protocol Language Identifier (PLI), which applies regular expressions to assign semantic labels such as memory access, block upload, or authentication. The tagged message is then processed by the FSM Core, which determines the next state and corresponding output based on the current state and input. For recognized interactions, the appropriate function handler generates a deterministic response.

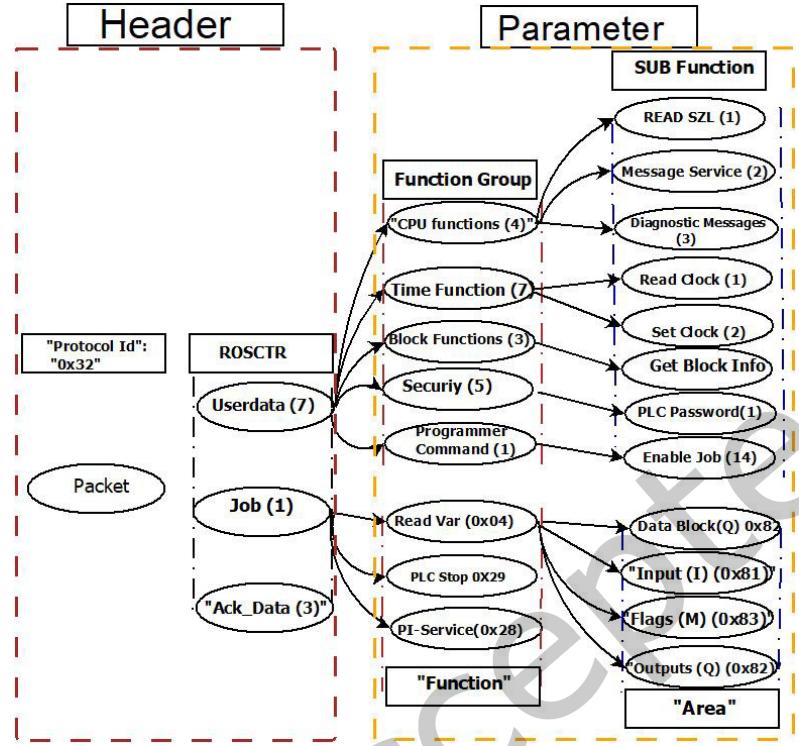


Fig. 7. Packet Types and Function Code

Unrecognized or malformed requests are forwarded to the Response Generation Engine for LLM-based synthesis. The following subsections detail the key components of the FSM Emulation Engine.

- (1) **Protocol Language Identifier(PLI):** This module classifies incoming packets using predefined regular expressions and extracts key S7comm parameters such as function codes, subfunction codes, and data block identifiers. These expressions determine the correct transition in the FSM. An example pattern is:

`^(0x32)\s+(0xXX)\s+(0xYY)\s+(0xZZ)$`

where:

- 0x32 represents the protocol ID for S7comm.
- 0xXX indicates the function group (e.g., CPU functions, Security, Block Functions).
- 0xYY specifies the subfunction code (e.g., Read SIZL, Set Clock, PLC Password).
- 0xZZ represents an additional parameter relevant to the request.

This module generalizes well across S7comm message types, enabling consistent classification despite minor dynamic variations.

- (2) **Finite State Machine Model:** The FSM is implemented as a Mealy machine, where state transitions depend on both the current state and the received input. It follows the S7comm communication flow and includes states representing different protocol stages. Table 5 presents the defined transitions. Formally, the Mealy machine is represented as the tuple  $M = (S, \Sigma, \Lambda, \delta, \omega, S_0)$ , where:
  - $S$  is a finite set of states:  $\{S1, S2, S3, S4, S5, S6\}$ .

Parameter	Description	Example Value
Protocol ID	Identifies S7comm protocol	0x32
Function Group	Major category of request	0x07 (Time Function)
Subfunction	Specific action in function group	0x01 (Read Clock)
Additional Data	Extra parameter values	0x0001 (Clock Index)

Table 4. Protocol Language Identifier Parameters

- $\Sigma$  is the input alphabet, representing S7comm request types.
- $\Lambda$  is the output alphabet, representing protocol responses.
- $\delta : S \times \Sigma \rightarrow S$  is the state transition function.
- $\omega : S \times \Sigma \rightarrow \Lambda$  is the output function.
- $S_0$  is the initial state, set to **S1** (Connection Setup).

Current State	Input ( $\Sigma$ )	Next State ( $\delta$ )	Output ( $\omega$ )	Description
S1	COTP Connection Request (CR)	S2	F1_Handler()	COTP Connection Confirm (CC)
S2	S7comm Job Setup	S3	F2_Handler()	Send Setup Acknowledgment
S3	PLC Password Request	S4	F3_Handler()	Authentication Success/Failure
S4	Read Memory Request	S4	F4_Handler()	Return Memory Data/Error
S4	Write Memory Request	S4	F5_Handler()	Confirm Write Success/Error
S4	Upload Block Request	S5	F6_Handler()	Return Block Data
S5	Download Block Request	S6	F7_Handler()	Confirm Download Success

Table 5. S7comm Protocol Transition Table

(3) **Protocol Transition Table(PTT)** : This table maps each input request to a corresponding next state and response, ensuring protocol-compliant interactions. By capturing valid state transitions, it supports deeper behavioral modeling and detection of multi-step adversarial strategies. The table structure links the current state ( $S$ ), input ( $\Sigma$ ), next state ( $S'$ ), and associated function handler ( $F\_Handler$ ). Regular expressions assist in labeling and classifying incoming messages, improving adaptability and supporting real-time engagement. This modular design ensures scalability while maintaining fidelity in Industrial Control System (ICS) emulation.

(4) **Function Handler:** This module dynamically processes protocol-specific parameters and enhances attack detection using regular-expression-based message classification. It leverages Python's dynamic function invocation to load new handlers without recompilation, thereby ensuring modularity and extensibility. The function handler also supports detection of adversarial behaviors such as fingerprinting, Denial-of-Service (DoS) attempts, and unauthorized control actions. Examples of regex-based attack detection include:

- **SZL ID Fingerprinting (Honeypot detection):**

```
^03 00 [0-9A-F]{2} 02 F0 80 32 01 00 00 00 00 00 00 00 00 01 [0-9A-F]{2}
```

Detects requests targeting SZL\_ID = 0x0011, commonly used to identify honeypots.

- **Denial-of-Service (DoS) via System Shutdown:**

```
^03 00 [0-9A-F]{2} 02 F0 80 32 01 [0-9A-F]{2} 29 00 00 00 00 01
```

Matches attempts to shut down the PLC.

- **PLC Stop Command (Unauthorized Control):**

```
^03 00 [0-9A-F]{2} 02 F0 80 32 01 [0-9A-F]{2} 28 00 00 00 00 01
```

Detects malicious attempts to stop the PLC programmatically.  
By incorporating these detection rules, the honeypot can identify and respond to known attack signatures in real-time.

### 4.3 Response Generation Engine

The *Response Generation Engine* is responsible for handling S7comm protocol requests that fall outside the deterministic transition rules of the FSM Emulation Engine. When the FSM Core encounters an unrecognized request—typically due to novel exploit attempts or malformed payloads—it logs the request in the Request-Response Database and delegates further handling to the Response Generation Engine. The full lifecycle of such a request is illustrated in Figure 4, which captures the integrated operation of deterministic FSM logic and adaptive LLM-based synthesis. The core components of this engine are detailed below:

- **Knowledge Graph (KG) Hierarchical Tree:** The IIoT protocol emulation, structured relationships between protocol types, function groups, S7-ID, and device-specific rules are critical for generating valid responses. GraphRAG addresses this need by leveraging Neo4j-based graph retrieval, ensuring that responses adhere to S7comm protocol specifications. To ensure structured representation and efficient retrieval, we processed and transformed S7comm protocol traffic from raw network captures (PCAPs) to Neo4j. The process involved:
  - Extracting S7comm requests from PCAP files.
  - Parsing packets into JSON format with Protocol ID, ROSCTR, Function Groups, S7-ID, and Parameter Sections.
  - Ingesting structured request packets into Neo4j using a well-defined graph schema.
 The Knowledge Graph Schema represents protocol interactions, function hierarchies, and device dependencies in a structured manner. This improves the search time and accuracy.
- **GraphRAG Retrieval:** The GraphRAG Retrieval Module enhances S7comm protocol emulation by enabling context-aware response synthesis through a structured Knowledge Graph (KG). Traditional exact-match retrieval fails with unknown or semantically similar S7comm requests, necessitating hierarchical embedding-based retrieval for adaptive protocol emulation. When an incoming JSON request lacks a known Finite-State Machine (FSM) transition, the system extracts protocol metadata, including Packet Type, Function Group, and Subfunction Identifiers, and applies a structured retrieval strategy, as shown in Table 6. Each request is vectorized into an embedding and matched against stored embeddings in the

Table 6. GraphRAG Hierarchical Retrieval Strategy

Retrieval Level	Description
Level 1: Function Group & Sub function Match	Attempts to retrieve responses with an exact match on both function group and sub function.
Level 2: Function Group-Level Retrieval	If no exact match is found, the system retrieves responses matching only the function group.
Level 3: Packet Type-Level Fallback	If function-based retrieval fails, the system retrieves interactions belonging to the same packet type.

