2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

IRPFuzz: Fuzzing Industrial Robot Protocol via LLM-driven Traffic Semantic Analysis

Laile Xi[1], Weicheng Lin[1], Yang Zhang[2], Shenghao Lin[1], Hao Sun[3], Yimo Ren[1†], Hongsong Zhu[1†]

Abstract— **Industrial robots are widely used in modern factories, interacting with external systems via network protocols. Any vulnerabilities in these protocols could be exploited to control the robot, potentially disrupting production and even endangering human lives. Protocol fuzzing is commonly used to discover vulnerabilities in protocol implementation. However, existing fuzzers are inefficient due to the proprietary nature and complex state relationships of industrial robot protocols.**

In this paper, we present IRPFuzz, a state-aware fuzzer for industrial robot protocol. By integrating large language models (LLMs) to analyze network traffic, IRPFuzz infers robot states and request templates, enabling the automatic construction of state model and data model for efficient state-aware fuzzing and structured message mutation. Evaluated on a real robot, IRPFuzz discovered 36 crashes, outperforming boofuzz by 157.14%, PCFuzzer by 89.47%, and MSGFuzzer by 16.13%. Five of these crashes were confirmed and assigned vulnerability IDs, including three classified as high-severity, which demonstrates the effectiveness of IRPFuzz.

I. INTRODUCTION

Industrial robots are typical cyber-physical systems that are widely deployed in modern manufacturing. The network protocols they employ serve as critical channels for interaction with external systems. However, increased network connectivity has significantly expanded the potential attack surface of devices that were once isolated [1] [2]. Network protocols and associated software for industrial robots often prioritize functionality over security, making them prone to vulnerabilities that attackers can exploit to compromise control systems and cause severe consequences [3] [4]. Therefore, identifying potential vulnerabilities is essential to ensuring security and reliability of industrial robots.

Protocol fuzzing is a widely adopted technique for discovering vulnerabilities [5]. It is the process of repeatedly generating inputs and feeding them to the system under test (SUT). Based on the degree of internal knowledge they have about the SUT, fuzzers can be divided into white-box, gray-box, and black-box. White-box and gray-box fuzzers leverage internal program information, using techniques such as symbolic execution [6] or instrumentation [7] to iteratively optimize seed for efficient fuzzing [8] [9] [10]. Black-box fuzzers rely on user-defined data models to generate a large number of protocol-compliant inputs, making them particularly suitable for closed-source protocols [11] [12].

However, due to the high degree of customization in industrial robots, manufacturers rarely release source code, posing two major challenges for fuzzing robot protocols.

How to capture states and their relationships. Industrial robots encompass various states, such as login, enable and motion states, governed by strict transition rules. Violating these rules may cause message rejection. For example, a robot responds to a "move" message only after a "motor on" message that activates the enable state. However, as industrial robot protocols are proprietary, traditional fuzzers based on response codes [9], memory [10] or state variable [13] cannot capture states and their relationships, which are essential for effective fuzzing.

How to perform efficient mutation. In the absence of source code and documentation, feedback-driven mutation in white-box or gray-box fuzzers is not feasible, while the user-defined data models required by black-box fuzzers are likewise challenging to construct. Protocol reverse engineering has enabled fuzzing of proprietary protocols [14] [15] [16], but lacks generality for industrial robots. MSGfuzzer depends on manual protocol analysis for seed mutation, making it labor-intensive and limited in generality [17].

In this paper, we propose IRPFuzz, an efficient state-aware fuzzer for industrial robot protocols. By conducting an in-depth analysis of real-world interaction traffic between robot and teach pendant, we observe that they consistently embed semantic information about robot states. Motivated by this insight, we propose an LLM-driven traffic semantic analysis approach that facilitates the construction of state model and data model to guide efficient fuzzing. Specifically, we leverage LLMs to analyze traffic between robot and teach pendant, extracting both runtime state and field-level request templates, which are then used to construct the state model and data model that capture state relationships and guide efficient message mutation. Moreover, the state model and data model are progressively refined as fuzzing proceeds, thereby further enhancing the effectiveness of the IRPFuzz.

We evaluated IRPFuzz on a widely used commercial-grade industrial robot ROKAE XB7L. Within 24 hours of fuzzing, IRPFuzz efficiently constructed state model and data model that accurately reflect the operational characteristics of real industrial robots. Moreover, IRPFuzz discovered 36 crashes, achieving improvements of 157.14%, 89.47%, and 16.13% over state-of-the-art fuzzers Boofuzz, PCFuzzer,

The first two authors contribute equally. †Corresponding author.

This work was supported by the the Talent ProgramProject of the Institute of Information Engineering, CAS, China (Grant No. E5YY07111K4).

[1]These authors are with the Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China and School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China. {xilaile1998, linweicheng, linshenghao, renyimo, zhuhongsong}@iie.ac.cn.

[2]Yang Zhang is with the School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing, China. zhang970929@163.com.

[3]Hao Sun is with the Police Information Department, Shangdong Police College, Jinnan, China. 1229694198@qq.com.

