

# LLMPot: Dynamically Configured LLM-based Honeypot for Industrial Protocol and Physical Process Emulation

Christoforos Vasilatos, Dunia J. Mahboobeh, Hithem Lamri, Manaar Alam, and Michail Maniatakos

*Center for Cyber Security, New York University Abu Dhabi, UAE*

*{cv43, dam10098, hl5743, alam.manaar, mihalis.maniatakos}@nyu.edu*

**Abstract**—Industrial Control Systems (ICS) are extensively used in critical infrastructures ensuring efficient, reliable, and continuous operations. However, their increasing connectivity and addition of advanced features make them vulnerable to cyber threats, potentially leading to severe disruptions in essential services. In this context, honeypots play a vital role by acting as decoy targets within ICS networks, or on the Internet, helping to detect, log, analyze, and develop mitigations for ICS-specific cyber threats. Deploying ICS honeypots, however, is challenging due to the necessity of accurately replicating industrial protocols and device characteristics, a crucial requirement for effectively mimicking the unique operational behavior of different industrial systems. Additionally, the difficulty is increased by the substantial manual effort involved in replicating the PLC’s control logic. This is necessary to capture attacker traffic that seeks to interfere with critical infrastructure operations. In this paper, we propose LLMPot, a novel approach for designing honeypots in ICS networks harnessing the potency of Large Language Models (LLMs). LLMPot aims to provide a dynamic framework that can be used to optimize the creation of realistic honeypots with vendor-agnostic configurations and for various control logic, aiming to eliminate the manual effort and specialized knowledge traditionally required by existing strategies. We conducted extensive experiments focusing on a wide array of parameters, demonstrating that LLMPot can effectively create honeypot devices implementing different industrial protocols, PLC configurations, and diverse control logic.

## 1. Introduction

Contemporary industrial frameworks incorporate vital infrastructures, such as Programmable Logic Controllers (PLCs), which serve as integral components of Industrial Control Systems (ICS) deployed across various sectors including power generation and distribution, water treatment plants, and manufacturing facilities, among others. Because of the essential role these infrastructures play, ICS have become a frequent target for cyber attacks [35], while at the same time researchers are developing tools to uncover vulnerabilities [54], [16], [55]. With the heightened interconnectivity of ICS devices and the expanded attack surface resulting from the numerous capabilities provided by modern ICS, attackers have numerous potential entry points into these systems.

A honeypot serves as a security mechanism specifically designed in a controlled and isolated environment to improve the cyber security posture of ICS devices [42],

[17], [38], [56]. The objective of honeypots is to mimic the functionalities of real ICS devices so convincingly that they attract and redirect malicious activities away from legitimate devices. This allows security professionals to monitor and analyze intercepted network traffic, providing insight into attacker methods and improving the security of critical infrastructures. Therefore, the primary challenge lies in developing a honeypot that closely mimics the behavior of ICS devices with such precision that it becomes indistinguishable from the actual devices, ensuring its effectiveness as a security decoy.

**Limitations and Challenges.** Existing ICS honeypot frameworks primarily focus on emulating industrial protocols of specific PLC devices or given PLC setups. That is, predefined analog and digital inputs are available and/or a predefined control logic that the PLC is capturing. In order to develop such a honeypot, extensive domain knowledge and implementation capabilities are required; therefore, related works base their honeypots on existing frameworks for protocol emulation [29], [12]. Besides the industrial protocol emulation, ICS would typically control some physical process, therefore realistic honeypots should also support some form of emulation of an underlying physical process. Related work either does not support physical process emulation [18], [59], [33], [49], or their capabilities tend to be restricted to very specific scenarios (e.g., water treatment processes [11]) and lack the flexibility to modify or extend the emulated control logic [32], [19]. Current emulation strategies, which rely on static scripting or fully-fledged applications [33], often fail to cover all possible scenarios and do not give the user the ability to alter the behavior of the honeypot. Therefore, the main challenge of efficient honeypot development is the tedious manual work needed to emulate the hardware, the industrial protocol that depicts the unique PLC configuration, and the control logic. Especially the protocol emulation needs deep understanding of protocol manuals and details that are hard to be depicted accurately and exhaustively in a rule-based program.

**Why Large Language Models (LLMs).** While LLMs have primarily been applied to natural language and code generation, their potential for generating network traffic (bytes) remains largely unexplored. LLMs excel at generating sequences of responses to queries. In this context, queries and responses correspond to network packets. With proper training, LLMs can generate precise, context-aware network responses, making them ideal core components for ICS honeypots. In other words, LLMs

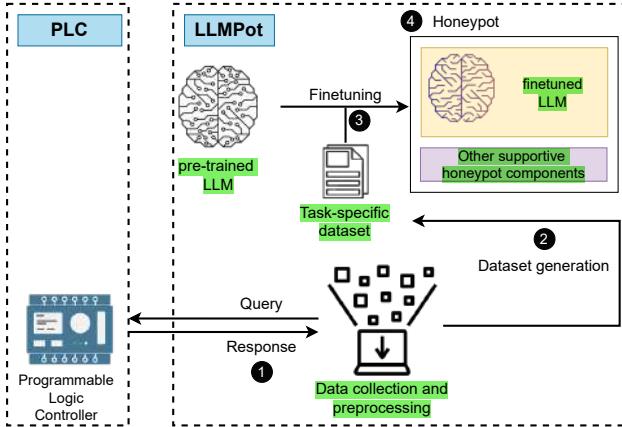


Figure 1: LLMPot high-level diagram. 1. A client that automatically probes the PLC and captures responses. 2. Captured traffic forms a training dataset. 3. Fine-tuning of the LLM using the dataset. 4. Generated LLM-based honeypot with supportive components.

have the potential to significantly reduce the manual effort required to emulate hardware, protocols, and industrial/physical processes in honeypot development. Our framework establishes a specialized interaction with PLCs to generate customized datasets optimized for training LLMs as network emulators. Traditional methods often require implementing detailed physical process simulations using complex tools such as Simulink/MATLAB, Hardware-in-the-Loop (HIL) simulations, and extensive client-side implementations for testing. LLMPot takes a different approach: instead of relying on pre-existing physical process models, it collects traffic by probing PLCs with targeted requests using only the client-side implementation, decoupling from server complexity. This probing can be performed on both simulated and real-world physical processes. This method offers significant advantages, including adaptability to diverse physical process emulation and freeing engineers and researchers from the time-consuming task of manually developing process simulations.

Therefore, in this paper, we introduce LLMPot, an LLM-based approach for protocol and physical process emulation, leveraging pre-trained LLMs. To the best of our knowledge, this is the first work where pre-trained LLMs are used to emulate ICS protocols, physical processes, and control logic installed in PLCs. The use of pre-trained machine learning models offers two advantages. First, it significantly reduces resource consumption and eliminates the need for costly and time-intensive training from scratch. Second, these pre-trained LLMs can be efficiently fine-tuned with a relatively small and domain-specific dataset, enhancing their ability to generate specialized and accurate responses.

In conclusion, this effort aims to enable users to customize the analog/digital configuration or the control logic running on a PLC on demand and thus create a more realistic and adaptive honeypot that meets the user's requirements. As Figure 1 shows, our methodology enables “cloning” of an industrial protocol given a specific PLC configuration on a specific port (example, “Modbus:502”) and instantiates it as an LLMPot instance. After LLM-

Pot has been successfully trained, it can be deployed as a single or multiple instances with different public IPs to capture diverse attacker traffic. The PLC device is used only during the offline phase to build the honeypot and is not needed during the honeypot deployment.

Our contributions can be summarized as follows:

- A novel dynamic framework for generating a customized dataset that **captures the PLC device-specific setup including the analog and digital configuration**. The approach is flexible and can support a diverse set of industrial protocols.
- A platform for fine-tuning LLMs using the previously generated dataset, resulting in a cloned PLC device that accurately emulates both the ICS network protocol and control logic. This fine-tuned model can then serve as the emulator module within an ICS honeypot setup.
- Three new metrics for evaluating the performance of LLMs as ICS emulators: *Byte-to-byte Comparison Accuracy* (BCA), *Response Validity Accuracy* (RVA), and *Response Validity Accuracy - Epsilon* (RVA-e).
- The source code, models, and datasets are open source at <https://github.com/momalab/LLMPot>.

## 2. Background

A key component in ICS networks is the PLC, a compact industrial computer programmed to perform logical operations based on inputs from sensors, relays, switches, and others to control actuators, valves, and more. Prominent PLC vendors include Allen-Bradley, Siemens, WAGO, and ABB. ICS communication protocols like Modbus [36], S7comm [58], EtherCAT [23], and Profinet [43] have been developed to enable reliable and efficient data exchange within these PLCs to monitor physical processes across various sectors.

Table 1 depicts the packet frame formats for two different industrial communication protocols: Modbus and S7comm. The standard Modbus protocol is wrapped in a TCP/IP layer, enabling communications over Ethernet networks. Modbus query **a** and response **b** packet frames contain a Modbus Application Protocol (MBAP) header and a Protocol Data Unit (PDU). On the other hand, the S7comm query **c** and response **d** packet frames generally include a header, a parameter, and a data field. Headers in both protocols are crucial for correlating queries with responses by matching several identifiers (e.g., Transaction ID, Protocol ID, etc.) and verifying frame length. A key element in both protocols is the function code (FC) in the PDU or parameter section, which dictates the command or operation a client requests to the server. The data fields, which depend on the FC, detail the variables involved and their count. In both protocols, the data is represented as a series of digital and analog input/output; therefore, both protocols support functions for reading and writing to these data types.

## 3. Preliminaries

### 3.1. Research Questions

Using an LLM to emulate industrial protocols and physical processes involves intricate tasks. For industrial

TABLE 1: Modbus and S7 Communication Frames

Modbus Communication								
Modbus Application Protocol (MBAP) Header				1B Protocol Data Unit (PDU)				
Transaction ID	Protocol ID	Length	Unit ID	Function Code (FC)	Data			
00 01	00 00	00 06	00	03	00 01	00 01	00 00	00 00 00 01
00 01	00 00	00 05	00	03	00 02	00 00	00 00	02 00 00

S7 Communication - Query								
Header				Parameter				
Protocol ID	ROSCTR	Redundancy ID	(PDU) Reference	Parameter length	Data length	Function Code (FC)	Item count	Items
32	01	00 00	01 00	00 0e	00 00	04	01	12 0a ..