KG using cosine similarity-based nearest neighbor search, ensuring retrieval of the most semantically relevant responses. The hierarchical KG structure improves retrieval efficiency compared to flat ANN-based Retrieval-Augmented Generation (RAG), which operates at  $O(N^p \log N)$  complexity. The tree-structured KG approach, incorporating structured query constraints before vector-based retrieval, reduces complexity to  $O(\log N + k \log k)$ , lowering computational overhead while maintaining accuracy. Additionally, the

flat ANN-based RAG model has a space complexity of  $O(Nd)$ , whereas the KG-based hierarchical model introduces an additional  $O(N \log N)$  term for structured relationships, yielding a total complexity of  $O(Nd + N \log N)$ .

- **Fact Checker and Structure Verifier:** Validates the accuracy of generated responses by cross-referencing them with known facts and protocol specifications. Fact-checking LLM chain [12] to ensure correctness and reduce hallucinations. This validation step verifies whether the response contains all necessary sections and follows the correct format. If the fact-checker identifies any issues, the system regenerates the response until it meets the required standards. Incorporating this verification step ensures that the honeypot's responses are not only accurate but also indistinguishable from those of real systems, further enhancing the honeypot's realism and effectiveness.
- **Advanced Prompting Techniques:** The GraphRAG approach for S7comm response generation bridges the gap between structured knowledge representation and LLM-driven reasoning. By leveraging a Neo4j-based Knowledge Graph (KG), the system ensures that generated responses are contextually accurate, device-aware, and protocol-compliant. This is achieved through **prompt engineering**, which dynamically constructs input templates incorporating operational metadata, state transitions, and protocol-specific fields. To support realistic response generation, especially for device-specific and previously unseen requests, we employ two complementary prompting strategies:

Table 7. Illustration of Contextual and Instruction Prompting for S7comm Response Generation

Prompting Type	Example Content
Contextual Prompting	<ul style="list-style-type: none"> <li>• Previous FSM State: S3 (Authentication Completed)</li> <li>• Previous Request: { "function": "Read SZL", "S7_ID": "0x0011", "Index": "0x0000" }</li> <li>• Previous Response: { "Header": { "ROSCTR": "0x07", "PDULength": "0x002C" }, "Parameter": { "Function": "Read SZL" }, "Data": { ... } }</li> <li>• Device Profile: S7-400 CPU 414-3 DP (Firmware v3.2)</li> <li>• Current Request: { "function": "Read Clock", "FunctionCode": "0x07", "Subfunction": "0x01", "Target": "System Time" }</li> <li>• Protocol Metadata (from Neo4j): Request Type: Job Function Group: 0x07 (Time Function) Subfunction: 0x01 (Read Clock)</li> </ul>
Instruction Prompting	<p>"You are emulating a Siemens S7-400 PLC currently in state S4. The client has sent a 'Read Clock' request (Function Group 0x07, Subfunction 0x01). This request is of type Job. Generate a valid S7comm response with the following structure:</p> <p>(1) <b>Header:</b> include 'ROSCTR', 'PDUlength';  (2) <b>Parameter:</b> include 'Function', 'Subfunction', and response class;  (3) <b>Data:</b> return system time encoded in 7-byte format (BCD-coded). Ensure compliance with S7comm format, consistency with the emulated device, and proper mapping of protocol roles."</p>

- **Contextual Prompting:** This strategy conditions the LLM on recent interaction history and extracted protocol metadata [57]. Specifically, the prompt includes:
  - \* The FSM state and the last request-response pair (e.g., state S3, *req1\_json*, *o\_resp1\_json*).
  - \* The device profile being emulated (e.g., S7-400 CPU 414-3 DP, with firmware version).
  - \* The current request packet.
  - \* Parsed protocol metadata extracted using Neo4j graph queries—this includes:
    - **Request Type:** e.g., Job, Ack\_Data, Userdata
    - **Function Group:** e.g., 0x07 (Time Function)
    - **Subfunction Code:** e.g., 0x01 (Read Clock)

These structured fields help the LLM accurately determine the type of response to be generated, guide the format (e.g., job response vs. acknowledgment), and align outputs with protocol expectations.

- **Instruction Prompting:** Building upon the context, this strategy issues explicit formatting instructions to the model. It defines the output structure in terms of required sections—**Header**, **Parameter**, and **Data**—and guides the model to return responses in S7comm-compliant order [50]. Instruction prompting ensures field coverage, correct function encoding, and reduction of hallucinated fields.

Table 7 presents an illustrative prompt that combines FSM state, device profile, protocol metadata, and structured output instructions to enable accurate response synthesis.

## 5 Experimentation and Results

This section presents the experimental setup and evaluation results of the proposed honeypot framework. The experiments were designed to assess key aspects, including protocol emulation accuracy, retrieval efficiency, response generation quality, and comparative performance against existing honeypot solutions. A summary of the conducted experiments is outlined below:

- *Experiment I: Protocol Emulation Accuracy* – Evaluates the fidelity of finite state machine (FSM)-based protocol emulation by comparing generated responses with real Siemens PLC interactions.
- *Experiment II: Comparison of GraphRAG and Standard RAG for Retrieval Efficiency* – Analyzes the effectiveness of GraphRAG in improving response relevance and retrieval speed compared to traditional retrieval-augmented generation (RAG) techniques.
- *Experiment III: Evaluating LLM Response Generation in a RAG-based honeypot framework* – Assesses various LLM models based on response coherence, latency, and contextual accuracy when handling unknown protocol queries.
- *Experiment IV: Benchmarking Against AIIPot (BERT-Based Honeypot)* – Compares the adaptive IoT honeypot framework with AIIPot, a BERT-based honeypot, to evaluate deception effectiveness and adversary engagement.
- *Experiment V: Performance Comparison with Conpot* – Examines the overall performance, scalability, and deception capabilities of the proposed framework relative to Conpot, a widely used IoT honeypot.

One of the key challenges in honeypot research is the limited availability of publicly accessible proprietary protocol datasets. To address this, a comprehensive dataset of S7comm interactions was developed by integrating real-world testbed captures with publicly available PCAP datasets. The collected data was systematically structured and stored in a relational database, serving as a foundation for protocol emulation and attack analysis. Further details on the test setup, data sources, and database structure are provided in subsequent sections.

- *Testbed Setup:* A dedicated testbed was deployed using Siemens SIMATIC S7-300/400 PLCs, programmed via STEP 7 in the TIA Portal suite, along with an EXOR Esmart10 HMI, a relay module control board, and a managed switch in a Class C local network. Designed for ICS security research, it focuses on Siemens S7-300 PLC-based environments and enables dataset generation for intrusion and anomaly detection. The Siemens S7-300 PLC (IP: 192.168.8.129) acts as the central control unit, interfacing with an EXOR Esmart10 HMI (IP: 192.168.8.25) for real-time monitoring, while a managed switch connects components, including an engineering station (IP: 192.168.8.79) for programming and configuration. The PLC communicates with a relay module control board, controlling an electric motor and other actuators, with physical buttons and emergency stop switches ensuring manual control and safety. This setup simulates real-world ICS network traffic, aiding cybersecurity research. Figure 8 illustrates the testbed architecture, highlighting network interactions and component integration.
- *Public PCAP Data:* To enhance dataset diversity, publicly available PCAP datasets were incorporated, including:

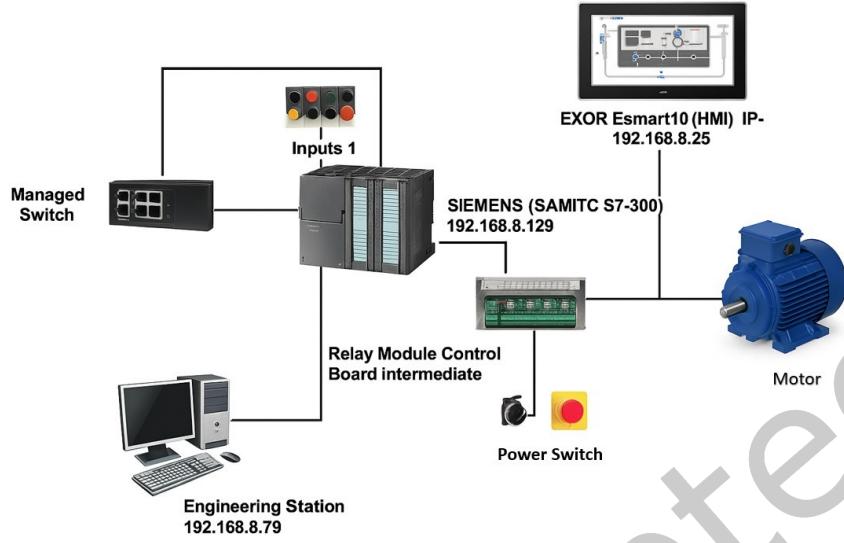


Fig. 8. Testbed Architecture for Experimental Network Deployment

- Wireshark Sample Captures [64]
- ICS/SCADA PCAPs by Jason Smith [55]
- ICS-PCAP Collection by Emre Ekin [14]

This combined dataset serves as a valuable resource for analyzing S7comm interactions, modeling protocol states, and improving honeypot deception capabilities. The following subsections present detailed experimental results, providing insights into the efficiency, adaptability, and robustness of the proposed framework.