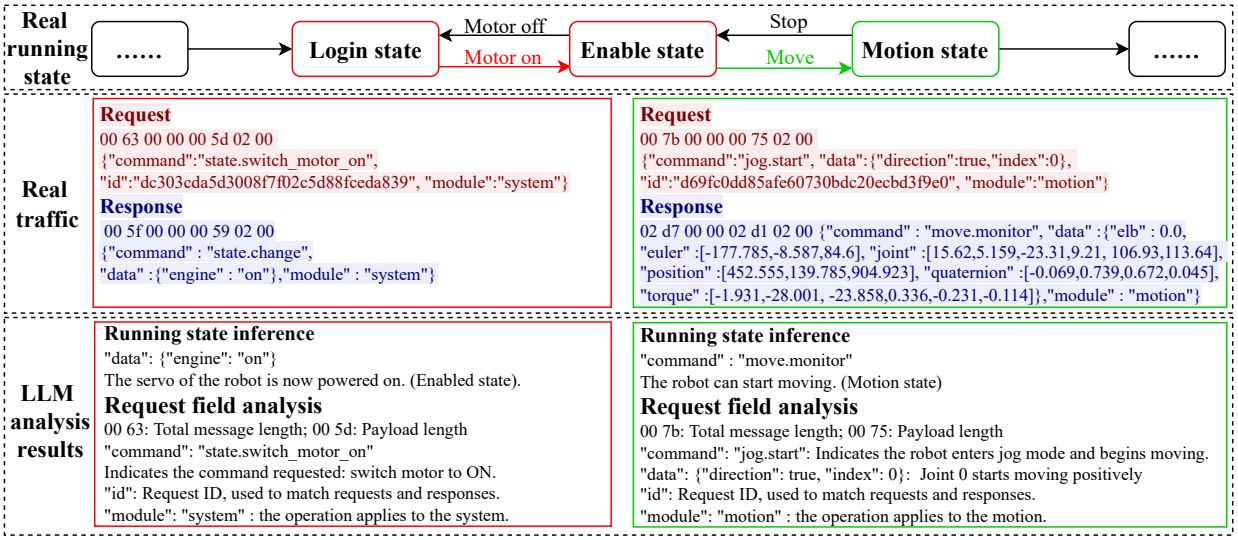

Fig. 1. Analysis Results of Industrial Robot Traffic by the LLM.

and MSGFuzzer, respectively. We reported these crashes to the manufacturer, five of which have been confirmed and assigned official vulnerability IDs, demonstrating the effectiveness of IRPFuzz. In addition, we discussed the impact of different LLMs on the effectiveness of IRPFuzz.

In summary, we make the following contributions:

- We present IRPFuzz, which leverages LLMs to analyze industrial robot protocol traffic, extract semantic information on robot states and message fields, and adaptively construct state model and data model to enable state-aware fuzzing and structured message mutation.
- We evaluated IRPFuzz on a real industrial robot. Within 24 hours of fuzzing, IRPFuzz discovered 36 crashes, surpassing all baseline methods. Five of these crashes have been confirmed and assigned vulnerability IDs, with three classified as high-severity vulnerabilities.

II. BACKGROUND AND MOTIVATION

A. Industrial Robot

Industrial robot is an automatically controlled, reprogrammable, and versatile manipulator that can be programmed along three or more axes [18]. It can be either fixed in place or mobile and is used for various industrial automation applications. It typically consists of multiple sensors and actuators that receive signals from external devices and execute specific commands, enabling it to perform a wide range of production tasks such as material handling, welding, assembly, and painting without human intervention.

Industrial robot consists of multiple modules, with the main controller and servo motor drivers being the most critical components [19] [20]. The remaining components are connected to these core units via internal buses and are housed within a dedicated control cabinet. The main controller runs a manufacturer-customized operating system and communicates with other devices in the control cabinet through Ethernet or serial buses. It also stores the control logic and various configuration parameters of robots. The servo motor drivers serve as intermediaries between the main controller and the robotic arm, receiving control commands from the main controller and driving the arm's movements. The robotic arm is the primary actuator of an industrial robot, consisting of multiple independently movable axes capable of performing complex tasks similar to the human arm.

In addition, industrial robots are equipped with a teach pendant that connects to the robot controller, allowing operators to send commands and control the robot [21]. The teach pendant software runs on this device and is provided by the robot manufacturer, though it can also be executed independently on a computer. It connects to the industrial robot via Ethernet and typically uses proprietary protocols to transmit control instructions, tailored to specific robot brands. The industrial robot offers a comprehensive set of control commands to the teach pendant software, covering nearly all operational functionalities of the robot.

B. Protocol Fuzzing

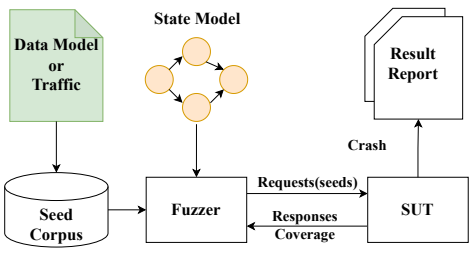

Fig. 2. The Workflow of Stateful Network Protocol Fuzzer.

Protocol fuzzing has been widely demonstrated to be an effective technique for discovering protocol vulnerabilities.

To effectively perform fuzzing on stateful network servers, researchers have explored state-model-based fuzzers [22] [11] [23]. A state model is typically represented as a graph consisting of a finite set of states and their transitions. Different studies have adopted various strategies to abstract server states. For example, AFLNET [9] and CHATAFL [24] represent states based on server response codes (e.g., 200 OK, 400 ERR), StateAFL [10] leverages memory states to capture protocol states, and NSFuzz [13] uses variables extracted through static analysis to represent states. By incorporating a state model and monitoring responses from the SUT, a fuzzer can infer the current protocol state and generate seeds according to the expected message format of that state, as shown in Fig. 2. These messages may be generated completely at random [25], mutated from existing messages [9], or constructed based on predefined data models [11] [12].