S7 Communication - Response								
Header				Parameter				
Protocol ID	ROSCTR	Redundancy ID	(PDU) Reference	Parameter length	Data length	Error class/code	Function Code (FC)	Item count
32	03	00 00	01 00	00 02	00 04	00 00	04	01 05 ..

protocols, it is essential to capture and replicate the complex interactions between devices accurately. Likewise, for physical processes, accurately modeling variable changes is important. These efforts require careful data preparation for the model training, which presents various research questions in achieving realism and maintaining accuracy:

- **RQ1: Protocol Emulation Challenge:** How can an LLM emulate the behavior of an ICS protocol with its wide range of functions and values? Can it provide realism by incorporating exception handling, and to what extent? Can the dataset generation be extended to other protocols and dynamically configured to adapt to different PLC configurations and circumstances? [Section 5]
- **RQ2: Process Emulation Challenge:** How can an LLM learn to emulate the control logic of industrial physical processes and accurately capture the states of the variables? What is the limit of complexity for control logic that this framework can handle without failing? [Sections 6 and Section 7]
- **RQ3: Honeypot Efficiency:** How effective is the generated honeypot? Does it reflect the PLC device's authentic characteristics upon scanning and disguise itself as a realistic device? Can the honeypot deceive network reconnaissance tools? [Section 9]

### 3.2. LLM Evaluation Metrics

A key element of ICS honeypots is their interactivity, which relies on their ability to respond accurately to requests with appropriate data. Since LLMPot utilizes machine learning models to emulate the ICS device network communications we define three different metrics in order to assess LLMPot's ability to respond appropriately:

#### Definition 3.1: Byte-to-byte Comparison Accuracy (BCA)

A byte-to-byte comparison of the expected response with the response generated from the fine-tuned LLM. The total accuracy is a percentage of the exact correct responses over the total requests sent.

The BCA metric measures whether the LLM can produce the exact response as the PLC for the same request, tested byte-by-byte. While BCA is a straightforward way to assess LLM's accuracy, sometimes the bytes may not match for various reasons, for example, the physical process returns a different integer value despite the response packets being valid. Thus, a new metric, RVA, is defined to measure the ability of the LLM to respond with valid packets.

#### Definition 3.2: Response Validity Accuracy (RVA)

A metric that measures if a produced response from the LLM is a valid network packet (see Table 1) in terms of a protocol's specific fields, like *transaction id* or *function code*, that ignores the actual value of the analog or digital inputs/outputs and only validates if they are within the protocol defined range.

Merely producing a valid network packet that is meaningless can be easily detected as a honeypot. Typically, attackers check if they are receiving a valid response and rarely further analyze the data fields of the response to understand whether the underlying physical process produces expected responses. For sophisticated attackers that would do further analysis, BCA needs to be maximized, as RVA maximization may not be sufficient. However, since the expected responses are based on the underlying process that could include noise and disturbances, they can potentially differ by some amount  $\epsilon$ . This variation can still mislead the attacker into thinking a real process is being controlled. Therefore, we introduce a new metric RVA- $\epsilon$  that captures this "epsilon" difference.

#### Definition 3.3: RVA-Epsilon (RVA- $\epsilon$ )

RVA- $\epsilon$  expands the RVA metric by also checking the actual value of analog or digital inputs/outputs. The response is considered accurate if the value is within  $\pm \epsilon$  of the actual value.

The following equations stand given Definitions 3.1, 3.2, and 3.3.

$$\text{RVA-0} = \text{BCA}, \text{RVA-}\infty = \text{RVA}, \text{BCA} \leq \text{RVA-}\epsilon \leq \text{RVA} \quad (1)$$

### 3.3. Base LLM

Generally, LLMs excel in generative tasks, primarily through a tokenizer, which converts incoming text into tokens that can then be used to formulate the response. Effective tokenization for network protocols would require a deep understanding of each protocol, considering the structured layers of packets, each with its unique fields and parameters. Thus, creating a universal tokenizer capable of handling multiple protocols poses significant challenges. A machine learning model capable of understanding network packets and generating responses without requiring tokenization is needed.

The ByT5 model [60], developed by Google, introduces a novel approach to natural language processing by operating at the byte level, diverging from traditional token-based architectures. By processing text as a sequence of UTF-8 encoded bytes, ByT5 can handle a diverse range of languages and special characters without relying on a predefined vocabulary. This byte-level processing enables the model to manage inputs and outputs simply as byte sequences. Given this capability, if we consider only a specific ICS network protocol layer within the entire TCP packet stack, we can input just this segment to the ByT5 model and receive a response without needing to tokenize the packet. The finetuning process only needs two inputs, the source and target texts. In the context of LLMPot, this will correspond to the hexadecimal format of the request and response packets respectively. This method highlights the clear advantages of choosing the

ByT5 model as the base LLM for LLMPot, as it eliminates the need for tokenization.

## 4. LLMPot Overview

The basic components used in the offline process are described in the following subsections, while a graphical representation of them is depicted in Figure 2.

### 4.1. Dataset Generation Process

The efficacy of fine-tuning LLMs for specific tasks heavily relies on the quality of the dataset used. Ensuring diversity across various scenarios is crucial in dataset generation, as it prevents model bias and enhances its ability to generalize across different conditions, leading to more robust performance.

**4.1.1. PLC Profile.** Step ① in Figure 2, illustrates the flow diagram of LLMPot's dataset generation process based on a user defined protocol specific configuration file. This file essentially describes the PLC device configuration and its parameters which are crucial for constructing a customized/user-specific protocol client.

**4.1.2. Protocol Client Initialization.** *Protocol* attribute in the configuration file, specifies the communication layer that the PLC uses. The next attribute is the list of *function* codes that are supported. Each function corresponds to a single or multiple read/write commands to a specific data type, digital or analog inputs/outputs, or device information. For example, the packet sample in Table (1, ②) shows the FC as “03”, which is decoded to read analog data from a specific address in Modbus, while FC “04” in Table (1, ③) is decoded to read the data address from a specific data block area in S7comm, which are part of the official protocol documentation [36]. The ranges defined for *analog\_addresses* and *digital\_addresses* attributes determine the available accessible areas within the PLC memory.

**4.1.3. PLC Probing.** After the initialization, the protocol client establishes a connection to the target PLC device, allowing interaction through various request/response commands (step ④). Subsequently, an extensive network interaction traffic between the protocol client and the PLC is captured using a tool like Unix *tcpdump* [52] (step ⑤). The captured network traffic is stored in a packet capture (PCAP) file, which is sent as input to a parser (step ⑥) to extract the request and response commands.

**4.1.4. PCAP Parser.** The parser is specifically designed to convert the raw binary data from the pcap file into hexadecimal byte strings. The packets might not be in sequence in the pcap file, thus the parser needs to correlate responses with their corresponding requests to ensure the correct interactive sequences. The data are saved in a CSV format file (step ⑦) that contains the request followed by the response.

**Context-Sensitive Dataset Generation:** For protocol emulation, historical data is not needed in the dataset. The dataset provides correct functionality effectively with individual requests and their corresponding responses

since each pair is independent. However, when emulating physical processes, the response to a request might be affected by changes from previous interactions. Thus, for this purpose, the dataset must include contextual information to reflect the dynamic and interconnected nature of these events accurately. As a result, the PCAP parser includes an additional pre-processing step that organizes data into a context-aware format, where the dataset would look as follows:

$$req_1 : resp_1 | req_2 : resp_2 | \dots | req_{n-1} : resp_{n-1} | req_n : resp_n$$

where  $req_i$  and  $resp_i$  represent the  $i$ -th request and its corresponding response, respectively, and  $n \geq 2$ . This structured dataset helps in fine-tuning the LLM to produce context-aware responses and adjust to new queries based on past interactions.

**Automation:** Figure 2 highlights the module that needs manual effort with green color, “protocol client” while the remaining modules are fully automated. The manual intervention in the “protocol client” module is required in case of adapting new protocols which is further discussed in Section 5.1. Regarding the JSON configuration file, the user can configure it to match the desired PLC profile.

### 4.2. LLM Fine-tuning Process

LLMPot utilizes the PyTorch Lightning framework [5] to facilitate a structured method for model training, including efficient dataset management, validation during training, and post-training testing cycles. This framework has been adapted to fine-tune a pre-trained ByT5 model (e.g., google/byt5-small), which we use as our base LLM. Performance of the fine-tuned LLM is measured explicitly as how accurately the model produces replies to specific requests (see Definitions 3.1, 3.2, and 3.3).

**4.2.1. Model Fine-tuning.** The adopted LLM fine-tuning process is depicted in step ⑧. The process begins with the dataset distributor module, which inputs the generated dataset into the trainer module. Simultaneously, the trainer module loads the pre-trained base LLM and initiates the fine-tuning phase. The trainer module performs a test cycle at the end of each training epoch, enabling continuous evaluation against a test dataset to monitor the actual model performance with an early stopping, where the fine-tuning phase is terminated if there is no improvement. The PCAP Parser dictates the value of ByT5 model’s *source\_max\_token\_len* and *target\_max\_token\_len* hyperparameters, which define the maximum input and output lengths respectively.

**4.2.2. Iterative Dataset Sizing.** An appropriate dataset size selection is crucial when fine-tuning. A dataset that is too small may not fully capture the complexity of the task, leading to under-fitting. In contrast, an excessively large dataset can lead to over-fitting and unnecessary computational expenses. Therefore, the fine-tuning process begins with a small dataset size and iteratively doubles as the fine-tuned model is evaluated using the RVA- $\epsilon$  metric (step ⑨). This is done through a dataset expansion module requesting additional data from the protocol client (step ⑩). The model is further fine-tuned with each increase in dataset size until no further improvements in

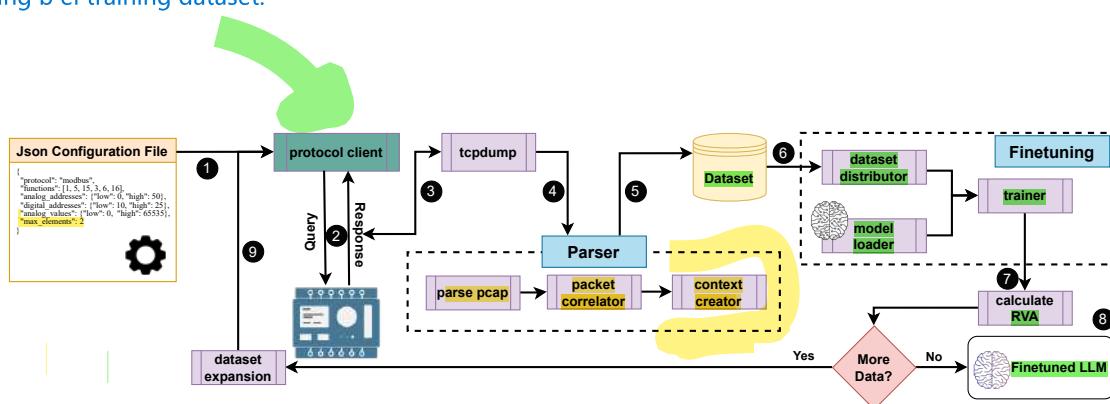


Figure 2: LLMPot’s offline stage framework for dynamic configuration and PLC cloning. The process is fully automated, except for the box marked in the green (“protocol client”), which has to be performed manually for every new protocol.

the RVA- $\epsilon$  metric are observed across multiple consecutive iterations.