### 5.1 Experiment I: Protocol Emulation Accuracy

The objective of this experiment is to evaluate whether FSM-based S7comm emulation accurately replicates real Siemens PLC responses, ensuring the honeypot remains indistinguishable from actual devices during adversarial interactions. The evaluation focuses on:

- Protocol Compliance – Assessing byte-level fidelity between honeypot and real PLC responses.
- Response Timing – Comparing response latency to detect timing discrepancies.

The testbed consists of two parallel environments: one with a real Siemens S7-300 PLC and another with the FSM-based S7comm honeypot. The real PLC setup includes an S7-300 PLC (IP: 192.168.8.129), an engineering workstation running STEP 7/TIA Portal for packet capture, and an EXOR Esmart10 HMI for real-time monitoring. The honeypot operates in a virtualized environment, processing S7comm packets and generating protocol-compliant responses, with a packet logger for post-analysis. Honeypot accuracy is assessed using four performance metrics:

- (1) Protocol Compliance – Measures byte-level similarity between real and honeypot responses.
- (2) Semantic Similarity – Evaluates textual consistency using BLEU scores.
- (3) Critical Field Accuracy – Verifies correctness of function codes and identifiers.
- (4) Dynamic Field Deviation – Quantifies variations in dynamic response fields for realism.

These metrics collectively evaluate the honeypot's fidelity in emulating Siemens PLC communication. The accuracy of the FSM-based S7comm honeypot is evaluated using key protocol emulation metrics. Figure 9

presents a Box Plot illustrating the distribution of these metrics across various test cases, offering insights into how closely the honeypot replicates real PLC responses. Protocol Compliance (%) metric shows a high median value, indicating structural alignment between honeypot-generated responses and real PLC packets. However, Semantic Similarity (%) shows greater variance, suggesting inconsistencies in textual response formatting. The observed variance in the Semantic Similarity metric (Figure 9) arises from differences in textual response formatting between the real Siemens S7-300 PLC and the FSM-based honeypot when handling edge-case protocol interactions. These discrepancies were particularly pronounced for rarely used S7 IDs such as 0x0174 (module status), 0x0C91 (time synchronization), and 0x0132 (time system diagnostics), where real PLCs returned dynamic, device-specific strings—often including timestamps, module identifiers, or internal status codes. In the honeypot environment, responses to such queries were generated using LLM-based synthesis guided by GraphRAG retrieval. Although semantically accurate, these generated responses often varied in lexical form, formatting conventions (e.g., date/time styles, numeric precision), and field ordering compared to those from the real PLC, resulting in lower BLEU-based similarity scores. Additional variance emerged in cases involving malformed or fuzzed S7comm packets, where the real PLC produced specific diagnostic responses while the honeypot returned generalized fallback messages synthesized from prompt templates. Furthermore, non-deterministic elements in the LLM generation process—such as temperature-controlled sampling—led to slight phrasing inconsistencies even for logically equivalent responses. These factors collectively contributed to the wider distribution observed in semantic similarity, highlighting the challenge of faithfully replicating dynamic textual fields in high-fidelity protocol emulation. Critical Field Accuracy (%) remains at 100%, confirming that essential protocol fields, such as function codes and identifiers, are faithfully replicated. Similarly, Dynamic Field Deviation (%) remains at 100%, ensuring consistency in dynamically changing response fields. Table 8 provides a statistical summary of these metrics, further reinforcing the observed trends. Figure 10 presents the latency distribution of honeypot responses,

Table 8. Statistical summary of protocol emulation accuracy metrics.

Statistic	Protocol Compliance (%)	Structural Integrity (%)	Critical Field Accuracy (%)	Semantic Similarity (%)	Dynamic Field Deviation (%)
Count	105.000	105.000	105.000	105.000	105.000
Mean	88.052	99.686	99.295	79.121	99.419
Std Dev	14.238	0.858	2.787	24.625	2.594
Min	50.699	95.000	79.000	7.862	79.000
25% (Q1)	82.772	100.000	100.000	69.206	100.000
50% (Q2)	90.702	100.000	100.000	89.408	100.000
75% (Q3)	99.766	100.000	100.000	95.674	100.000
Max	100.000	100.000	100.000	100.000	100.000

offering insights into timing consistency. The median response latency is approximately 20–25 ms, with an inter-quartile range (IQR) of 10–35 ms. However, occasional outliers exceed 100 ms, indicating intermittent delays.

The experiment confirms that the FSM-based S7comm honeypot effectively replicates real Programmable Logic Controller (PLC) behavior, achieving high protocol compliance and accurate reproduction of critical field values. However, variations in semantic similarity and structural accuracy indicate areas requiring further optimization, particularly in handling edge-case protocol interactions and dynamic state transitions. Response latency measurements validate the honeypot's ability to maintain low and stable response times, demonstrating its feasibility for real-time adversary engagement. However, to achieve complete indistinguishability from real PLCs, deploying the system on higher-performance hardware would be necessary to further reduce latency fluctuations and support real-time processing. With the current computational resources, the response generation system operates efficiently in offline mode, where responses are precomputed and retrieved dynamically, ensuring consistency while minimizing processing overhead.

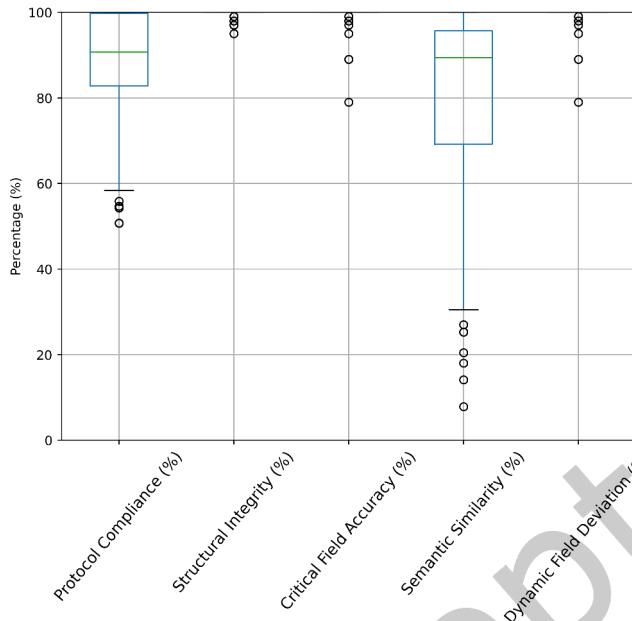


Fig. 9. Box plot of protocol emulation accuracy metrics.

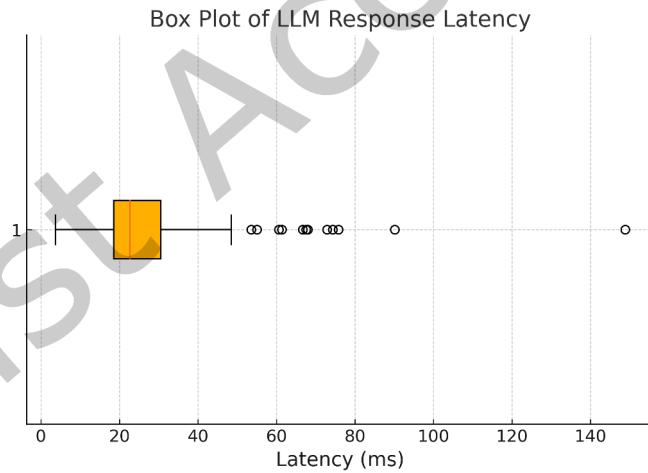


Fig. 10. Box plot of response latency distribution.

## 5.2 Experiment II: Comparison of GraphRAG and RAG for Retrieval Efficiency and Accuracy

The objective of this experiment is to evaluate the efficiency and accuracy of GraphRAG, implemented using Neo4j Graph Database, compared to RAG, implemented using FAISS. The experiment aims to determine which retrieval

approach provides faster response times and more precise contextual accuracy in IoT honeypot applications. Specifically, the study measures:

- *Retrieval Efficiency* – Assessing how quickly responses are retrieved.
- *Retrieval Accuracy* – Evaluating the alignment of retrieved responses with the expected ground truth.
- *Scalability* – Examining the stability of retrieval performance under different workloads.