C. Motivation

Due to the highly proprietary nature of industrial robot protocols, existing fuzzers are unable to construct accurate state model and data model of the target protocols, which severely limits their effectiveness in discovering vulnerabilities within industrial robot protocols.

We analyzed industrial robot traffic and found that, although the protocols are proprietary, certain fields implicitly convey robot state semantics. Inspired by the success of LLMs in protocol analysis [26], we propose leveraging LLMs to analyze state semantics and field structures, enabling the automated construction of state model and data model that replace manual efforts and facilitate efficient state-aware fuzzing.

Fig. 1 shows a segment of real industrial robot traffic and the corresponding semantic analysis by the ChatGPT-4o [27]. The traffic records the process of controlling the robot via the teach pendant to power it on and move, involving transitions from the login state to the enable state and then to the motion state. The analysis results align with the actual execution, correctly identifying the enable and motion states. Moreover, in analyzing request field structures, the LLM successfully recognized key elements such as length and command.

III. PROPOSED METHOD

Fig. 3 depicts the workflow of IRPFuzz. It first captures the traffic between the teach pendant and the industrial robot, preprocesses them, and then performs LLM-driven traffic semantic analysis to infer runtime states and request templates, thereby assisting in the construction of state and data models. In the fuzzing loop, IRPFuzz selects a target state from the state model and drives the robot into it. A corresponding request template is then retrieved from the data model based on the state name and corresponding requests, mutated by the message mutator to generate new inputs, which will be feed into the robot. The new inputs and responses are continuously collected and reanalyzed by the LLM, progressively refining both models and enabling efficient state-aware fuzzing.

A. Initial Traffic Collection

The teach pendant software supports nearly the full range of robot operations and serves as the primary interface for industrial robots, providing favorable conditions for fuzzing.

Traffic collection. We begin by performing a series of operations on the industrial robot via the teach pendant software and capturing the interactive traffic with Wireshark [28]. The captured traffic is organized into chronologically ordered request–response pairs to facilitate subsequent analysis.

Preprocessing. However, the raw traffic contains numerous periodic heartbeat packets and automatic state synchronization packets that used primarily to confirm whether the robot is online. Retaining these packets imposes unnecessary overhead and undermines fuzzing efficiency, making their removal essential. Algorithm 1 outlines this process. Since formats of repetitive messages differ from other control messages, we first cluster the raw requests using HDBSCAN [29] (line 1). Next, we apply a top-down iterative pruning strategy. After the initial clustering, clusters are processed in descending order of size (lines 4-11): each cluster is tentatively removed (line 5), and the remaining requests are replayed in their original order to compare responses with the robot (line 6). If the responses remain equivalent, the cluster is deemed redundant and discarded; otherwise, it is retained (lines 7-10). Once all clusters from the first round have been examined, the retained clusters are re-clustered, and the same pruning procedure is recursively applied to achieve further reduction (lines 12-21). Ultimately, we obtain a traffic subset T^* that preserves the sequence of teaching operations while eliminating redundancy, forming the basis for subsequent analysis.

B. LLM-driven Traffic Semantic Analysis

We employ prompt engineering [30] to guide the LLM in analyzing the semantics of each traffic, inferring the robot's current running state and the corresponding request template, thereby constructing the state model and data model.

Running state inference. To infer running state of robot hidden in the traffic, we employ the prompt template shown in Fig. 4(a). In this prompt, we provide the current state model and request–response pair along with illustrative examples to constrain the output of LLM. The LLM then produces the name and description of the inferred state. To mitigate hallucination, we perform multiple rounds of interaction with the LLM and adopt the majority-consistent result as the final inference.

State model construction. We define an initial state as the root of the state model and construct the model in two phases: initialization and update. In the initialization phase, state inference is performed on the preprocessed traffic subset T^*. For each state inferred by the LLM, if the state does not exist in the model, it is appended to the most recently inferred state, and a directed edge is created to store the corresponding request, reflecting the sequential nature of the teaching operations. If the state already exists, we further check whether a directed edge from the most recent state to this state exists; if not, a new edge is created. In the update