This approach provides a controlled and systematic exploration of the relationship between dataset size and model performance, identifying a point of diminishing returns where additional data no longer contributes to better model performance. The framework returns the fine-tuned LLM that effectively emulates the target PLC device.

## 5. LLMPot ICS Network Protocol Emulation

stopped here

Our initial goal is to fine-tune a pretrained LLM capable of accurately responding to network packets based on the network protocol used by the PLC, tailored to a specific device configuration. The dataset generation in this section will not encompass context, as historical data is not required for protocol emulation experiments.

### 5.1. Dataset Generation Methodology

To generate a representative dataset that emulates a specific network protocol, for example, Modbus protocol, an exhaustive dataset that consists of 65536<sup>2</sup> (the maximum possible number of registers times the available values stored in 2 bytes per register) different requests is needed to cover all registers and their potential values for just one type of command (e.g., read or write). Following an exhaustive dataset generation strategy would demand significant resources for training. Therefore, Algorithm 1 implements the *protocol\_client* shown in Figure 2 which also describes the proposed *boundaries* method. This method allows us to obtain a dataset that describes a protocol in a non-exhaustive manner while preserving accuracy.

By employing the *boundaries* method, we significantly reduce the dataset size from a scale of billions to just a few thousand. This is achieved by covering the entire address and value space supported by a protocol with only selecting the minimum, maximum, and one randomly chosen intermediate value.

The method starts with the *execute* function [line: 10], which essentially probes the PLC with the read/write commands included in the user defined *functions* list [line: 18]. The initial loop [line: 11] iterates through values from one to *m\_elem* [line: 2], setting how many elements are going to be used from the selected read/write functions in the current loop. The second loop [line: 14] iterates over

### Algorithm 1 LLMPot Protocol Emulation using *boundaries*

Require:

```

1: read, write: abstract functions specific to the network protocol.
2: m_elem (max_elements): the number of elements up to which to
   read or write using only one command.
3: a_low, a_high: the range of addresses available according to the
   protocol running on the PLC.
4: max_addr: max address supported by PLC-protocol combination.
5: val_low, val_high: the range of values that the PLC is configured
   to reply with normal responses.
6: bounds: the range for normal operations
7: e_bounds: the range of values for exception cases (address out of
   bounds)
8: functions: the list of protocol specific functions to call.
9:
10: function EXECUTE
11:   for elem in range(1, m_elem) do
12:     bounds  $\leftarrow$  TRIPLET(a_low, a_high, m_elem)
13:     e_bounds  $\leftarrow$  TRIPLET(a_high+1, max_addr, m_elem)
14:     for addr in [bounds, e_bounds] do
15:       cartesian  $\leftarrow$  COMBS(val_low, val_high, elem)
16:       for data in cartesian do
17:         for FUNC in functions do
18:           FUNC(addr data, elem)
19:         end for
20:       end for
21:     end for
22:   end for
23: end function
24:
25: function COMBS(low, high, elem)
26:   bounds  $\leftarrow$  TRIPLET(low, high, 0)
27:   combs  $\leftarrow$  CARTESIAN_PRODUCT(bounds, elem)
28:   result  $\leftarrow$  empty dictionary
29:   for comb, index in combs do
30:     result[index]  $\leftarrow$  comb
31:   end for
32:   return result
33: end function
34: function TRIPLET(low, high, elem)
35:   return [low, RAND(low+1, high-elem-1), high-elem]
36: end function
37: 
```

this generate 3 elements represent the range with min, max, and random in between

values derived from the *triplet* function [line: 35]. This function generates three specific values: the lowest, the highest minus the provided elements, and a random number in between, which forms the basis of the *boundaries* method. To add a layer of realism, this method includes requests that trigger erroneous responses (i.e., exceptions). Exceptions can occur when a request is made using an inactive address outside the valid range. This approach allows the dataset to reflect typical operational data and include scenarios that simulate potential error conditions.

Before the loop [line: 16], the *combs* func-

tion [line: 25] is called to produce all possible data combinations based on the given range of possible values and the number of elements to accommodate. *Combs* function starts by calling the *triplet* [line: 35] function to obtain a set of three numbers. These numbers are used to compute the Cartesian product of the specified number of elements [line: 27], which is later used in the request commands that include multiple inputs. By that, all possible combinations of values for the number of elements are covered. The resulting values are formatted into triplets to be used in the *execute* function. The *execute* function then sends requests to a PLC or simulator, which is called repeatedly to align with the dataset size in the configuration file.

**Why boundaries method** This algorithm significantly enhances the adaptability, scalability, and efficiency of ICS honeypots by dynamically generating training data across a range of operational states. This method ensures a smart and effective dataset generation for training the LLM to understand the range of accessible addresses/values without the need to generate traffic that includes them. This helps in making the framework more dynamic and adaptable to various PLC configurations, moving away from the traditional reliance on static and manual scripting specific to PLC setups.

**Adapting New Protocols** Algorithm 1 serves as the foundation for integrating a “protocol client” module (Figure 2) for other PLC protocols. Step 1: Identify the terminology used in the new protocol and map it to the corresponding terms in the existing implementations. For example, Modbus uses “registers” for analog input/output, while S7comm employs “datablocks”. Step 2: Identify the valid addresses and value ranges of the new protocol. Step 3: Identify the list of available function codes. Step 4: Modify the default configuration file to include the identified addresses and values into the analog inputs/outputs. Step 5: Instantiate Algorithm 1 per our examples existing in our codebase using readily available libraries to generate the protocol client module. The rest of the methodology is automated.

## 5.2. Fine-tuning LLM

**5.2.1. Ablation Study on input/output configuration.** To prove LLMPot’s independence of different input/output configurations, an ablation study was performed experimenting with powers of 2 to cover the maximum available inputs/outputs, 65536 which is  $2^{16}$ . Given so, Figure 3 presents the experiment starting with  $2^2$  up until  $2^{16}$ , excluding  $2^0$  and  $2^1$  as boundaries cannot be applied in this case, preventing any significant dataset minimization. This experiment further demonstrates that choosing any input/output configuration does not impact the BCA and RVA metrics or the overall performance of the LLM on the selected protocol. Both metrics exhibit the same trend, with variations attributed solely to the stochastic nature of LLMs. The experiments performed in the following sections use a setup that is attributed to specific hardware-in-the-loop testbeds employed at our lab.

**5.2.2. Ablation Study on model and dataset sizes.** We run the protocol emulation experiments for byt5-small, byt5-base, and Llama3.2 [53] which have 300 million,

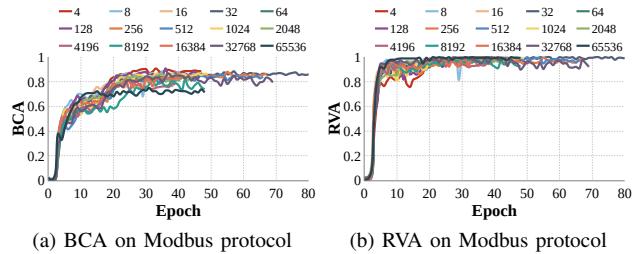


Figure 3: BCA and RVA per epoch of the byt5-small model when using different input/output configurations ( $2^2$ : 4 up to  $2^{16}$ : 65536). Similar values on the metrics prove that LLMPot is independent of the PLC configuration.

TABLE 2: Ablation study on the size of ByT5 model.

Size	Model	BCA	RVA
200	small	$59.50 \pm 6.35$	$85.00 \pm 7.53$
	base	$71.13 \pm 5.29$	$96.43 \pm 3.96$
400	small	$71.94 \pm 8.27$	$93.33 \pm 5.59$
	base	$73.04 \pm 7.92$	$91.25 \pm 8.13$
800	small	$85.83 \pm 0.88$	$98.47 \pm 1.95$
	base	$85.00 \pm 5.96$	$96.88 \pm 6.14$
1600	small	$85.31 \pm 1.25$	$99.30 \pm 0.62$
	base	$82.55 \pm 3.97$	$95.87 \pm 4.79$
3200	small	$80.27 \pm 0.97$	$97.77 \pm 1.42$
	base	$79.67 \pm 2.91$	$97.34 \pm 2.83$
*6400	small	$82.62 \pm 1.49$	$98.81 \pm 1.10$
	base	$82.74 \pm 2.10$	$98.91 \pm 1.31$

\*Dataset sizes above 3200 samples experience overfitting.

580 million, and 1 billion parameters respectively. Table 2 depicts that there is no advantage to using the bigger model (byt5-base), especially for dataset sizes above 1600 samples, as the results are relatively close. As the dataset size increases above 3200 samples, overfitting is experienced and thus we see a drop in performance of both byt5 model versions. The fact that byt5-small performs marginally worse in the 6400 samples dataset in terms of BCA and RVA is not particularly significant due to the impact of overfitting. The main objective of LLMPot is to fine-tune the LLM with a small dataset that gives us good metric values due to the boundaries method.

On the other hand, Llama3.2 BCA and RVA metrics were almost zero with very few correct emulated packets. This can be attributed to the tokenization need for this type of model. Finally, we use LoRA [27] to minimize training time on the byt5-small model by training 40 million instead of 300 million parameters, but this also yields poor results which can be attributed to the fact that we re-purpose the model’s objective in generating hexadecimal responses instead of doing language translation.

**Takeaway 1:** There is no need to move to a bigger model since similar results, on the RVA metric, can be achieved using different dataset sizes and no substantial advantage will be gained by the additional parameters of a bigger model. Hence, we consider byt5-small for the remaining experiments.

reread this part to understand the results that we got from running the code.

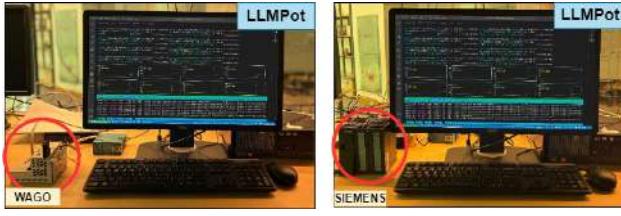


Figure 4: PLCs used in our experiments, WAGO:Modbus[8] (left) and SIEMENS:S7comm[6] (right), being copied by LLMPot.