The experiment processes 3,000 request-response pairs against a 200-pair knowledge base using both retrieval mechanisms. GraphRAG employs Neo4j's hierarchical graph-based querying, while RAG uses FAISS with a flat index for vector search. The experiment is conducted on a high-performance system with an Intel Xeon Gold 6338 processor, 128GB RAM, an NVMe SSD, and an NVIDIA A100 GPU. Performance evaluation is based on key retrieval parameters, including retrieval time, indexing time, and memory usage, to optimize real-time response generation. The retrieval mechanisms are evaluated using the following key metrics:

- **Latency (s)**: Measures the response time required to fetch a relevant response.
- **Retrieval Efficiency**: Quantifies retrieval speed, defined as the inverse of latency.
- **BLEU Score**: Evaluates precision by comparing n-gram overlap with reference responses.
- **METEOR Score**: Measures semantic similarity, incorporating synonym matching and stemming.
- **ROUGE Score**: Assesses recall-based relevance, capturing response completeness.

Table 9 compares retrieval efficiency and accuracy between GraphRAG and RAG. The results demonstrate that GraphRAG significantly outperforms RAG in all key performance metrics.

Table 9. Retrieval Efficiency and Accuracy Comparison Between GraphRAG and RAG

Metric	GraphRAG (Mean)	GraphRAG (Variance)	RAG (Mean)	RAG (Variance)	p-value
Retrieval Efficiency	16.95	10.36	5.97	1.36	< 0.0001
Latency (s)	0.08	0.02	0.42	0.06	< 0.0001
BLEU Score	0.9026	0.0196	0.8715	0.0579	< 0.0001
METEOR Score	0.9478	0.0065	0.9101	0.0303	< 0.0001
ROUGE Score	0.9432	0.0070	0.9056	0.0286	< 0.0001

*Retrieval Efficiency*: GraphRAG achieves significantly higher retrieval efficiency (16.95 vs. 5.97) and lower latency (0.08s vs. 0.42s,  $p < 0.0001$ ), confirming its superior speed in response generation. Its lower variance also ensures more stable performance.

*Retrieval Accuracy*: Across BLEU (0.9026), METEOR (0.9478), and ROUGE (0.9432) scores, GraphRAG consistently outperforms RAG, demonstrating enhanced response precision and semantic relevance.

*Statistical Analysis*. The ANOVA test results in Table 10 confirm a statistically significant difference between GraphRAG and RAG across all evaluated metrics. High F-statistic values, particularly for latency (3846.86,  $p < 0.0001$ ), reinforce GraphRAG's superior retrieval speed. Similarly, BLEU (601.40), METEOR (834.63), and ROUGE (625.33) scores validate its enhanced response accuracy.

Table 10. ANOVA Test Results Comparing GraphRAG and RAG

Metric	F-Statistic	P-Value	Conclusion
Latency	3846.86	< 0.0001	Significant Difference
BLEU	601.40	< 0.0001	Significant Difference
METEOR	834.63	< 0.0001	Significant Difference
ROUGE	625.33	< 0.0001	Significant Difference

The results demonstrate that GraphRAG significantly outperforms RAG in both retrieval efficiency and accuracy. With a mean retrieval efficiency nearly three times higher, GraphRAG ensures faster and more efficient response generation. Additionally, it achieves higher BLEU, METEOR, and ROUGE scores with lower variance, ensuring greater accuracy and consistency in retrieval quality. The ANOVA test results confirm statistical significance, indicating that these improvements are not random but rather a fundamental advantage of GraphRAG over RAG.

### 5.3 Experiment III: Evaluating LLM Response Generation in a RAG-based honeypot framework

This experiment evaluates the response generation capabilities of different Large Language Models (LLMs) in a Retrieval-Augmented Generation (RAG) framework for industrial honeypots. The evaluation focuses on:

- **Linguistic accuracy and relevance**, evaluated using BLEU and ROUGE scores.
- **Precision, Recall, and F1-score** to assess retrieval accuracy and completeness.
- **Latency**, to measure trade-offs between model performance and computational overhead.

Selecting an optimal LLM is critical for effective adversarial deception in honeypots. The evaluation considers four LLM architectures: Mixtral-8x7B-32768, Llama3-70B-8192, Llama3-8B-8192, and Gemma2-9B-IT, using a dataset of industrial honeypot request-response pairs. The retrieval process is powered by Neo4j's knowledge graph, ensuring structured query-based retrieval.

- **BLEU Score: Response Fluency and Precision** – The BLEU score, shown in Figure 11a, evaluates linguistic fluency and alignment with ground truth responses. Llama3-70B-8192 achieves the highest BLEU score, indicating superior accuracy. Mixtral-8x7B-32768 and Llama3-8B-8192 perform competitively, but Gemma2-9B-IT exhibits the lowest median score and higher variance.
- **Precision, Recall, and F1-Score: Response Accuracy** – Figures 11b, 11c, and 11d show the precision, recall, and F1-score distributions, which evaluate retrieval accuracy. While all models achieve high precision, Mixtral-8x7B shows greater variance due to outliers. Llama3-70B and Llama3-8B excel in recall, ensuring more comprehensive retrieval for coherent honeypot interactions. Llama3-70B-8192 attains the highest F1-score, confirming its superior balance between precision and recall.
- **Latency** Figure 11e compares the time delay (in seconds) of four language models—Mistral, LLaMA-7B, LLaMA-8B, and Gemma. The LLaMA-7B model exhibits the highest latency with a wider variance, while Mistral, LLaMA-8B, and Gemma demonstrate comparatively lower and more consistent response times. Outliers in each model highlight occasional deviations in performance, likely due to variable processing loads or system conditions.

This experiment demonstrates that, LLaMA3-70B-8192 offers the most reliable and consistent performance for industrial honeypots, while LLaMA3-8B-8192 balances accuracy with efficiency for real-time use. Mixtral-8x7B-32768 shows variable latency, reducing its predictability, and Gemma2-9B-IT, though fast, lacks precision for complex interactions.

### 5.4 IV Comparative Analysis of GraphRAG based SAGE with BERT-based AIIPoT

The AIIPoT [34] system utilizes BERT-based retrieval to generate responses for ICS protocol queries, leveraging transformer-based contextual embeddings and semantic similarity. While effective, BERT's static retrieval limits adaptability to unseen requests. GraphRAG (Retrieval-Augmented Generation using Graphs) addresses this by incorporating graph-based knowledge structures, enabling context-aware retrieval and response generation. Unlike BERT's direct embedding-based matching, GraphRAG captures semantic relationships between protocol components, ensuring greater relevance and compliance. To compare both approaches, an experiment was conducted with 3,000 ICS protocol queries, evaluating responses on protocol compliance and semantic similarity. The results demonstrate GraphRAG's advantage in dynamic response generation and context-aware retrieval, mitigating BERT's adaptability constraints. The Box Plots Figure 12 compares the Protocol Compliance (%) and