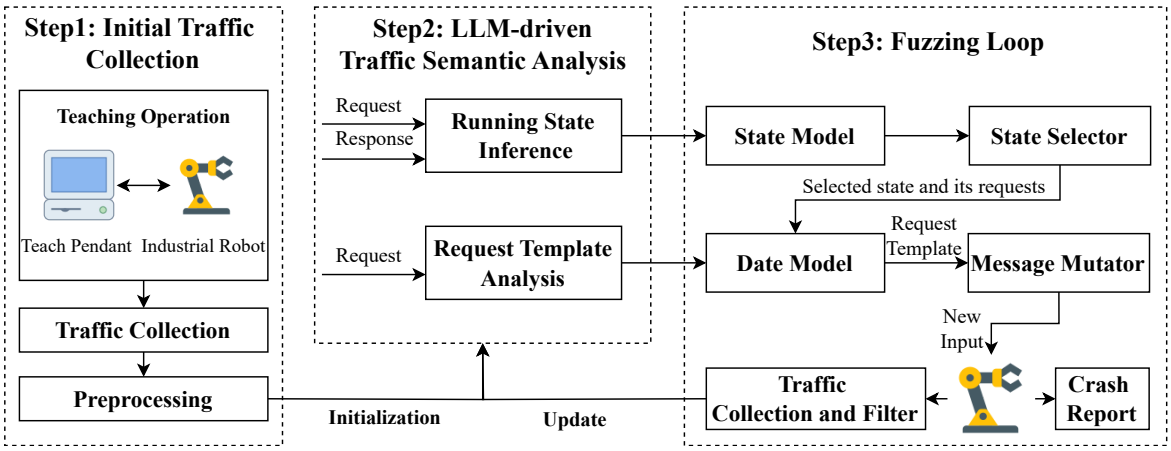

Fig. 3. The Workflow of IRPFuzz.

Algorithm 1 Traffic Preprocessing

Input: IT: Initial traffic set
Output: T^*: Traffic subset for subsequent analysis.

$C \leftarrow \text{HDBSCAN}(IT)$
$R \leftarrow IT.response$
$T^* \leftarrow IT$
for $c \in C$ in descending order of size **do**
  $T' \leftarrow T^* \setminus c$
  Replay $T'.request$ and record responses R'
  if EQUIVALENT(R', R) **then**
    $T^* \leftarrow T'$
    $C^* \leftarrow C \setminus c$
  end if
end for
for $c' \in C^*$ in descending order of size **do**
  $C' \leftarrow \text{HDBSCAN}(c')$
  for $c'' \in C'$ in descending order of size **do**
    $T'' \leftarrow T^* \setminus c''$
    Replay $T''.request$ and record responses R''
    if EQUIVALENT(R'', R) **then**
      $T^* \leftarrow T''$
    end if
  end for
end for
return T^*

Instruction:
Analyze {*Request*} and {*Response*} of industrial robot to infer running state.
For inferred state:
• If its semantics closely match a state in {*State Model*}, return the state.
• Otherwise, return the new state name and provide a description.

Output Format:
"Enable state": "The servo of the industrial robot is now powered on. The robot can execute motion commands."

(a) Running state inference.

Instruction:
Analyze the field structure of the {*Request*} of the industrial robot and output the content, type, and mutability of each field in order.

Output Format:
[{"content": "{", "type": "delim", "mutability": false },
{"content": ""command"", "type": "string", "mutability": true },
{"content": ":", "type": "delim", "mutability": false },
.....]

(b) Request template analysis.

Fig. 4. Prompt Templates used in LLM-based Traffic Analysis.

phase, state inference is performed on the traffic collected during fuzzing loop. For each inferred state, if it does not exist in the model, it is appended to the currently selected state, and a directed edge is created to store the request. If the state already exists, we check whether an edge from the selected state to this state exists, a new edge is added only when missing.

Request template analysis. We only perform request template analysis when the state model changes, such as discovering new states or new edges, using the prompt template as shown in the Fig. 4(b). The prompt includes examples of different fields to guide the LLM in correctly interpreting and adhering to the given grammar. The LLM is expected to identify and annotate the mutability of each field. After the LLM processes a request, the extracted fields are reassembled and compared with the original input. If certain fields are missing, the procedure is repeated until no omissions remain.

Hash	Request Template

Fig. 5. The Structure of Entries in the Data Model.

Data model construction. We maintain the data model in a table, where each entry consists of two fields: hash and request template, as illustrated in the Fig. 5. The hash value is computed from the state name and request content, serving both as a unique identifier for each request template and as an index for efficient retrieval. The request template field stores the analyzed request template, enabling structured message mutation. After each request template analysis, IRPFuzz computes the hash of the current state name and request and sequentially inserts a new entry into the table.

C. Fuzzing Loop

After performing LLM-driven traffic analysis, the initial state model and data model are constructed. IRPFuzz then leverages them to initiate state-aware fuzzing of the industrial robot protocol.

State selector. IRPFuzz sequentially traverses the states in the state model. For each selected state, IRPFuzz derives a feasible path from the initial state, gathers the requests stored on all edges along the path, and constructs a guiding sequence that drives the robot into the selected state. We search the state model for outgoing edges from the selected state and collect the associated requests. The selected state and its requests are then forwarded to the data model, which retrieves the corresponding request templates based on their hashes.

Message mutator. IRPFuzz performs structural mutations guided by the constraints on field structure and mutability specified in the request templates. Leveraging Boofuzz's mutation strategy, it iteratively mutate each mutable field to generate new requests. These requests are then combined with the guiding sequence to form complete test cases, which are subsequently executed on the industrial robot.

Crash detection. IRPFuzz detects crashes by analyzing network connectivity and robot responses. After each test case execution, IRPFuzz sends three heartbeat packets to the robot and monitors the responses. Under normal conditions, the robot responds to each heartbeat packet with system time and identification information. If no response is observed, the robot is deemed to have crashed. IRPFuzz then logs the crashing input and reboots the robot to resume fuzzing.