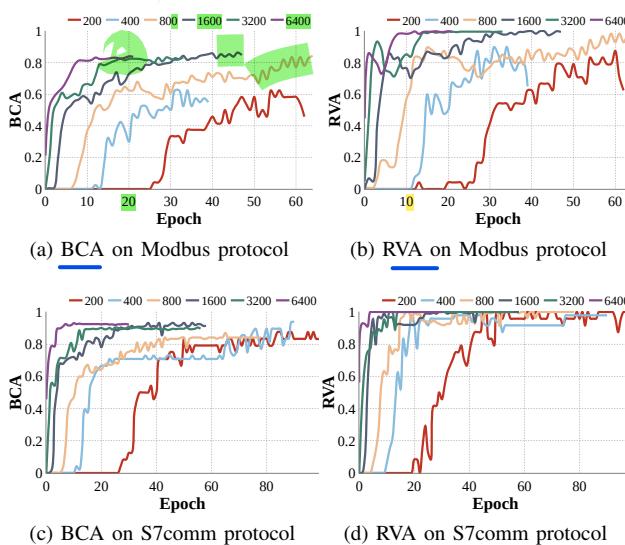


Figure 5: BCA and RVA per epoch of the byt5-small model when using different dataset sizes and protocols to fine-tune. A patience value of 10 epochs was used to stop the fine-tuning in case the validation loss was not improving, thus the different ending epochs for each dataset size.

**5.2.3. Protocol Generalization.** To demonstrate LLMPot’s generalizability in emulating different protocols, we evaluated LLMPot on WAGO and SIEMENS PLC devices (see Figure 4), which use Modbus and S7comm protocols, respectively. Following Algorithm 1, our exploration starts by generating several combinations of dataset sizes as shown in Configuration 5.1 to identify the optimal dataset size and obtain the best performance. Based on the minimum number of samples generated in one iteration by Algorithm 1, which is 144 samples, the minimum size of the fine-tuning dataset is determined irrespective of the protocol and is dictated by the `max_elements` parameter specified in the configuration file. Given that, the minimum meaningful sample size is set to 200 and doubled until no further improvement is observed, as discussed in Section 4.2.2.

Moreover, we focus on a specific set of read and write commands that handle both analog and digital inputs/outputs since these are involved in physical processes. As shown in Configuration 5.1, the covered subset of functions are denoted by their particular numerical FCs (digital read/write: 1, 5, 15, and analog read/write: 3, 6, 16). It is worth pointing out that the LLM fine-tuned only on the functions specified in the configuration file

will respond with an invalid packet if it encounters any other function. Furthermore, the range of the number of permissible inputs and outputs across the allowable range for both Modbus and S7comm protocol was randomly chosen to be 40 for both analog and digital inputs/outputs.

#### Configuration 5.1: Fine-tuning Datasets

Model: byt5-small

Protocols: Modbus, S7comm

Sizes: 200, 400, 800, 1600, 3200, 6400 samples

Split: 80% training, 10% validation, 10% testing

Functions\*: 1, 5, 15, 3, 6, 16

Digital\*: 40

Analog\*: 40

Each dataset essentially corresponds to a separate fine-tuned model.

The trained model with dataset size 1600 is marked as **o1**.

\*Generalizability is tested and proven in Section 5.2.1

During the fine-tuning phase, we evaluate the LLM’s performance using the BCA and RVA metrics at the end of each epoch, using 10% of the fine-tuning dataset. This assessment helps determine how many epochs are necessary for the model to reach its maximum performance, given the dataset size. Figure 5 presents the BCA and the RVA for both protocols for different dataset sizes. We can observe that the RVA improves more rapidly and achieves higher values as the dataset size increases. For smaller datasets, overfitting is evident as depicted by the validation loss surpassing the training loss (see Figure 15a and Figure 15b in Appendix A). Additionally, losses for these smaller datasets remain relatively high, indicating inadequate fine-tuning.

**Takeaway 2:** Byt5-small can effectively learn and respond to network protocols, providing responses with high RVA, and verifying the responses are valid network packets.

**Takeaway 3:** Saturation of the model’s performance with increasing dataset size supports the effectiveness of using the *boundaries* method to create a smaller fine-tuning dataset rather than adopting an exhaustive approach for dataset generation.

**5.2.4. Experimental Evaluation.** We also experimented with different numbers of permissible inputs and outputs across the allowable ranges for both the Modbus and S7comm protocols, testing values from smaller sets (e.g., 40, 100) to larger ones (e.g., 5000, 10000), refer to Table 3. This aimed to demonstrate the efficiency of protocol emulation across varying input/output sizes by using 1600 samples as the dataset size, based on the findings from Section 5.2.2.

Based on that, we demonstrate LLMPot’s generalizability with two datasets, **g1** and **g2**, each with 1600 samples but differing in the number of active addresses for inputs/outputs, 100 and 5000 respectively. Additionally, we consider two more distinct emulation datasets: **d1**, having the same PLC configurations as the fine-tuning datasets, and **d2**, which utilizes different configurations (10000 instead of 40 inputs/outputs).

**Address Range Generalization.** We use models **o1**, **g1** and **g2** to demonstrate the generalizability of LLMPot across varying numbers of active addresses for digital and analog inputs/outputs in PLC setups. Figure 6a and

Stopper  
Here

TABLE 3: Additional datasets used to fine-tune ByT5 model and run experimental evaluations. Each dataset essentially represents a fine-tuned model.

Characteristics	Generalization		Emulation	
	g1	g2	d1	d2
Sizes	1600			
Functions	1-5-15-3-6-16			
Number of Digital	100	5000	40	10000
Number of Analog	100	5000	40	10000

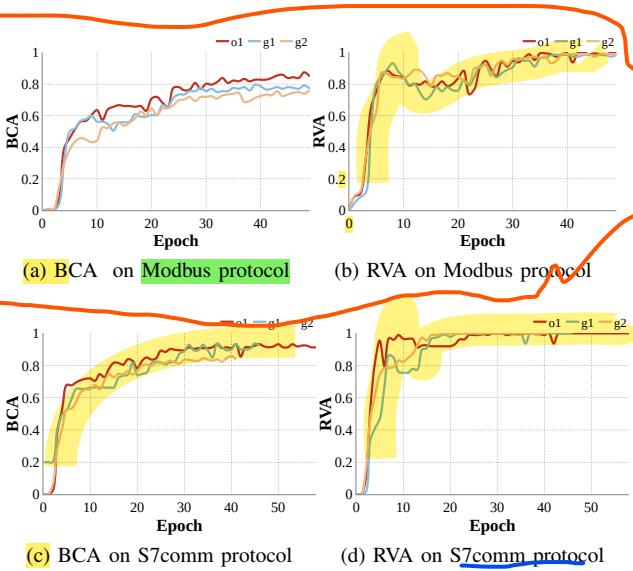


Figure 6: BCA and RVA per epoch of the byt5-small model when using different configurations of analog and digital inputs/outputs o1, g1, g2.

Figure 6c present the BCA and Figure 6b and Figure 6d present the RVA metrics for different combinations of analog and digital inputs/outputs for both Modbus and S7comm protocols. We can observe that all the plots share similar characteristics with almost identical final values. The BCA values over the training epochs converge closely, highlighting the stochastic nature of LLM training. Whereas the RVA results present similar outcomes but with slight improvement due to the metric's characteristics. The dataset generation here is balanced and not biased toward any particular function to ensure fair representation.

**Takeaway 4:** Regardless of the PLC configuration in terms of address availability for digital and analog inputs/outputs, the model's performance remains consistent when the same dataset size is used for fine-tuning.

**PLC profiling given different digital/analog configurations.** Given datasets d1 and d2 shown in Table 3, we fine-tuned byt5-small, essentially profiling two PLCs with different digital/analog configurations. *Do these two models properly reply to corner case requests?*

Table 4 depicts the corner case requests, the ground truth results for both PLC configurations d1 and d2, the response from the model fine-tuned based on Configuration 5.1, and the corresponding BCA and RVA metrics result. The model's responses in bold mean a correct

TABLE 4: Example request-response set for the different emulation scenarios, the model's responses and the validation using both BCA and RVA.

Request	Ground Truth		Model Response	BCA		RVA	
	d1	d2		d1	d2	d1	d2
read(addr=41)	exc	val	exc	✓	✗	✓	✓
read(addr=41)	exc	val	val	✗	✓	✗	✗
read(addr=10001)	exc	exc	exc	✓	✓	✓	✓
read(addr=10001)	exc	exc	val	✗	✗	✗	✗
read(addr=35)	val	val	exe	✗	✗	✗	✗
read(addr=35)	val	val	val	✓	✓	✓	✓

response and striked-out ones mean an unexpected or erroneous response. Next columns mark if the reply of the model is correct (✓) or wrong (✗) with respect to BCA and RVA metrics.

As an example, in the second line of Table 4, given the request to read address 41, the reply in d1 dataset is an exception since the valid address range was up to 40, while in dataset d2 includes a reply with a value of this address because the valid range is up to 10,000. Considering that we use d1 as a dataset for the validator, BCA will mark this as a wrong answer but correct if we use d2. It is worth mentioning here that BCA is just the exact match of the model's reply against the one in the dataset. For RVA, in both cases, the reply is wrong, because the validator checks, given the requested address what should have been the correct reply, considering the PLC configuration that was used to train the model.

Finally, we analyze the emulation datasets for different fine-tuning dataset sizes on both Modbus and S7comm protocols and demonstrate the results in Figure 7a and Figure 7c for BCA and Figure 7b and Figure 7d for RVA. The most important outcome of the emulation is depicted in Figure 7a and Figure 7c, where the BCA is pretty low for different PLC configurations. In contrast, the RVA is high since the validator incorporates logic that compares against the PLC configuration used when training the model. The performance across various configurations being unaffected by the size of the training dataset demonstrates that our results are independent of dataset size variety.

**Takeaway 5:** The model “cloned” the PLC with a specific configuration and behaves in the same manner and not in a random way in terms of responses.