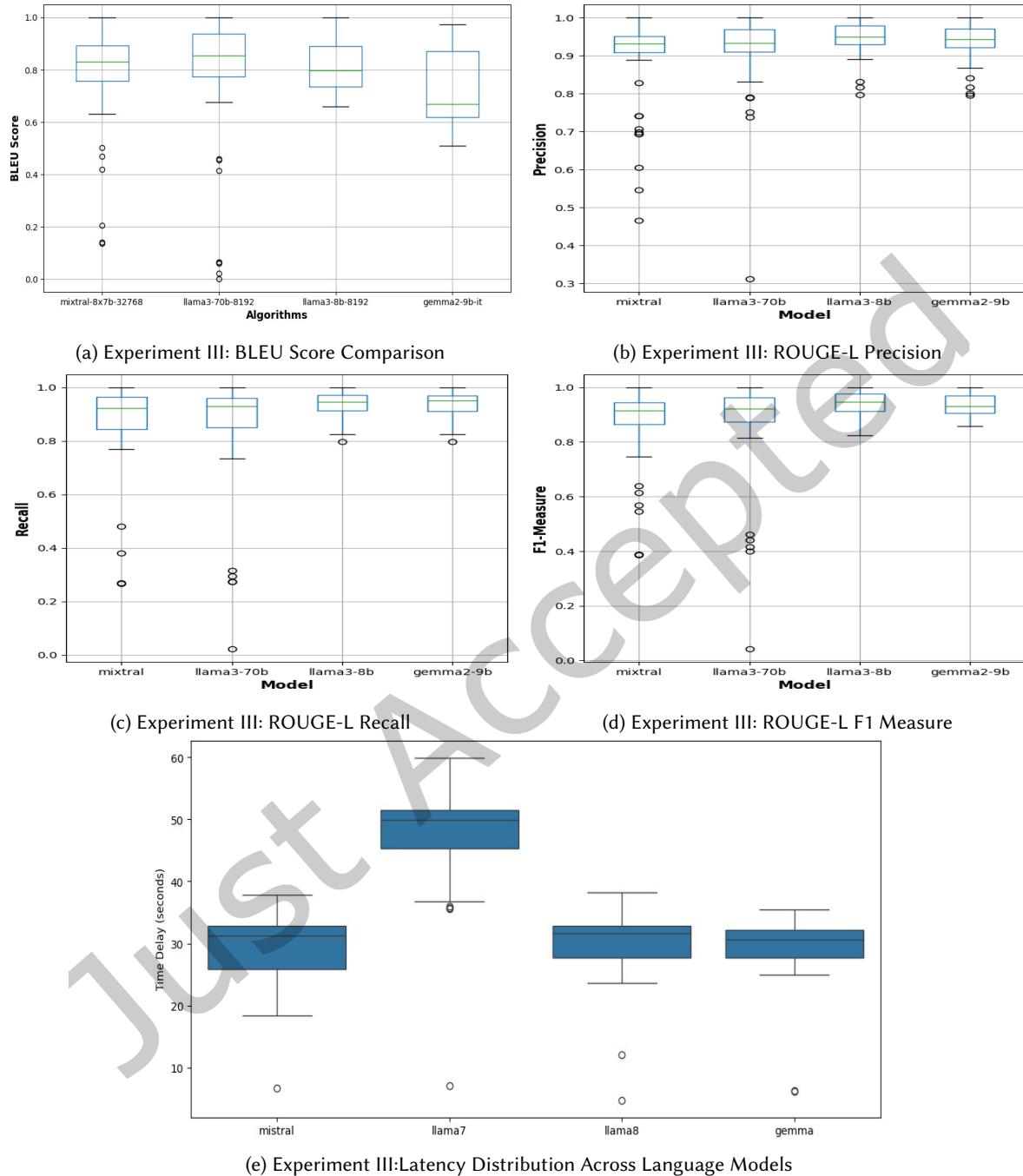


Fig. 11. Comparison of BLEU, ROUGE-L scores and Latency Distribution across Language Models: Mistral, LLaMA, and Gemma.

Semantic Similarity (%) for both approaches. The results indicate that GraphRAG provides a more robust and reliable approach for protocol emulation and response generation in ICS honeypots.

- **Protocol Compliance (%)**: GraphRAG 12a consistently achieves higher protocol adherence compared to BERT, with less variance and fewer outliers.
- **Semantic Similarity (%)**: GraphRAG 12b demonstrates superior semantic alignment with ground truth responses, maintaining a higher mean similarity.

The results indicate that GraphRAG surpasses BERT in protocol compliance and semantic similarity, achieving higher mean values with lower variance, ensuring greater consistency and reliability. BERT's lower minimum values suggest difficulties in handling certain queries, leading to erroneous or incomplete responses. The latency distribution histogram (12c) further highlights GraphRAG's efficiency, showing a more stable and lower-latency response, while BERT exhibits higher variance and occasional spikes. This demonstrates GraphRAG's suitability for real-time honeypot implementations, balancing response quality with faster execution, making it a more practical choice for intrusion deception and emulation.

### 5.5 Experiment V: Comparison of SAGE with Conpot

This experiment evaluates the Advanced IoT Honeypot Framework, which integrates LLM-driven response generation, against Conpot, a widely adopted ICS honeypot. Both honeypots were deployed in a containerized Ubuntu 20.04 environment, with public IP exposure to facilitate real-world adversarial interactions. To ensure a uniform and unbiased attack surface, both honeypots were deployed on public IPv4 addresses without geo-IP filtering or access control restrictions. Standard firewall rules were configured to allow inbound traffic only on protocol-specific ports (e.g., TCP/102 for S7comm) while blocking all other ports. No rate-limiting or authentication barriers were applied, allowing reconnaissance tools, bots, and adversarial scanners to interact freely. This configuration enabled consistent exposure across experiments and ensured that differences in attacker engagement could be attributed to honeypot behavior rather than network accessibility. The deployment utilized high-performance infrastructure (Intel Xeon Gold 6226R, 64 GB RAM, 1 TB NVMe SSD, and 10 Gbps NIC) to efficiently handle interactions over a five-month period. A two-tier evaluation methodology was employed:

- (1) Empirical Data Analysis – Analyzed real-world attack data from the CDAC honeypot and the Honey honeypot, assessing adversary interactions, attack diversity, and engagement effectiveness.
- (2) Controlled Penetration Testing – Used Snap7 scripts [37] to simulate adversarial interactions and systematically assess ICS protocol emulation capabilities.

This approach ensured a comprehensive assessment of the honeypots' response generation, deception effectiveness, and adversarial engagement.

- (1) Functionality and Protocol Emulation Capabilities: The experiment evaluated S7comm protocol emulation by executing test scripts covering system diagnostics, time functions, memory operations, control functions, security mechanisms, and error handling. Each functionality was assessed for resilience against adversarial techniques such as fingerprinting, unauthorized modifications, replay attacks, and protocol fuzzing. The results, as summarized in Table 11, show that the Advanced Honeypot Framework successfully handled all test cases, demonstrating robust protocol emulation. In contrast, Conpot exhibited limitations, particularly in system diagnostics, time functions, and security mechanisms, indicating weaker ICS protocol interaction capabilities. This comparison highlights the Advanced Honeypot's superior realism, adaptability, and effectiveness in deception, adversary engagement, and ICS security research.

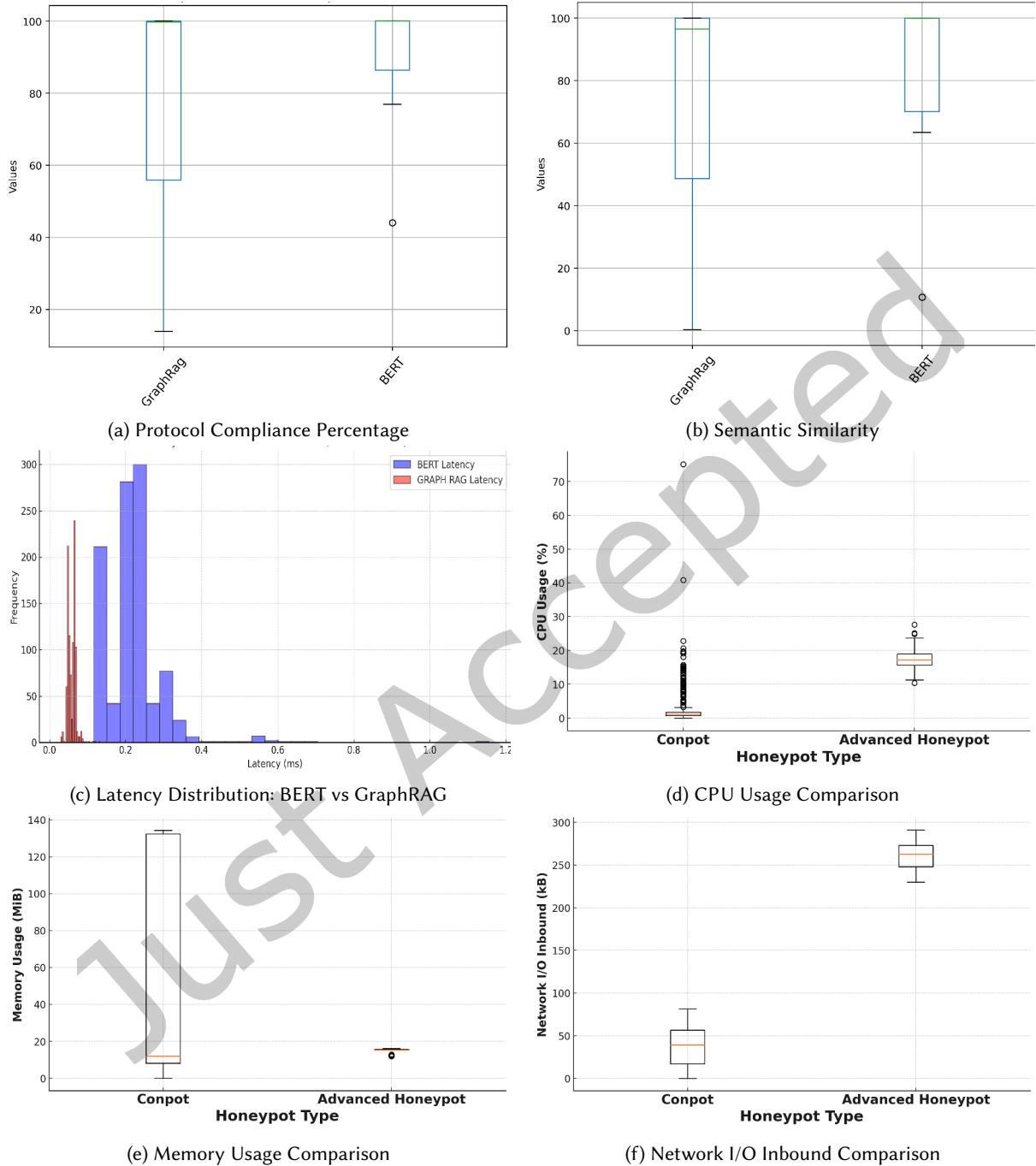


Fig. 12. Comparison of protocol compliance, semantic similarity, and latency for BERT and GraphRAG (top two rows); and system resource utilization for Conpot vs. the proposed Advanced Honeypot Framework (bottom row).