Traffic Collection and Filter. During fuzzing, IRPFuzz continuously collects request and response. After fuzzing each selected state, we employ a similarity-based filtering strategy using edit distance [31] to reduce redundancy. Specifically, we compare each collected response string $CR[1..i]$ against each known successor-state response string $SR[1..j]$ of the current state and compute their distance $D[i][j]$ according to Eq. 1. The 50 responses with the largest differences, along with their corresponding requests, are then submitted to the LLM for a new round of analysis to identify previously undetected states and their associated request templates, gradually refining the state and data models.

$$D[i][j] = \begin{cases} i + j & \text{if } i = 0 \text{ or } j = 0 \\ D[i-1][j-1] & \text{if } SR[j] = CR[i] \\ \min \begin{cases} D[i-1][j] + 1 \\ D[i][j-1] + 1 \\ D[i-1][j-1] + 1 \end{cases} & \text{if } SR[j] \neq CR[i] \end{cases} \quad (1)$$

IV. EVALUATION

We implemented IRPFuzz on top of the Boofuzz framework. During the initial traffic collection phase, typical operations were performed through the teach pendant software, including robot motion, end-effector control, program execution, and point teaching, while initial traffic was captured using Wireshark. Then, we adopted ChatGPT-4o [27] as the expert model for traffic semantic analysis. we set the temperature to 0.5 to ensure precise and fact-based responses. All the LLMs used in IRPFuzz can be replaced with similar models without modifying the overall architecture.

To evaluate IRPFuzz, we conducted a set of experiments designed to answer the following questions:

RQ1: Can IRPFuzz effectively infer the state model and data model used in industrial robot protocol fuzzing?

RQ2: Can IRPFuzz efficiently discover potential vulnerabilities in industrial robot protocol?

RQ3: How do different LLMs affect the effectiveness of IRPFuzz?

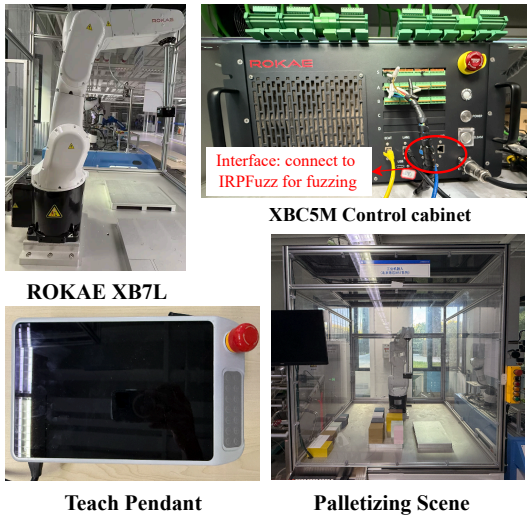

Fig. 6. Real Experimental Environment for Fuzzing.

A. Experimental Setup

We constructed a real experimental environment based on a ROKAE industrial robot to evaluate IRPFuzz, as shown in Fig. 6. The environment consists of a ROKAE six-axis robotic arm XB7L [32], coupled with an XBC5M control cabinet, a main controller running the RokaeStudio 5935 and an xPad2 teach pendant. To emulate a real-world industrial scenario, we implemented a palletizing workflow that integrates the industrial robot with a suction-type end effector, a MECHEYE NANO 3D camera, an industrial PC, and physical workpieces. The total cost of the experimental environment is approximately $35,000.

We executed IRPFuzz on a machine running Ubuntu 18.04 (64-bit), equipped with an Intel(R) Xeon(R) Gold 5120 CPU @ 2.20GHz, 125 GB RAM, two Tesla V100 GPUs, and a 1 TB HDD. The machine was connected to the same subnet as the ROKAE robot to facilitate communication.

Baselines. We compared IRPFuzz with the state-of-the-art black-box protocol fuzzers, including Boofuzz [11], PCFuzzer [33], and MSGFuzzer [17]. Boofuzz is widely adopted in general black-box protocol fuzzing, PCFuzzer focuses on fuzzing programmable logic controllers, and MSGFuzzer targets industrial robot protocols. These fuzzers have successfully discovered vulnerabilities in a wide range of real-world protocols. Since these fuzzers all rely on manually constructed data models, we applied the data model

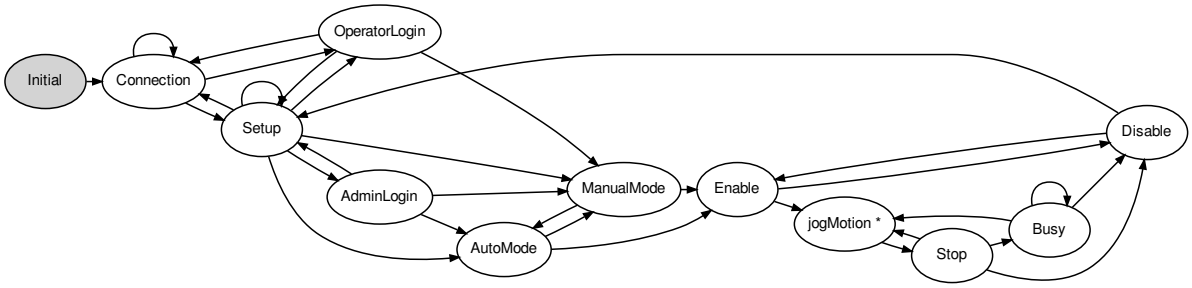

Fig. 7. State model of IRPFuzz after 24 hours of fuzzing. Due to the robot's multiple joints, coordinate systems, and motion directions, the LLM identified 14 distinct motion states. For simplicity and clarity, these states are denoted as jogMotion in the figure.

from MSGFuzzer to all baseline methods to ensure a fair evaluation. For comparison, we used number of crash as the metric, which indicates the ability to discover vulnerabilities. We ran each experiment for 24 hours and repeated it 10 times to establish statistical significance of results.