**Varying supported function codes.** Another experiment consists of two sets of various FCs, illustrated in Figure 8 using red and cyan colors. The red set includes Read Discrete Coils (FC 1), Write Single Coil (FC 5), Read Holding Registers (FC 3), and Write Single Register (FC 6). The cyan set is composed of Device Identification (FC 43), Read Discrete Coils (FC 1), Write Multiple Coils (FC 15), Read Holding Registers (FC 3), and Write Multiple Registers (FC 16). For BCA, both sets yield similar final values, whereas for RVA, both achieve maximum values. Additionally, it is observed that convergence for the cyan set is slower, which is anticipated due to the set containing requests that involve multiple elements of digital or analog input(s)/output(s) (FC 15 and FC 16). This complexity

why they  
are  
duplicate  
?  
maybe to  
show  
what  
should b  
the  
correct  
answer in  
each cas  
and wha  
is the  
wrong  
answer.

BCA on different is bad, while in same is s  
RVA is good in both cases.

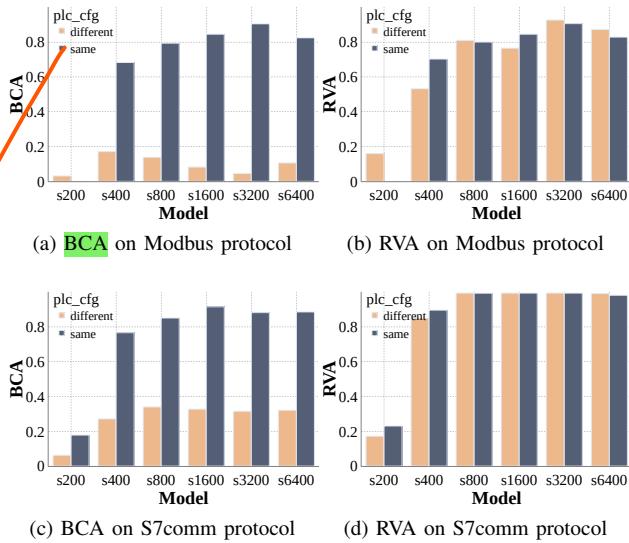


Figure 7: Best BCA and RVA of the byt5-small model when using different dataset sizes to fine-tune for same and different configuration compared to the one used to fine-tune the model for generating the test dataset.

why have they gathered these functions together ?

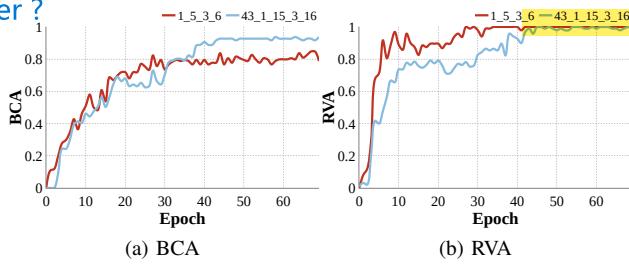


Figure 8: BCA and RVA for different set of functions for Modbus protocol.

contrasts with the red set, which comprises requests with a single element of digital or analog input(s)/output(s) and one additional function (FC 43).

**Takeaway 6:** LLMPot cloning is function independent and can achieve optimum performance in any set of functions.

Overall, these results provide strong evidence related to Research Question (RQ1) in Section 3.1, proving that LLMPot effectively emulates the behavior of industrial network communication protocols with high realism.

**Takeaway 7:** The successful emulation of both Modbus and S7comm protocols by any ByT5 model size demonstrates LLMPot's ability to generalize to different ICS protocols and PLC configurations using the proposed dataset generation method.

stopped here :) 18.1'

## 6. LLMPot Physical Process Emulation

In an ICS framework, control logic reflects the real-time interaction between devices and components involved in a physical process. Mathematical functions are the basic components of a physical process. Thus, exhibiting the LLM's ability to understand and replicate these functions is crucial for accurately mimicking physical processes.

### 6.1. Exploration of ByT5 Approximating Control Functions

Recent work is exploring the ability of LLMs to learn math, with varying degrees of success [9], [61]. In this work, we explore ByT5's capability to approximate functions that typically appear in control logic, in the context of deceiving attackers. Thus, our aim is not to fully replicate math functions, rendering our scope much more limited than works focusing on LLMs learning math. Such efforts are orthogonal to our work, and can be incorporated if accurate math emulation is required.

To explore ByT5 model's ability to approximate various mathematical functions, we utilized the Open Source Community for Automation Technology (OSCAT) library [4], as it provides a wide range of functionalities that are used for industrial automation projects. This library was accessed using the CODESYS [2] platform running on a WAGO PFC200 PLC (shown in Figure 4). The mathematical functions included in this study were classified into discrete methods, involving conditional statements (signum function "sgn"), and continuous methods, subdivided into exponential (exponential base 10 "expo10" and hyperbolic cosine "cosh"), and bounded ("sigmoid" and "cauchy") categories.

When trying to "clone" the behavior of a PLC, it may not be possible to know apriori the mathematical function it uses. A practical approach would be to sample the PLC's responses across a range of uniformly spaced  $x$ -values to create a dataset. This dataset can then be used to train an LLM to approximate the PLC's function. However, uniform sampling may not effectively capture regions where the function changes more rapidly. By focusing on the regions where the function's slope varies significantly, we can optimize our sampling strategy for generating a dataset representing the function's characteristics more accurately. The optimized dataset helps estimate a probability density function (PDF), highlighting areas where denser sampling is required to capture the function's rapid changes accurately.

For example, flat regions of a function might be well represented by just a few points, whereas more complex curves could require many more, ideally an infinite number. This concept is illustrated with *sigmoid* and *cauchy* functions, as shown in Figure 9. In the figure, a histogram illustrates the distribution of sampled points along the  $x$ -axis, with yellow representing points sampled using the PDF mentioned above and gray indicating a discrete uniform distribution. We can observe that the PDF sampling is denser in areas where the functions change rapidly with  $x$  and sparser where the function values are more stable. Algorithm 2 outlines the strategy for generating this weighted dataset. The main idea is to create a distribution of points on the  $x$ -axis that best represents the  $y$ -axis values of a mathematical function.

This algorithm requires uniformly distributed points as a base for sampling along the  $x$ -axis to facilitate the application of the Inverse Transform Sampling (ITS) method [47]. It starts by calculating the derivative of the target function which is numerically estimated based on the responses of the PLC for an unknown function [line: 9], which is then raised to a selected *power* between zero and one to soften its values, thus reducing the amount

the orange sampling shows that they take much more samples in the locations where the function changes rapidly.

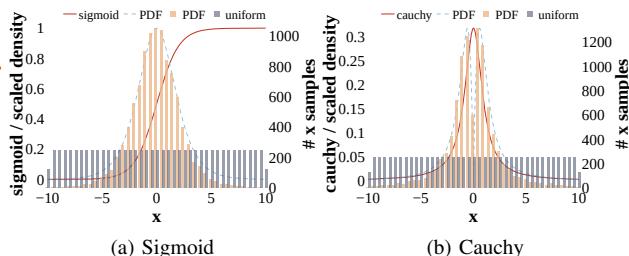


Figure 9: Visual examples of the sampling method to sigmoid and cauchy functions. Plotted are the corresponding functions along with the derivative and  $x$  axis bins.

**Algorithm 2** Algorithm to generate a weighted distribution of  $x$ -axis points for the math function explored.

**Require:**

```

1: mix_ratio: parameter that dictates the weighting between the
   uniform and derivative-based probability density in the mixed PDF.
2: power: value between (0, 1] that softens high values of the df.
3: df: derivative of the math function studies, returns a set of x, y.
4: ct: function that returns the cumulative integral of y along x.
5: inverse_cdf: function that returns values of x corresponding to
   the cumulative density values using interpolator.
6:
7: function X_AXIS_VALUES_DISTRIBUTED(x_values)
8:   samples  $\leftarrow$  LEN(x_values)
9:   dv  $\leftarrow$  DF(x_values, samples)
10:  dv  $\leftarrow$  POWER(dv, power)
11:  diff  $\leftarrow$  dv[0]  $-$  dv[1]
12:  pdf  $\leftarrow$   $\frac{dv}{\text{sum}(dv \cdot diff)}$ 
13:  uni  $\leftarrow$  ONES(samples) / samples
14:  adj_pdf  $\leftarrow$   $(1 - mix\_ratio) \cdot dv + mix\_ratio \cdot uni$ 
15:  adj_pdf  $\leftarrow$   $\frac{adj\_pdf}{\text{sum}(adj\_pdf \cdot diff)}$ 
16:  cdf  $\leftarrow$  CT(adj_pdf, x_values)
17:  cdf  $\leftarrow$  cdf / cdf[-1]
18:  uni  $\leftarrow$  RANDOM(samples)
19:  inv_cdf  $\leftarrow$  INTERP_1D(cdf, x_values, 'linear')
20:  x  $\leftarrow$  inv_cdf(uni)
21:  x  $\leftarrow$  SORT(x)
22:  return x
23: end function

```

of points incorporated in regions of rapid change. The derivative is normalized [line: 12] using the difference between two uniformly distributed points [line: 11], resulting in a PDF. To avoid zeros when the function flattens and ensure a comprehensive exploration of the input space, a mixed PDF is constructed by combining the uniformly distributed PDF with the derivative-based PDF, controlled by the *mix\_ratio* [line: 14]. This ratio adjusts the contribution of each PDF to the mix, focusing on critical regions while covering the entire range. The resulting adjusted PDF is normalized [line: 15] by dividing it by the sum of the product of the mixed PDF and the interval between successive  $x$ -values [line: 11], ensuring the area under the curve equals one and forms a valid probability distribution. To derive the adjusted  $x$ -axis data points from the adjusted PDF, we first calculate the cumulative distribution function (CDF) by integrating the PDF across uniformly distributed  $x$ -values using the trapezoidal rule [line: 16] [14]. This step forms the basis for generating the inverse CDF through linear interpolation [line: 19], using the initial distribution of  $x$ -values. The inverse CDF then interacts with a uniformly distributed set of values to produce the adjusted  $x$ -axis data points, which are subsequently sorted [line: 21] for the sake of clarity.

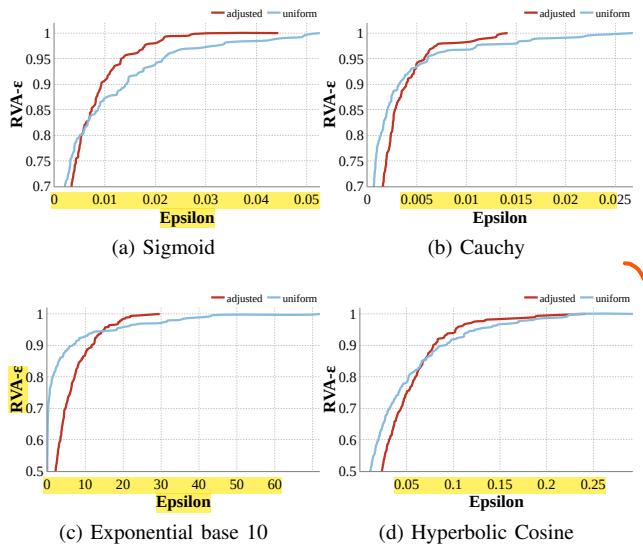


Figure 10: Context, RVA- $\epsilon$  for math functions. Red shows the metric using Algorithm 2 distribution; cyan shows the result with the uniform discrete distribution.