Test Category	Description (S7L ID)	Potential Attacks	SAGE	Conpot
System Diagnostics	Reads system status, CPU diagnostics (S7L 0x001C, 0x0232)	Fingerprinting, Unauthorized Modifications, Reconnaissance	✓	✗
Time Functions	Synchronizes, retrieves system clock (S7L 0x0074, 0x001A)	Time-Based Attacks, Replay, Process De-Synchronization	✓	✗
Memory Operations	Read/write DB, OB, I/O memory (S7L 0x011C, 0x001B)	Data Injection, Memory Manipulation, Unauthorized Access	✓	✗
Control Functions	Starts/stops CPU, changes modes (S7L 0x0001, 0x003F)	DoS, Privilege Escalation, Unauthorized Control	✓	✓
Security Mechanisms	Handles authentication, privilege management (S7L 0x0234)	Password Bypass, Session Hijacking, Brute Force Attacks	✓	✗
Error Handling	Processes malformed packets, protocol fuzzing (S7L 0x0112)	Buffer Overflow, DoS, Protocol Manipulation	✓	✓
Ladder Logic Capture	Collects attacker-injected PLC ladder logic for analysis	Malware Collection, Unauthorized Code Injection	✗	✗
Network Deception	Emulation of real PLC behavior against reconnaissance tools	Evasion of Nmap, PLCScan, Shodan HoneyScore	✓	✗

Table 11. S7comm Protocol Emulation Capabilities: Conpot vs. SAGE

- (2) Resource Consumption Analysis: To evaluate the resource utilization of the honeypots, Docker Stats was used to monitor CPU usage, memory efficiency, and network traffic processing for the honeypots. The Advanced Honeypot showed higher CPU usage than Conpot due to its dynamic response generation but maintained better memory efficiency and scalability. It also processed greater inbound network traffic (Figure 12f), indicating enhanced attacker engagement and a more interactive deception environment. These findings underscore the trade-off between computational overhead and deception effectiveness, demonstrating the Advanced Honeypot’s capability to sustain realistic, adaptive adversarial interactions.
- (3) Data Analysis Results: During the five-month deployment, the Advanced Honeypot recorded 12,320 interactions, more than double the engagement observed in Conpot (5,180 interactions). This higher engagement was also evident in the number of unique attacking IPs: the Advanced Honeypot identified 1,750 unique attackers, compared to only 650 identified by Conpot. State transition analysis revealed that the Advanced Honeypot supported a median of 6,000 state transitions (Figure 13a), more than double Conpot’s 2,500 transitions, indicating broader protocol emulation and deeper deception capabilities. Additionally, the Advanced Honeypot sustained daily engagement with 70 unique attacker IPs (Figure 13b), whereas Conpot maintained a slightly lower engagement of 60 unique attacker IPs per day. These findings underscore the Advanced Honeypot’s effectiveness in prolonging adversarial interactions, thereby enabling richer data collection for threat intelligence and attack pattern analysis.
- State Transition Analysis: The Advanced IoT Honeypot exhibited a broader range of state transitions (Figure 13c) compared to Conpot (Figure 13b), which remained confined within a limited transition loop. The presence of additional states (S4 and S5) in the Advanced Honeypot signifies deeper engagement, suggesting that adversaries were able to progress further into the honeypot’s interaction model. In contrast, Conpot was restricted to looping within S3, indicating a less dynamic and adaptable response mechanism.
  - S7L ID Exploration: Request analysis revealed patterns of adversarial reconnaissance behavior. The most frequently accessed S7L ID, 0x001C, indicated repeated attempts to retrieve system component details, a common fingerprinting technique for ICS identification. Access to other S7L IDs, such as 0x0011, 0x0C91, and 0x0174, suggested varied attack attempts focused on system diagnostics, control operations, and synchronization mechanisms. Table 12 provides a mapping of S7L ID values to their corresponding attack types, illustrating how adversaries use specific S7L requests for reconnaissance, system manipulation,

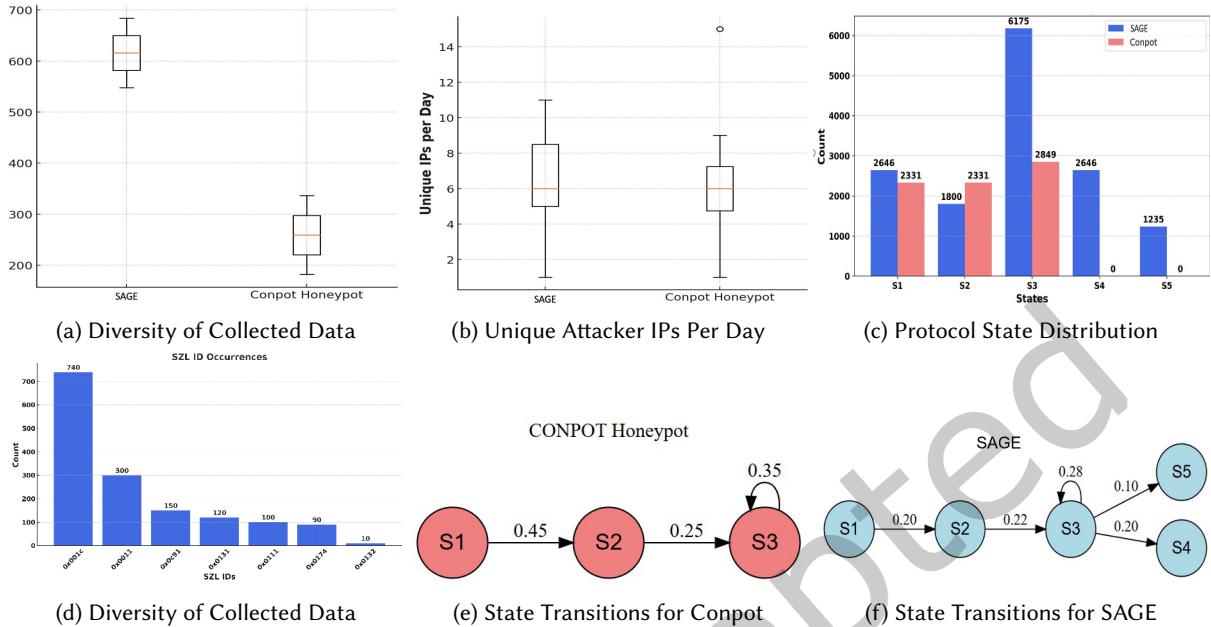


Fig. 13. Attack Engagement Metrics: Conpot vs Advanced Honeypot Framework

and information gathering. These findings highlight the Advanced Honeypot's capability to attract and log sophisticated ICS-specific attacks, supporting improved deception and detection strategies.

SZLID	Brief Description	Mapped Attack Type
0x001c	Identification of all components	Fingerprinting
0x0011	Module identification	Fingerprinting
0x0131	General data about communication of a communication unit	Information Gathering
0x0111	Module or System Diagnostics	Information Gathering
0x0c91	Time-of-Day (TOD) clock synchronization information	System Manipulation
0x0174	Module Status Information (detailed status of modules)	Information Gathering
0x0132	Status of the module's time system	System Manipulation

Table 12. Mapping of SZLID values to attack types

The SAGE provided a more realistic, engaging attack surface, capturing diverse ICS adversarial behaviors more effectively than Conpot. These findings confirm its superiority in deception, engagement, and attack data collection.

## 6 Conclusion

This paper presents SAGE (State-Aware Graph-Enhanced Honeypot), an adaptive deception framework that combines FSM-driven protocol emulation with GraphRAG-powered LLM response generation. By bridging structured finite state modeling with dynamic, knowledge-grounded language synthesis, SAGE overcomes key limitations of traditional and ML-based honeypots—namely, static behavior, weak protocol coverage, and poor adaptability to evolving adversarial tactics. The Mealy Machine-based FSM ensures protocol-consistent transitions,

while GraphRAG retrieval enhances response fidelity by injecting semantically structured context, mitigating hallucinations, and enabling reasoning across protocol states. These capabilities allow SAGE to generate deceptive yet realistic responses, even for undocumented or proprietary protocols.