B. Experimental Results for RQ1

Table I presents the details of state model and data model inferred by IRPFuzz after 24 hours of fuzzing. For ROKAE industrial robot, IRPFuzz inferred a total of 24 states, 78 state transitions, and 78 corresponding request templates. The overall inferred state model is illustrated in Fig.7.

To evaluate the fidelity of the inferred state model, we individually replayed the request sequences associated with each state and systematically compared the resulting behaviors, as observed through both the teach pendant interface and the physical robot, against the expected state descriptions. The results show that IRPFuzz successfully reconstructed almost all operational states and transitions of the target robot, including some hidden states and transitions that are not normally exposed during standard operation. For example, IRPFuzz was able to discover a busy state that only appears when multiple motion commands are issued in rapid succession.

To assess the accuracy of the data model, we manually examined the inferred templates for field completeness and mutability. Thanks to our field validation mechanism, all templates satisfied the completeness constraint, and most field mutability annotations were accurate. Only a few fields were incorrectly marked, such as format symbols like "[" being labeled as immutable. However, such minor inaccuracies have negligible impact on the mutation process during fuzzing and are considered acceptable.

C. Experimental Results for RQ2

Fig. 8 shows the crash-triggering performance of IRPFuzz compared with baselines within 24 hour. Overall, IRPFuzz triggered 36 crashes, outperforming boofuzz by 157.14%, PCFuzzer by 89.47%, and MSGFuzzer by 16.13%, demonstrating its effectiveness in vulnerability discovery. Specifically, while IRPFuzz exhibited trends similar to other

TABLE I
STATE MODEL AND DATA MODEL CONSTRUCTED BY IRPFUZZ AFTER 24 HOURS OF FUZZING.

Metric	State Model		Data Model
	State	Transition	Template
Number	24	78	78

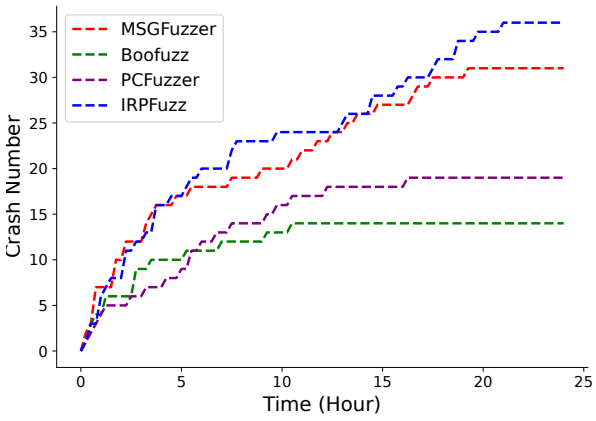

Fig. 8. Average Number of Crashes for IRPFuzz and Baselines.

methods in the early stage, it gradually surpassed them as fuzzing progressed. This improvement stems from the combination of state-model–guided fuzzing and data-model–driven structural mutation. Moreover, unlike boofuzz, PCFuzzer, and MSGFuzzer, which plateaued over time, IRPFuzz continued to show progressive improvements, highlighting the effectiveness of dynamically updating state model and data model rather than relying on static templates.

We reported the these crashes to the manufacturer for remediation and submitted them to the China National Vulnerability Database (CNVD) and the China Industrial Control System Vulnerability Database (CICSVD). To date, five vulnerabilities have been assigned official IDs, three of which are classified as high-severity, details are shown in Table II. Most of these vulnerabilities exhibit low attack

TABLE II

DETAILS OF VULNERABILITIES DISCOVERED BY IRPFUZZ.

Vulnerability ID	Description	Vulnerability Rating
CNVD-2024-11014	Denial of service	Medium
NVDB-CICSVD-2024239414	Denial of service	High
NVDB-CICSVD-2024360593	Denial of service	High
NVDB-CICSVD-2024587081	Denial of service	High
NVDB-CICSVD-2024691453	Denial of service	Medium

complexity and can fully compromise the availability of industrial robots.

D. Experimental Results for RQ3

TABLE III

PERFORMANCE OF IRPFUZZ USING DIFFERENT LLMS.

Subject	Model Parameters	State Model		Data Model	Crash
		State	Transition	Template	
IRPFuzz(ChatGPT-4o)	unknown	24	78	78	36
IRPFuzz(ChatGPT-4o-mini)	unknown	27	86	86	35
IRPFuzz(DeepSeek-R1)	671B	23	75	75	36
IRPFuzz(Qwen3-8B)	8B	36	104	104	33

To evaluate the impact of different LLMs on the effectiveness of IRPFuzz, we compared four representative LLMs, including ChatGPT-4o [27], ChatGPT-4o-mini [27], DeepSeek-R1 [34], Qwen3-8B [35]. These models vary in parameter size, encompassing both commercial and open-source models, and can be accessed via local deployment or API. They have been widely applied across various domains and in previous academic studies [36].