The output of Algorithm 2 is used in an iterative loop to probe the PLC with the sampled  $x$ -axis values to generate necessary packets for model fine-tuning. This loop is much simpler than the one used for protocol emulation in Section 5. It consists of a write and then a read on the analog input/output for each sample collected by the algorithm and controlled by the range that communication protocols (e.g., Modbus, S7comm, etc.) can support. The amount of collected data follows the same method discussed in Section 5.

**Results.** Figure 10 depicts the results obtained from experimenting with the capability of the LLM model to emulate math functions. The  $x$ -axis and  $y$ -axis represent the epsilon value and the cumulative RVA- $\epsilon$ , respectively. Starting with Figure 10a for *sigmoid* and Figure 10b for *cauchy*, Algorithm 2 manages to create an advantage over the uniform sampling method by increasing RVA- $\epsilon$  metric faster and providing full accuracy in smaller epsilon. This effect is seen to be bigger in exponential functions, like *exp10* in Figure 10c and *cosh* in Figure 10d. For these functions, the standard deviation is significantly smaller for the adjusted distribution, 5.1 for *exp10* and 0.036 for *cosh*, while it is 8.23 and 0.045 respectively when using a uniform distribution. This is depicted by the fact that RVA- $\epsilon$  curve reaches its maximum sooner. On the other hand, the *sgn* outputs -1, 0, or 1 depending on whether the input is negative, zero, or positive respectively. Due to the simplicity of the function's conditional nature, the derivative used to calculate PDF will always be zero. This allows LLMPot to quickly capture the pattern and emulate the function accordingly with BCA and RVA of 100%.

**Takeaway 8:** A weighted approach outperforms uniform sampling for LLM-based math function approximation. LLMPot excels with simple and bounded functions, while unbounded ones remain challenging—though acceptable given the target honeypot application since PLC variables are inherently constrained.

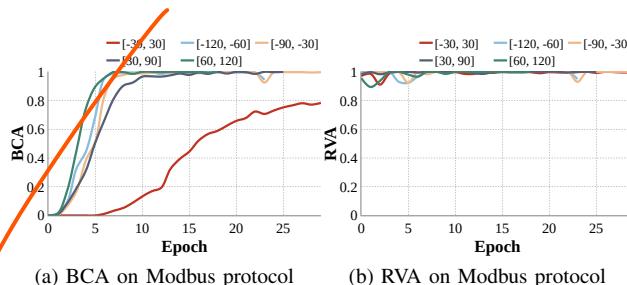


Figure 11: BCA and RVA per epoch of byt5-small model for different probe value ranges for the ICSPatch anaerobic physical process using contextual prompt.

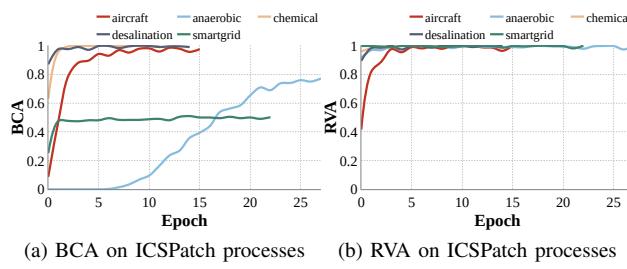


Figure 12: BCA and RVA per epoch of byt5-small model emulating ICSPatch processes using contextual prompt.

## 6.2. ByT5 Emulation of Physical Processes

Building on the previous analysis, we extend our exploration to evaluate ByT5's capabilities in emulating various control programs. We apply this evaluation to five industrial processes from ICSPatch [45]: *Aircraft Flight Control*, *Anaerobic Digestion Reactor*, *Chemical Plant*, *Desalination Plant*, and *Smart Grid*. These systems were accessed through CODESYS on a WAGO PFC200 PLC using the Modbus communication protocol.

The dataset generation for control logic processes follows Algorithm 1 by using read and write commands towards the PLC to capture the inherent functionality of the physical process. The values for the write commands are derived from evenly distributed values within a defined interval. This was initially random but later adjusted incrementally to ensure consistent model performance across different intervals.

**Results.** Figure 11 depicts LLMPot's ability to effectively emulate a physical process regardless of the value ranges. Depending on the value range selected, the physical process has different behavior and this is depicted in the range [-30, 30]. The fluctuation of the returned value is more rapid and thus the BCA is not reaching 100% as the other ranges do. Furthermore, Figure 12 represents the capability of the byt5-small model to emulate control logic processes from the ICSPatch [45] case study. The results of BCA and RVA shown in Figure 12a and Figure 12b, respectively, prove the ability of LLMPot to capture the states of the variables across multiple processes. However, it can be noted that the emulation for the anaerobic (value range [-30,30]) and smart grid processes includes some complex functionality and large input and output numbers, which explains the reduction in performance as depicted in Figure 12a. This requires additional ablation studies to

improve the dataset generation to capture the variables' states accurately.

**Takeaway 9:** Physical processes can achieve high BCA and RVA similarly to protocol emulation in Section 5 proving that context prompt does not impact the performance of the model.

**Takeaway 10:** Consistent with the math function exploration, LLMPot can capture effectively processes with limited input/output range. Capturing processes with more complex functionality require deeper knowledge of the process to properly guide the dataset generation strategy.

## 7. Case Study: Desalination Plant Testbed stopped here

To understand the efficiency of LLMPot in emulating real-world industrial processes, we experiment with a Multi-Stage Flash (MSF) desalination plant testbed [46]. This process consists of various components such as flow rate, temperature and pressure sensors, valves, and PID controllers. In the setup, MATLAB Simulink simulates the desalination process, while the connected PLCs, in Hardware-In-The-Loop (HIL) setup, manage part of the physical process by receiving the Initial Brine Temperature (TB0) and the Distillate Product Flow Rate (Wd) as inputs to regulate the Steam Flow Rate (Ws) using a cascaded PID controller setup [10].

In the default setup, the PID set-point value is set to 93, directly affecting the steam flow rate and producing a steady-state value. During data collection, the set-point value, the timestamp, and the steam flow rate value were integrated into the dataset's context and used as input to the LLMPot fine-tuning process. For testing, minimal variations were randomly added to timestamps to avoid overlapping with the training set, and random set-point values were used at different timestamps to monitor LLMPot emulation results.

Through experimentation, it was noted that set-point values above 85, closer to 93, showed minimal to no fluctuation in Ws. Therefore, we decided not to experiment with larger set-point values. Furthermore, we observed that with abnormally low set-point values, below 70, LLMPot starts to fail. The high number of fluctuating points affected by the low set-point values make it challenging for LLMPot to capture the disturbance fluctuations accurately and thus resulted in the model averaging the data points and losing the Ws steady-state value (refer to Figure 16 in Appendix B). Thus, a trade-off must be made between fine-tuning LLMPot for all possible set-point values and maintaining the Ws steady-state value. Since the ultimate objective is to emulate the physical process and to accurately reflect the values, four set-point values with intervals of 5 points (85, 80, 75, and 70) were chosen for conciseness.

Figure 13 shows the distribution of the data points against the time axis taken from the testbed and the emulated results overlapped, where at each time interval a random set-point value was sent to LLMPot. Figure 13a and Figure 13b show results with two different dataset sizes, 1600 and 3200 used for training LLMPot, respectively. It is worth noting that with a dataset size below 1600, LLMPot fails to predict points beyond the Ws steady-

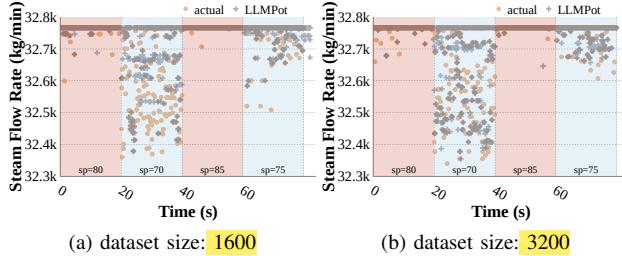


Figure 13: Steam Flow Rate ( $W_s$ ) fluctuations over time during set-point changes across trials.

state value, as it does not include enough fluctuations for LLMPot to be trained properly. Conversely, with a larger dataset size (3200 samples), more disturbance points were included, providing a greater advantage in fine-tuning LLMPot. Furthermore, it can be seen that the majority of the data points from the testbed and LLMPot are clustered around the  $W_s$  steady-state value. Results depict that for different set-points, LLMPot is capable of learning the range and general distribution and emulating the disturbance fluctuation of a real testbed within a normal operational range.<sup>1</sup>

To statistically prove this, we measured the similarity and calculated the p-value for both real and emulated distributions using the Kolmogorov-Smirnov test (KS-test) [20]. The experiment was repeated 10 times and results show p-values above 0.05 indicating no significant difference between both distributions (refer to Table 6 in Appendix B).

**Takeaway 11:** LLMPot can provide suitable steady-state responses over time, effectively capturing the effect of disturbances in a real testbed and providing the output fluctuations an attacker could be observing. Nonetheless, Section 6.1 also proves that the  $x$ -axis can potentially be the time dimension in a physical process.

## 8. Comparison to Related Work

Honeypots are typically categorized as either *low-interactive* or *high-interactive*, based on their realism and interaction levels.

**Low-interactive honeypots** This type simulates essential limited services and are cost-effective and relatively easy to deploy. CryPLH [15] explicitly emulates the S7comm protocol using a manually implemented Python script by querying a specific PLC with a particular configuration and integrating these values into the script. Unfortunately, the source code for this project is not available for further investigation. Furthermore, SHaPe [30] focuses on emulating the IEC 61850 MMS protocol using a library libiec61850 and a framework called Dionaea, however, this project has been discontinued. Gaspot [57] implements Guardian AST protocol using Python scripts that reply to specific requests for a use-case scenario (gas station) implemented for the project. Moreover, Conpot [29] supports emulating multiple protocols using Python scripting. For Modbus, the implementation supports FCs 1,2,3,4,5,6,15,16,17,43 by simulating the PLC's

1. Additional results on different static set-point values appear in Figure 17 in Appendix B.

TABLE 5: ICS honeypot comparison.