Beyond technical performance, the core insight of this work is that honeypots can evolve from static decoys into adaptive, protocol-aware agents that actively co-learn from attacker behavior. This positions deception not just as a monitoring mechanism, but as an integral part of proactive defense in IoT ecosystems. SAGE demonstrates that blending symbolic modeling with retrieval-augmented generation enables scalable and accurate emulation in resource-constrained, protocol-diverse environments, a defining characteristic of modern IoT and IIoT deployments. From a broader security perspective, SAGE contributes to shifting the IoT defense paradigm from reactive signature-based detection to context-aware, attacker-engaging deception systems. This is particularly critical in operational environments like smart manufacturing and critical infrastructure, where traditional controls struggle with zero-day attacks, undocumented protocols, and fast-evolving threat landscapes.

Future work will focus on extending SAGE to support lightweight IoT protocols (e.g., MQTT, CoAP), integrating real-time threat intelligence feeds, and exploring reinforcement learning for autonomous deception strategy optimization. These directions aim to further establish adaptive deception as a foundational pillar in next-generation IoT cybersecurity architectures.

## References

- [1] 51CTO Blog. 2023. S7 Protocol Packet Capture Analysis (with PCAP Data Packet). [https://blog.51cto.com/u\\_11837698/6082457](https://blog.51cto.com/u_11837698/6082457). Accessed: 2025-08-11.
- [2] Meta AI. 2023. *Introducing LLaMA: Open and Efficient Foundation Language Models*. Meta. <https://ai.meta.com/blog/large-language-model-llama-meta-ai/> Accessed: 2025-08-11.
- [3] Tarek Ali and Panos Kostakos. 2023. HuntGPT: Integrating machine learning-based anomaly detection and explainable AI with large language models (LLMs). *arXiv preprint arXiv:2309.16021* (2023).
- [4] Anthropic. 2023. *Introducing Claude*. Anthropic. <https://www.anthropic.com/index/clause> Accessed:2025-08-11.
- [5] Daniele Antonioli, Anand Agrawal, and Nils Ole Tippenhauer. 2016. Towards high-interaction virtual ICS honeypots-in-a-box. In *Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy*. ACM, New York, NY, USA, 13–22.
- [6] Joao Antunes, Nuno Neves, and Paulo Verissimo. 2011. Reverse Engineering of Protocols from Network Traces. In *2011 18th Working Conference on Reverse Engineering*. Placeholder Publisher, 169–178. doi:10.1109/WCRE.2011.28
- [7] Giuseppe Bernieri, Mauro Conti, and Federica Pascucci. 2019. MimePot: a model-based honeypot for industrial control networks. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. IEEE, Bari, Italy, 433–438. doi:10.1109/SMC.2019.8913916
- [8] Muhammad Burhan, Rana Asif Rehman, Bilal Khan, and Byung-Seo Kim. 2018. IoT elements, layered architectures and security issues: A comprehensive survey. *sensors* 18, 9 (2018), 2796.
- [9] Censys. 2023. The Leading Internet Intelligence Platform for Threat Hunting and Exposure Management. <https://censys.io/> Accessed: 2025-08-11.
- [10] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. 2001. Progress on the state explosion problem in model checking. In *Informatics: 10 Years Back, 10 Years Ahead*, Reinhard Wilhelm (Ed.). Lecture Notes in Computer Science, Vol. 2000. Springer, Berlin, Heidelberg, 176–194. doi:10.1007/3-540-44577-3\_11
- [11] Riccardo Coppola and Maurizio Morisio. 2016. Connected car: technologies, issues, future trends. *ACM Computing Surveys (CSUR)* 49, 3 (2016), 1–36.
- [12] Dinil Mon Divakaran and Sai Teja Peddinti. 2024. LLMs for Cyber Security: New Opportunities. *arXiv preprint arXiv:2404.11338* (2024).
- [13] Amir Djenna, Saad Harous, and Djamel Eddine Saidouni. 2021. Internet of things meet internet of threats: New concern cyber security issues of critical cyber infrastructure. *Applied Sciences* 11, 10 (2021), 4580.
- [14] Emre Ekin. 2023. ICS-PCAPS: Industrial Control System Packet Captures. <https://github.com/EmreEkin/ICS-Pcaps>. Accessed:2025-08-11.
- [15] Omer Erdem. 2023. HoneyThing: A Lightweight Honeypot for IoT Devices. <https://github.com/omererdem/honeything>. Accessed: 2025-02-06.
- [16] Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, New York, NY, USA, 6491–6501. doi:10.1145/3637528.3671841
- [17] Wenjun Fan, Zhihui Du, David Fernández, and Victor A Villagra. 2017. Enabling an anatomic view to investigate honeypot systems: A survey. *IEEE Systems Journal* 12, 4 (2017), 3906–3919.

- [18] Javier Franco, Ahmet Aris, Berk Canberk, and A Selcuk Uluagac. 2021. A survey of honeypots and honeynets for internet of things, industrial internet of things, and cyber-physical systems. *IEEE Communications Surveys & Tutorials* 23, 4 (2021), 2351–2383.
- [19] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* (2023).
- [20] Niv Goldenberg and Avishai Wool. 2013. Accurate modeling of Modbus/TCP for intrusion detection in SCADA systems. *international journal of critical infrastructure protection* 6, 2 (2013), 63–75.
- [21] Google. 2023. *Introducing Bard by Google*. Google. <https://bard.google.com> Accessed: 2025-08-11.
- [22] Andy Greenberg. 2021. A Hacker Tried to Poison a Florida City’s Water Supply. <https://www.wired.com/story/oldsmar-florida-water-utility-hack/>. Accessed: 2025-08-11.
- [23] Jindong Gu, Zhen Han, Shuo Chen, Ahmad Beirami, Bailan He, Gengyuan Zhang, Ruotong Liao, Yao Qin, Volker Tresp, and Philip Torr. 2023. A systematic survey of prompt engineering on vision-language foundation models. *arXiv preprint arXiv:2307.12980* (2023).
- [24] Chongqi Guan, Heting Liu, Guohong Cao, Sencun Zhu, and Thomas La Porta. 2023. HoneyIoT: Adaptive High-Interaction Honeypot for IoT Devices Through Reinforcement Learning. In *Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec ’23)*. ACM, Brisbane, QLD, Australia, 49–59. doi:10.1145/3558482.3590190
- [25] Muhammad A Hakim, Hidayet Akssu, A Selcuk Uluagac, and Kemal Akkaya. 2018. U-pot: A honeypot framework for upnp-based iot devices. In *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 1–8.
- [26] Arthur Jicha, Mark Patton, and Hsinchun Chen. 2016. SCADA honeypots: An in-depth analysis of Conpot. In *2016 IEEE conference on intelligence and security informatics (ISI)*. IEEE, 196–198.
- [27] Hamza Kheddar. 2024. Transformers and large language models for efficient intrusion detection systems: A comprehensive survey. *arXiv preprint arXiv:2408.07583* (2024).
- [28] Amit Kleinman and Avishai Wool. 2014. Accurate modeling of the siemens s7 scada protocol for intrusion detection and digital forensics. *The Journal of Digital Forensics, Security and Law: JDLS* 9, 2 (2014), 37.
- [29] Seungjin Lee, Azween Abdullah, and NZ Jhanjhi. 2020. A review on honeypot-based botnet detection models for smart factory. *International Journal of Advanced Computer Science and Applications* 11, 6 (2020).
- [30] Tongbo Luo, Zhaoyan Xu, Xing Jin, Yanhui Jia, and Xin Ouyang. 2017. Iotcandyjar: Towards an intelligent-interaction honeypot for iot devices. *Black Hat* 1 (2017), 1–11.
- [31] Abhishek Mairh, Debabrat Barik, Kanchan Verma, and Debasis Jena. 2011. Honeypot in network security: a survey. In *Proceedings of the 2011 International Conference on Communication, Computing & Security (ICCCS)*. ACM, Odisha, India, 600–605.
- [32] Mandiant. 2024. Sandworm Disrupts Power in Ukraine Through Operational Technology. <https://www.mandiant.com/resources/blog/sandworm-disrupts-power-ukraine-operational-technology>. Accessed: 2025-08-11.
- [33] George H Mealy. 1955. A method for synthesizing sequential circuits. *The Bell System Technical Journal* 34, 5 (1955), 1045–1079.
- [34] Volviane Saphir Mfogo, Alain Zemkoho, Laurent Njilla, Marcellin Nkenfack, and Charles Kamhoua. 2023. AIIPot: Adaptive intelligent-interaction honeypot for IoT devices. In *2023 IEEE 34th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*. IEEE, Toronto, ON, Canada, 1–6.
- [35] Ariana Mirian, Zane Ma, David Adrian, Matthew Tischer, Thasphon Chuenchujit, Tim Yardley, Robin Berthier, Joshua Mason, Zakir Durumeric, J. Alex Halderman, et al. 2016. An internet-wide view of ICS devices. In *2016 14th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, Auckland, New Zealand, 96–103.
- [36] Ariana Mirian, Zane Ma, David Adrian, Matthew Tischer, Thasphon Chuenchujit, Tim Yardley, Robin Berthier, Joshua Mason, Zakir Durumeric, J. Alex Halderman, and Michael Bailey. 2016. An Internet-wide view of ICS devices. In *2016 14th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, Auckland, New Zealand, 96–103. doi:10.1109/PST.2016.7906943
- [37] Gijs Molenaar and Stephan Preeker. 2023. python-snap7 Documentation. <https://buildmedia.readthedocs.org/media/pdf/python-snap7/latest/python-snap7.pdf>. Accessed: 2025-08-11.
- [38] Neo4j Inc. 2025. Neo4j: Graph Database Platform. <https://neo4j.com/> Accessed: 2025-02-28.
- [39] Nataliaia Neshenko, Elias Bou-Harb, Jorge Crichigno, Georges Kaddoum, and Nasir Ghani. 2019. Demystifying IoT security: An exhaustive survey on IoT vulnerabilities and a first empirical look on Internet-scale IoT exploitations. *IEEE Communications Surveys & Tutorials* 21, 3 (2019), 2702–2733.
- [40] Kim Anh Nguyen, Maximilian Köper, Sabine Schulte im Walde, and Ngoc Thang Vu. 2017. Hierarchical embeddings for hypernymy detection and directionality. *arXiv preprint arXiv:1707.07273* (2017).
- [41] Zainab Noor, Sadaf Hina, Faisal Hayat, and Ghalib A Shah. 2023. An intelligent context-aware threat detection and response model for smart cyber-physical systems. *Internet of Things* 23 (2023), 100843.
- [42] OpenAI. 2022. *Introducing ChatGPT*. Openai. <https://openai.com/blog/chatgpt> Accessed: 2025-08-11.
- [43] Yin Minn Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow. 2016. IoTPOT: A novel honeypot for revealing current IoT threats. *Journal of Information Processing* 24, 3 (2016), 522–533.
- [44] Baolin Peng, Michel Galley, Pengcheng He, Hao Cheng, Yujia Xie, Yu Hu, Qiuyuan Huang, Lars Liden, Zhou Yu, Weizhu Chen, et al. 2023. Check your facts and try again: Improving large language models with external knowledge and automated feedback. *arXiv preprint*