Table III presents the performance of IRPFuzz using different LLMs in terms of state model, data model, and crash discovery. Overall, all models perform well, successfully constructing state model and data model while guiding IRPFuzz to discover effective crashes. Specifically, models with larger parameter sizes, such as ChatGPT-4o and DeepSeek-R1, exhibit superior crash discovery capabilities and produce more compact and accurate state model and data model. Through manual analysis of the state model constructed by Qwen3-8B, we found that although it inferred the largest number of states and transitions, some states are redundant, which correspond to the same actual state during real industrial robot operation. These redundant states, along with their associated states and request templates, do not enhance fuzzing performance and instead reduce testing efficiency. Therefore, more capable models can better enhance the effectiveness of IRFuzz.

V. DISCUSSION

In this section, we discuss the manual work and the limitations of IRPFuzz.

A. Manual Work

Although IRPFuzz leverages LLM-driven traffic semantic analysis to avoid manual construction of state model and data model, we acknowledge that a small amount of manual effort is still required when adapting it to other devices, due to necessary preprocessing and the closed-source nature of proprietary protocols. This manual effort primarily involves operating the teach pendant software according to standard industrial robot operating procedures, which requires an operator familiar with the workflow of robot. The purpose of this step is to ensure the correctness of the operation sequence, thereby improving the efficiency of state model and data model initialization. It is noteworthy that, compared to existing methods, IRPFuzz has substantially minimized the manual effort involved in the fuzzing.

B. Limitations

Even though IRPFuzz has shown superior performance to existing fuzzers, our work still has the following limitations.

Depends on teaching operation. IRPFuzz requires collecting interaction traffic from teaching operations to initialize state and data models. Intuitively, a larger number of correct teaching operations can enhance the early-stage performance of fuzzing. However, section IV-B demonstrate that IRPFuzz can still discover industrial robot states beyond those included in the teaching operations, indicating that it does not require these operations to cover all possible states. Relying on prior knowledge is common in industrial protocol fuzzing, such as Boofuzz [11] and MSGFuzzer [17]. In comparison, conducting teaching operations is simpler and imposes lower requirements on initiating the fuzzing.

Number of evaluated devices. Due to budget constraints, we evaluated our approach using only one device. Such limitations are common in cyber-physical system testing and research [37]. Although industrial robot simulators may offer a relatively cost-effective testing solution, they cannot accurately emulate the inherent physical state transitions involved in real industrial robot operations. Therefore, simulators cannot fully replace real industrial robots in testing scenarios. Nevertheless, the device we used is representative, as its hardware architecture and operational workflow are largely consistent with those of other industrial robots. Our approach requires minimal manual intervention and can be easily adapted to other devices. In future work, we plan to extend our evaluation to a broader range of devices.

VI. CONCLUSION

In this paper, we present IRPFuzz, a state-aware fuzzer for industrial robot protocols. IRPFuzz leverages LLM-driven traffic semantic analysis to automatically construct protocol state model and data model, which in turn guide state-aware fuzzing and structured message mutation. During fuzzing, these models are iteratively refined to enhance effectiveness of fuzzing. We evaluated IRPFuzz in a real-world experimental environment using a ROKAE six-axis industrial robot. The results show that IRPFuzz discovered 36 crashes, outperforming Boofuzz by 157.14%, PCFuzzer by 89.47%, and MSGFuzzer by 16.13%. Notably, five vulnerabilities have been confirmed and assigned vulnerability IDs, demonstrating the effectiveness of IRPFuzz.

REFERENCES

[1] Honeywell. (2025, 7) Industrial ransomware attacks surge, honeywell reports. IoT World Today. Accessed: 2025-08-14. [Online]. Available: https://www.iotworldtoday.com/security/industrial-ransomware-attacks-surge-honeywell-reports

[2] (2023, 10) Stuxnet. Wikipedia. Version ID: 1178930938. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Stuxnet&oldid=1178930938

[3] A. Campbell. (2022, 7) Report: The state of industrial security in 2022. Barracuda Blog. [Online]. Available: https://blog.barracuda.com/2022/07/12/report-the-state-of-industrial-security-in-2022

[4] D. Quarta, M. Pogliani, M. Polino, F. Maggi, A. M. Zanchettin, and S. Zanero, "An experimental security analysis of an industrial robot controller," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 268–286.

[5] S. Jiang, Y. Zhang, J. Li, H. Yu, L. Luo, and G. S, "A survey of network protocol fuzzing: Model, techniques and directions," *arXiv preprint arXiv:2402.17394*, 2024.

[6] R. Baldoni, E. Coppa, D. C. D'elia, C. Demetrescu, and I. Finocchi, "A survey of symbolic execution techniques," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, pp. 1–39, 2018.

[7] M. M. Tikir and J. K. Hollingsworth, "Efficient instrumentation for code coverage testing," *ACM SIGSOFT Software Engineering Notes*, vol. 27, no. 4, pp. 86–96, 2002.

[8] K. Alshmrany and L. Cordeiro, "Finding security vulnerabilities in network protocol implementations," *arXiv preprint arXiv:2001.09592*, 2020.

[9] V.-T. Pham, M. Böhme, and A. Roychoudhury, "Aflnet: a greybox fuzzer for network protocols," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 2020, pp. 460–465.

[10] R. Natella, "Stateafl: Greybox fuzzing for stateful network servers," *Empirical Software Engineering*, vol. 27, no. 7, p. 191, 2022.

[11] J. Pereyda, "Boofuzz: Network protocol fuzzing for humans," https://github.com/jtpereyda/boofuzz, 2025, gitHub repository. Accessed: 2025-07-01.