Interactivity	Honeypot	Protocol Emulation Base	Emulation			Extensibility			Open Source
			Protocol	Ctrl Logic	Accuracy	PLC	Protocol	Ctrl Logic	
low	CryPLH [15]	CryPLH	Single	-	100	○	○	○	✓
	SHaPe [30]	SHaPe	Single	-	100	●	○	○	✗
	Gspot [57]	Gspot	Single	1	100	○	○	○	✗
	Conpot [29]	Conpot	Multi	-	100	●	●	○	✓
high	DiPot [18]	Conpot [29]	Multi	-	100	●	○	○	✗
	S7commTrace [59]	Conpot [29]	Single	-	100	●	○	○	✗
	<b>HoneyPLC</b> [33]	<b>snap7</b> [7]	Single	-	100	●	○	○	✓
	NeuralPot [49]	Conpot [29]	Single	-	100	●	○	○	✗
	Antonioli et al. [11]	MiniCPS [12]	Single	1	100	●	○	○	✓
	HoneyPhy [32]	HoneyPhy [32]	Single	1	100	●	○	○	✗
	ICSpot [19]	HoneyPLC [33]	Multi	1	100	●	○	○	✓
	<b>LLMPot</b>	<b>LLMPot</b>	<b>Multi</b>	<b>6+</b>	<b>99.30</b>	<b>●</b>	<b>●</b>	<b>●</b>	<b>✓</b>

●: Automated (modification of existing configuration),  
 ○: Semi-automated (guided manual effort needed),  
 ○: Not Extensible, - (dash): Not supported,  
 Accuracy: Packet Validity Percentage (RVA)

memory so their replies always comply with how a PLC device would work. This work supports XML-based configuration but still lacks the ability to modify the behavior of the honeypot for specific scenarios, that would need additional Python coding. Overall, the simplicity and limited functionality of the aforementioned *low-interactive honeypots* lead the attackers to easily identify them as decoys [33].

**High-interactive honeypots** On the other hand, this type provides a more complex and authentic environment that simulates real operating systems and their services, making them ideal for detailed data collection on attackers' methodologies. Key examples of high-interactive honeypots include DiPot [18], a distributed system that uses Conpot [29] to simulate industrial protocols. S7commTrace [59] claims superiority over Conpot by extending the S7comm protocol with additional FCs. However, the source code for both DiPot and S7commTrace is not available. Furthermore, HoneyPLC [33] enhances functionality by simulating the storage of ladder logic on PLCs and supports out-of-the-box PLCs such as Siemens S7-300, Allen-Bradley MicroLogix 1100, ABB PM554-TP-ETH PLCs, etc. Their implementation utilizes the Snap7 library which was modified (following a combination of design patterns [34] like proxy or delegator could be a more appropriate way instead of modifying it) and recompiled to suit the updated functionality, making the honeypot very difficult to configure for different PLCs with different analog and digital inputs/outputs. NeuralPot [49] employs a Generative Adversarial Network [25] and Auto-Encoders [26] to learn the behavior of the ICS device and dynamically generate Modbus traffic. They also utilize Conpot for their protocol emulation, however no source code is available. Overall, the aforementioned high-interactive honeypots lack the essential aspect of ensuring high fidelity by emulating a physical process.

Antonioli et al. [11] addressed this by designing a virtual high-interactive honeypot that mimics a water treatment process. However, developing a testbed that accurately simulates real-world conditions or integrates various technologies is challenging and highly complex. On top of that, building testbeds is costly and requires regular maintenance and technical support to keep the testbed operational. HoneyPhy [32] focuses on creating

realistic cyber-physical system (CPS) simulations using the Distributed Network Protocol (DNP3) protocol [21]. However, this work relies on constructing a white box model, which requires detailed knowledge of the ICS device, also no source code is available. ICSPot [19] is built on top of HoneyPLC [33], enhancing the emulation of network communications and physical-layer interactions using Industrial Hacking Simulator [31]. It utilizes a simplified simulation of the water treatment process provided by MiniCPS [12]. The authors in [12] use the PyModbus Python library to emulate Modbus protocol by implementing the server side, customized with if-else statements. On top of that, they implement two distinct physical processes, however, the effort needed to implement these is proportional to the complexity of the processes itself. Overall, the aforementioned related works support only a single predetermined physical process.

Table 5 illustrates a qualitative comparison of the aforementioned state-of-the-art honeypots with our proposed LLMPot. Interactivity column is an abstractly defined term in the literature and mostly adheres to the amount of time and data exchanged between a client/attacker and the honeypot. Each honeypot usually depends on a base software that performs the needed initialization and communications to emulate a custom PLC. The protocol emulation column is labeled as either “Single” or “Multi” based on whether the honeypot emulates one or more protocols. The control logic emulation column defines how many physical processes are emulated if any. Accuracy column reports the validity of the generated packet by the honeypot using the RVA metric. LLMPot demonstrates highly comparable accuracy to the existing state-of-the-art honeypots. The 99.30% is derived from Table 2 where the byt5-small model was trained using the 1600 samples dataset. Finally, the extensibility section qualitatively describes the ability of the honeypot to expand to additional PLC devices, protocols, and control logic. The PLC “Extensibility” term, defined in [33], allows effective PLC hardware emulation of different models and vendors.

For both protocol and control logic extensibility, LLMPot manages to excel compared to other honeypots. It provides the user the ability to extend the functionality by following the methodology described in Algorithm 1 for protocols, and by probing physical processes as described in Section 6 for the overall goal of replicating the user chosen configuration for the honeypot.

**Takeaway 12:** LLMPot becomes more dynamic and adaptable to various PLC configurations, moving away from the traditional reliance on static and manual scripting specific to PLC setups. Two distinct protocols and more than six physical processes were emulated as a case study.

## 9. Honeypot Development and Analysis

### 9.1. Honeypot Setup

LLMPot uses Nmap [39] and wget [24] to automatically gather essential parameters such as MAC addresses and operating systems from the target PLC device. We emulate the honeypot’s TCP/IP stack using Honeyd [44], configuring it with the appropriate Nmap fingerprint and

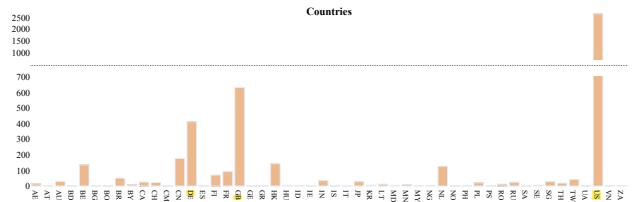


Figure 14: Geolocation of Honeypot Interaction

integrating real services via Docker containers [22]. These containers host both the website and the protocol emulation. The system operates on an Ubuntu 22.04 desktop setup with Honeyd and Docker, and the LLM runs on an RTX-3090 Ti. Network traffic is managed by a router that directs specific traffic to the host machine or the virtual hosts managed by Honeyd. Traffic on ports 502, 80, and 443 are routed to the host machine, where HAProxy [3] then redirects web traffic to ensure secure communication, directing HTTP to HTTPS and handling port-specific traffic.

The replication of the PLC website involves manually downloading the web server files. A Python Flask application mimics the site’s functionality, including a REST API, to provide dynamic web content [40]. We manually generate the web server’s SSL certificate using the OpenSSL tool [51]. For data logging and analysis, we employ MongoDB [37] with the Beanie ODM [13] for robust data management. The system logs detailed information about each interaction, such as IP addresses, web pages visited, and the timing of requests and responses. It also captures login credentials attempted during brute force attacks and records incoming and outgoing Modbus server traffic.

### 9.2. Interaction Analysis

We exposed the developed LLMPot honeypot to the internet to monitor network activities. Over two months, the honeypot recorded 4895 connection attempts from unique IP addresses. Figure 14 indicates that most of these attempts were from the United States (1399 interactions). Great Britain and Germany are the second most common sources with 632 and 412 attempts, respectively. The interaction analysis further revealed a total of 7010 requests sent to LLMPot, where 1891 of these were valid Modbus requests. Among the valid requests, 1752 were identified to use specific FCs: 1, 3, 5, 6, 15, 16, 17, 43. To be specific, 1629 requests focused only on acquiring the device information (FC 43). Overall, LLMPot managed to produce 1496 valid Modbus responses.

**Takeaway 13:** LLMPot’s capability to handle and respond to a variety of Modbus FCs from the internet demonstrates its effectiveness in emulating a honeypot and shows that LLMs can be used for ICS threat detection and analysis.

We further utilized the AbuseIPDB [1] database to identify the IP addresses that have been associated with malicious activity online by using a threshold value of 50 in the given scale of abuse confidence score (0 to 100). Among the total requests targeting our honeypot, 731 distinct IPs interacted more than once using the Modbus protocol, where 441 IPs were categorized as malicious.

As part of our analysis, it was also noticed that 164 code injection attempts (SQL, LDAP, etc.) were made using the “HTTP\_X\_FORWARDED\_FOR” http protocol header or the username/password fields in the PLC login web page.

**Takeaway 14:** The high interactivity of attackers further proves the realism of the developed LLMPot in emulating a real ICS device.

### 9.3. Resilience to Reconnaissance Tools

To evaluate the effectiveness of the honeypot in emulating real network devices, it was tested using various network reconnaissance tools. Nmap testing revealed that the honeypot effectively camouflages as a legitimate device; for instance, Nmap shows confidence of 95% for WAGO’s PLC operating system, while LLMPot obtained an 87% confidence rating. Further testing was done using Shodan [48], which evaluates devices based on characteristics such as the number of open ports and specific network configurations to determine if a device is potentially a honeypot. Our deployed honeypot was successfully disguised and not labeled as a honeypot by Shodan.

### 9.4. Real-time Responsiveness

The LLM response needs to be generated in a time comparable to the real PLC response time. LLMPot produces a valid response in 160ms using an RTX3090Ti GPU. However, we observed lower response times, around 110ms to 140ms, when serving multiple requests in parallel using an A100 GPU. This is comparable to the reported times of real PLCs in Honeyvp [62], which are approximately 100ms. The small difference between the two is expected to be masked by the much longer delays of the full stack network communication over public internet. LLMPot achieved a maximum performance of 30 requests/sec using one Nvidia A100 GPU with a multi-instance setup when simulating a variety of environments from the literature (ICSPatch [45]), such as a desalination process, chemical process, airplane controller etc. that simulate complex real-world scenarios. LLMPot’s response time was consistent in all processes, implying that performance does not directly depend on the complexity of the process, but rather on how LLMPot is structured (number of parameters) and the size of the reply defined by the protocol. The performance of LLMPot scales linearly with the number of GPUs: X Nvidia A100 GPUs can serve 30\*X requests/second.