- arXiv:2302.12813* (2023).
- [45] Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. 2024. Graph retrieval-augmented generation: A survey. *arXiv preprint arXiv:2408.08921* (2024).
- [46] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. 2013. Context aware computing for the internet of things: A survey. *IEEE communications surveys & tutorials* 16, 1 (2013), 414–454.
- [47] Morteza Safaei Pour, Dylan Watson, and Elias Bou-Harb. 2021. Sanitizing the IoT Cyber Security Posture: An Operational CTI Feed Backed up by Internet Measurements. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, Taipei, Taiwan, 497–506. doi:10.1109/DSN48987.2021.00059
- [48] Niels Provos. 2003. Honeyd-a virtual honeypot daemon. In *10th dfn-cert workshop, hamburg, germany*, Vol. 2. 4.
- [49] Jarrod Ragsdale and Rajendra V. Boppana. 2023. On Designing Low-Risk Honeypots Using Generative Pre-Trained Transformer Models With Curated Inputs. *IEEE Access* 11 (2023), 117528–117545. doi:10.1109/ACCESS.2023.3326104
- [50] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2024. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927* (2024).
- [51] SektorCERT. 2023. The Attack Against Danish Critical Infrastructure. SektorCERT Report. <https://sektorcrt.dk/wp-content/uploads/2023/11/SektorCERT-The-attack-against-Danish-critical-infrastructure-TLP-CLEAR.pdf> Accessed: 2025-08-11.
- [52] Yao Shan, Yu Yao, Tong Zhao, and Wei Yang. 2023. NeuPot: A Neural Network-Based Honeypot for Detecting Cyber Threats in Industrial Control Systems. *IEEE Transactions on Industrial Informatics* 19, 10 (2023), 10512–10522. doi:10.1109/TII.2023.3240739
- [53] SHODAN. 2022. Internet Enumeration Engine. <https://www.shodan.io/dashboard>. Accessed: 2025-08-11.
- [54] Ilias Sinosoglou, Georgios Efstatopoulos, Dimitrios Pliatsios, Ioannis D Moscholios, Antonios Sarigiannidis, Georgia Sakellari, Georgios Loukas, and Panagiotis Sarigiannidis. 2020. NeuralPot: An Industrial Honeypot Implementation Based on Deep Neural Networks. In *2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, Rennes, France, 1–7. doi:10.1109/ISCC50000.2020.9219693
- [55] Jason Smith. n.d.. A Collection of ICS/SCADA PCAPs. <https://github.com/automayt/ICS-pcap>. Accessed: 2025-08-11.
- [56] Ralf Spenneberg, Maik Brüggemann, and Hendrik Schwartke. 2016. PLC-blaster: A worm living solely in the PLC. *Black Hat Asia* 16 (2016), 1–16.
- [57] Tianyi Tang, Junyi Li, Wayne Xin Zhao, and Ji-Rong Wen. 2022. Context-tuning: Learning contextualized prompts for natural language generation. *arXiv preprint arXiv:2201.08670* (2022).
- [58] Vanessa Tay, Xinran Li, Daisuke Mashima, Bennett Ng, Phuong Cao, Zbigniew Kalbarczyk, and Ravishankar K. Iyer. 2023. Taxonomy of Fingerprinting Techniques for Evaluation of Smart Grid Honeypot Realism. In *2023 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. IEEE, Glasgow, UK, 1–7. doi:10.1109/SmartGridComm57356.2023.10313056
- [59] Tran Minh Cuong. 2022. Security Wall of S7CommPlus. <https://blog.viettelcybersecurity.com/security-wall-of-s7commplus-part-1/>. Accessed: 2025-08-11.
- [60] Christoforos Vasilatos, Dunia J Mahboobeh, Hithem Lamri, Manaar Alam, and Michail Maniatakos. 2024. LLMPot: Automated LLM-based Industrial Protocol and Physical Process Emulation for ICS Honeypots. *arXiv preprint arXiv:2405.05999* (2024).
- [61] Emmanouil Vasilomanolakis, Shreyas Srinivasa, Carlos Garcia Cordero, and Max Mühlhäuser. 2016. Multi-Stage Attack Detection and Signature Generation with ICS Honeypots. In *2016 IEEE/IFIP Network Operations and Management Symposium (NOMS)*. IEEE, Istanbul, Turkey, 1227–1232. doi:10.1109/NOMS.2016.7502958
- [62] Alexander Vetterl and Richard Clayton. 2019. Honware: A Virtual Honeypot Framework for Capturing CPE and IoT Zero Days. In *2019 APWG Symposium on Electronic Crime Research (eCrime)*. IEEE, Pittsburgh, PA, USA, 1–13. doi:10.1109/eCrime47957.2019.9037572
- [63] Meng Wang, Javier Santillan, and Fernando Kuipers. 2018. ThingPot: an interactive Internet-of-Things honeypot. *arXiv preprint arXiv:1807.04114* (2018).
- [64] Wireshark Foundation. 2024. S7comm - S7 Communication Protocol Sample Captures. <https://wiki.wireshark.org/SampleCaptures#s7comm--s7-communication> Accessed: 2025-08-11.
- [65] World of IoT. 2023. An Overview of the Global PLC Industry and Its Dynamics. <https://medium.com/world-of-iot/95-an-overview-of-the-global-plc-industry-and-its-dynamics-102ae2fc0b4>. Accessed: 2025-08-11.
- [66] Nan Zhang, Soteris Demetriadou, Xianghang Mi, Wenrui Diao, Kan Yuan, Peiyuan Zong, Feng Qian, XiaoFeng Wang, Kai Chen, Yuan Tian, et al. 2017. Understanding iot security through the data crystal ball: Where we are now and where we are going to be. *arXiv preprint arXiv:1703.09809* (2017), 34 pages.
- [67] Qinggang Zhang, Shengyuan Chen, Yuanchen Bei, Zheng Yuan, Huachi Zhou, Zijin Hong, Junnan Dong, Hao Chen, Yi Chang, and Xiao Huang. 2025. A Survey of Graph Retrieval-Augmented Generation for Customized Large Language Models. *arXiv preprint arXiv:2501.13958* (2025).
- [68] Hengye Zhu, Mengxiang Liu, Binbin Chen, Xin Che, Peng Cheng, and Ruilong Deng. 2024. HoneyJudge: A PLC Honeypot Identification Framework Based on Device Memory Testing. *IEEE Transactions on Information Forensics and Security* 19 (2024), 6028–6043. doi:10.1109/TIFS.2024.3407520
- [69] ZOOMEYE. 2022. Internet Search Engine for Cyberspace. <https://www.zoomeye.org/>. Accessed: 2025-08-11.

Received 3 March 2025; revised 27 May 2025; accepted 18 August 2025

Just Accepted