[12] P. Fuzzer, "Peach fuzzer," 2017.

[13] S. Qin, F. Hu, Z. Ma, B. Zhao, T. Yin, and C. Zhang, "Nsfuzz: Towards efficient and state-aware network service fuzzing," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 6, pp. 1–26, 2023.

[14] D. Fang, Z. Song, L. Guan, P. Liu, A. Peng, K. Cheng, Y. Zheng, P. Liu, and et al., "Ics3fuzzer: A framework for discovering protocol implementation bugs in ics supervisory software by fuzzing," in *Proceedings of the 37th Annual Computer Security Applications Conference*, 2021, pp. 849–860.

[15] Z. Luo, K. Liang, Y. y. Zhao, F. Wu, J. Yu, H. Shi, and Y. Jiang, "Dynpre: Protocol reverse engineering via dynamic inference," in *Proc. NDSS*, 2024, pp. 1–18.

[16] J. Jiang, X. Zhang, C. Wan, H. Chen, H. S, and T. Su, "Binpre: Enhancing field inference in binary analysis based protocol reverse engineering," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 3689–3703.

[17] Y. Zhang, D. Fang, P. Liu, and et al., "Msgfuzzer: Message sequence guided industrial robot protocol fuzzing," in *2024 IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2024, pp. 140–150.

[18] S.-G. Lee, "Vocabulary standard for robotics in iso," in *Proceedings of the 7th International Conference on Robot Ethics and Standards, Seoul, South Korea*, 2022, pp. 18–19.

[19] P. Bilancia, J. Schmidt, R. Raffaeli, M. Peruzzini, and M. Pellicciari, "An overview of industrial robots control and programming approaches," *Applied Sciences*, vol. 13, no. 4, p. 2582, 2023.

[20] C. Urrea and J. Kern, "Recent advances and challenges in industrial robotics: A systematic review of technological trends and emerging applications," *Processes*, vol. 13, no. 3, p. 832, 2025.

[21] S. Abhishek, Y. S. Jogi, U. K. Sahu, S. K. Dash, and U. K. Yadav, "Teach pendant at fingertips: Intuitive vision-based gesture-driven control of dexter er2 robotic arm," *IEEE Access*, 2025.

[22] C. y. Zheng, Y. Wang, H. Huang, Y. Wang, H. Chen, and Q. Wei, "Survey of network protocol fuzzers: Taxonomy, techniques, and directions," *Techniques, and Directions*.

[23] R. Meng, V.-T. Pham, M. Böhme, and A. Roychoudhury, "Aflnet five years later: On coverage-guided protocol fuzzing," *IEEE Transactions on Software Engineering*, 2025.

[24] R. Meng, M. Mirchev, M. Böhme, and A. Roychoudhury, "Large language model guided protocol fuzzing," in *Proceedings of the 31st Annual Network and Distributed System Security Symposium (NDSS)*, vol. 2024, 2024.

[25] B. P. Miller, L. Fredriksen, and B. So, "An empirical study of the reliability of unix utilities," *Communications of the ACM*, vol. 33, no. 12, pp. 32–44, 1990.

[26] H. y. Wei, L. Chen, Z. Du, Y. Wu, H. Huang, Y. Liu, G. Cheng, F. Xu, L. Wang, and B. Mao, "Unleashing the power of llm to infer state machine from the protocol implementation," *arXiv preprint arXiv:2405.00393*, 2024.

[27] OpenAI. (2025) Chatgpt. [Online; accessed: Aug. 29, 2025]. [Online]. Available: https://chat.openai.com/

[28] R. Soepeno, "Wireshark: An effective tool for network analysis," *CYBV-Introd. Methods Netw. Anal*, pp. 1–15, 2023.

[29] L. McInnes, J. Healy, S. Astels *et al.*, "hdbscan: Hierarchical density based clustering." *J. Open Source Softw.*, vol. 2, no. 11, p. 205, 2017.

[30] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM computing surveys*, vol. 55, no. 9, pp. 1–35, 2023.

[31] E. S. Ristad and P. N. Yianilos, "Learning string-edit distance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 5, pp. 522–532, 2002.

[32] Rokae Robotics. (n.d.) Xb7 rokae robotics. leading robots expert in industrial, commercial scenarios. [Online; accessed: Aug. 29, 2025]. [Online]. Available: https://www.rokae.com/en/product/show/239/XB7.html

[33] P. Liu, Y. Zheng, Z. Song, D. Fang, S. Lv, and L. S, "Fuzzing proprietary protocols of programmable controllers to find vulnerabilities that affect physical control," *Journal of Systems Architecture*, vol. 127, p. 102483, 2022.

[34] D. Guo, D. Y, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zh, S. Ma, P. Wang, X. Bi *et al.*, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.

[35] A. Y, A. Li, B. Y, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv *et al.*, "Qwen3 technical report," *arXiv preprint arXiv:2505.09388*, 2025.

[36] J. Zhang, H. Bu, H. Wen, Y. Liu, H. Fei, and et al., "When llms meet cybersecurity: A systematic literature review," *Cybersecurity*, vol. 8, no. 1, p. 55, 2025.

[37] Y. Chen, B. Xuan, C. M. Poskitt, J. S, and F. Zhang, "Active fuzzing for testing and securing cyber-physical systems," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2020, pp. 14–26.