## 10. Discussion and Limitations

**ByT5 Architecture and Resources:** LLMPot uses the ByT5-small model from ByT5 series, which includes various sizes to balance computational complexity and capacity to handle complex patterns. The sizes range from 300M parameters for ByT5-small to 12.9B parameters for ByT5-XXL. Although ByT5-base with 528M performed similarly to ByT5-small in our tests, we opted for ByT5-small due to its lower computational demands.

**Fine-tuning an LLM to Emulate Multiple Protocols** LLMPot uses separate ByT5 instances to emulate Modbus and S7comm protocols. Our initial experiments with one

model emulating multiple protocols have shown promising results and this will be part of our future research.

**Emulating Unbounded Functions** Another challenge encountered in exploring mathematical functions is related to exponential functions, which typically grow indefinitely. This characteristic poses significant difficulties in predicting accurate results, especially as function values become excessively large. However, in physical processes, variables are naturally constrained and do not extend infinitely. **Protocol Configuration File** LLMPot depends on a configuration file that includes data fields specific to various industrial protocols. This can be potentially automated in future work. However, this is a one-time effort per industrial protocol, and the number of industrial protocols is not very large.

**ICS Protocol Coverage** It is essential to highlight that LLMPot currently generates precise response packets for requests that involve functions within its training scope (FCs 1,2,3,4,5,6,15,16,17,43). However, the results may not be as precise for functions outside its training scope. Future work will incorporate automated function / attribute discovery for any protocol, so the configuration file describing all possible functions, features, attributes, and constraints can eventually be generated automatically.

**Real Time Performance Improvement** To minimize the response overhead of LLMPot, the first approach could be to use a more modern model like RWKV [41] which is also not reliant to a tokenizer, while other possible options which use a tokenizer could be Gemma-2 [50] and Phi-2 [28] that promise faster inference. Secondly, pruning can be applied to the trained model to reduce complexity. Finally, a lightweight version of a generative model can be implemented. These approaches could effectively reduce the number of parameters used, thereby minimizing inference time.

## 11. Conclusion

The LLMPot framework is, to the best of our knowledge, the first approach to use LLMs to emulate ICS network protocols and physical processes. It provides a novel platform that researchers can use to mimic network protocols, mathematical equations, and physical processes. This platform can also be used to create a real honeypot. LLMPot demonstrates the LLMs potential and capabilities in industrial protocol emulation, real-time interaction, and physical process capturing. LLMs can understand and emulate industrial protocols, such as Modbus, S7comm, etc. They also exhibit potential in producing credible responses during efforts to emulate some physical process. Future work will explore further automating the ICS protocol configuration file development, as well as further pushing the boundaries of the fidelity of the physical value responses when emulating a process.

## Acknowledgments

This work has been supported by the NYUAD Center for Cyber Security under RRC Grant No. G1104.

## Availability

The source code, models, and datasets are open source at <https://github.com/momalab/LLMPot>.

## References

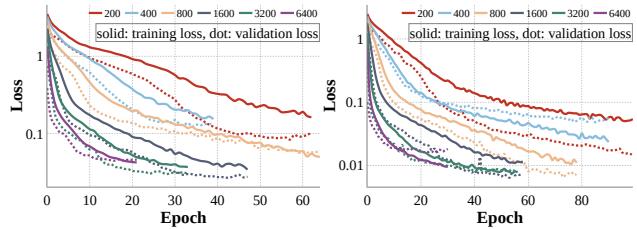
- [1] “AbuseIPDB - IP address abuse reports - making the internet safer, one IP at a time,” <https://www.abuseipdb.com/>.
- [2] “CODESYS Automation Platform,” <https://www.codesys.com/products/codesys-engineering/automation-platform.html>.
- [3] “HAProxy,” <https://www.haproxy.com/>.
- [4] “OSCAT Basic Library Documentation,” [http://www.oscat.de/images/OSCATBasic/oscat\\_basic333\\_en.pdf](http://www.oscat.de/images/OSCATBasic/oscat_basic333_en.pdf).
- [5] “PyTorch Lightning Documentation,” <https://lightning.ai/docs/pytorch/stable/>.
- [6] “Siemens Official Website,” <https://new.siemens.com/global/en.html>.
- [7] “Step7 open source ethernet communication suite,” <https://snap7.sourceforge.net/>, last accessed: Oct 21, 2024.
- [8] “WAGO Official Website,” <https://wago.com/global/>.
- [9] J. Ahn, R. Verma, R. Lou, D. Liu, R. Zhang, and W. Yin, “Large language models for mathematical reasoning: Progresses and challenges,” *arXiv preprint arXiv:2402.00157*, 2024.
- [10] E. Ali, “Understanding the operation of industrial msf plants part i: Stability and steady-state analysis,” *Desalination*, vol. 143, no. 1, pp. 53–72, 2002.
- [11] D. Antonioli, A. Agrawal, and N. O. Tippenhauer, “Towards high-interaction virtual ics honeypots-in-a-box,” in *Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy*, 2016, pp. 13–22.
- [12] D. Antonioli and N. O. Tippenhauer, “Minicps: A toolkit for security research on cps networks,” in *Proceedings of the First ACM workshop on cyber-physical systems-security and/or privacy*, 2015, pp. 91–100.
- [13] Benjamin Allévy, “Beanie - asynchronous python odm (object document mapper) for mongodb,” <https://github.com/roman-right/beanie>, last accessed: April 27, 2024.
- [14] R. L. Burden, J. D. Faires, and A. M. Burden, *Numerical Analysis*, 10th ed. Boston, MA: Cengage Learning, 2016.
- [15] D. I. Buza, F. Juhász, G. Miru, M. Félegyházi, and T. Holczer, “Cryph: Protecting smart energy systems from targeted attacks with a plc honeypot,” in *Smart Grid Security: Second International Workshop, SmartGridSec 2014, Munich, Germany, February 26, 2014, Revised Selected Papers 2*. Springer, 2014, pp. 181–192.
- [16] A. Bytes, P. H. N. Rajput, C. Doumanidis, M. Maniatakos, J. Zhou, and N. O. Tippenhauer, “Fieldfuzz: In situ blackbox fuzzing of proprietary industrial automation runtimes via the network,” in *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, 2023, pp. 499–512.
- [17] R. M. Campbell, K. Padayachee, and T. Masombuka, “A survey of honeypot research: Trends and opportunities,” in *2015 10th international conference for internet technology and secured transactions (ICITST)*. IEEE, 2015, pp. 208–212.
- [18] J. Cao, W. Li, J. Li, and B. Li, “Dipot: A distributed industrial honeypot system,” in *Smart Computing and Communication: Second International Conference, SmartCom 2017, Shenzhen, China, December 10-12, 2017, Proceedings 2*. Springer, 2018, pp. 300–309.
- [19] M. Conti, F. Trolese, and F. Turrin, “Icspot: A high-interaction honeypot for industrial control systems,” in *2022 International Symposium on Networks, Computers and Communications (IS-NCC)*. IEEE, 2022, pp. 1–4.
- [20] W. Daniel, *Applied Nonparametric Statistics*, ser. Duxbury advanced series in statistics and decision sciences. PWS-KENT Pub., 1990. [Online]. Available: <https://books.google.ae/books?id=0hPvAAAAMAAJ>
- [21] DNP Users Group, “Dnp3 protocol specification,” <https://www.dnp.org/About/Overview-of-DNP3-Protocol>, DNP Users Group, Tech. Rep., Year, last accessed: April 27, 2024.
- [22] Docker, Inc., “Docker - build, share, and run any app, anywhere,” <https://www.docker.com/>, last accessed: April 27, 2024.
- [23] EtherCAT Technology Group, “Ethercat technology,” <https://www.ethercat.org/>, accessed: April 27, 2024.
- [24] GNU Project, “Gnu wget - the non-interactive network downloader,” <https://www.gnu.org/software/wget/>, last accessed: April 27, 2024.
- [25] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” *Advances in neural information processing systems*, vol. 27, pp. 2672–2680, 2014.
- [26] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *Science*, vol. 313, no. 5786, pp. 504–507, 2006.
- [27] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, “Lora: Low-rank adaptation of large language models,” *arXiv preprint arXiv:2106.09685*, 2021.
- [28] M. Javaheripi, S. Bubeck, M. Abdin, J. Aneja, S. Bubeck, C. C. T. Mendes, W. Chen, A. Del Giorno, R. Eldan, S. Gopi *et al.*, “Phi2: The surprising power of small language models,” *Microsoft Research Blog*, vol. 1, no. 3, p. 3, 2023.
- [29] A. Jicha, M. Patton, and H. Chen, “Scada honeypots: An in-depth analysis of conpot,” in *2016 IEEE conference on intelligence and security informatics (ISI)*. IEEE, 2016, pp. 196–198.
- [30] K. Koltyś and R. Gajewski, “Shape: A honeypot for electric power substation,” *Journal of telecommunications and information technology*, no. 4, pp. 37–43, 2015.
- [31] C. Lannister, “IHS GitHub Repository,” <https://web.archive.org/web/20220522113148/https://github.com/CarlosLannister/IHS>.
- [32] S. Litchfield, D. Formby, J. Rogers, S. Meliopoulos, and R. Beyah, “Rethinking the honeypot for cyber-physical systems,” *IEEE Internet Computing*, vol. 20, no. 5, pp. 9–17, 2016.
- [33] E. López-Morales, C. Rubio-Medrano, A. Doupé, Y. Shoshitaishvili, R. Wang, T. Bao, and G.-J. Ahn, “Honeyplc: A next-generation honeypot for industrial control systems,” in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 279–291.
- [34] R. C. Martin, “Design principles and design patterns,” *Object Mentor*, vol. 1, no. 34, p. 597, 2000.
- [35] S. McLaughlin, C. Konstantinou, X. Wang, L. Davi, A.-R. Sadeghi, M. Maniatakos, and R. Karri, “The cybersecurity landscape in industrial control systems,” *Proceedings of the IEEE*, vol. 104, no. 5, pp. 1039–1057, 2016.
- [36] Modbus Organization, “Modbus protocol specification,” [https://www.modbus.org/docs/Modbus\\_Application\\_Protocol\\_V1\\_1b3.pdf](https://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf), accessed: April 27, 2024.
- [37] MongoDB, Inc., “Mongodb - the most popular database for modern applications,” <https://www.mongodb.com/>, last accessed: April 27, 2024.
- [38] C. Moore, “Detecting ransomware with honeypot techniques,” in *2016 Cybersecurity and Cyberforensics Conference (CCC)*. IEEE, 2016, pp. 77–81.
- [39] A. Orebaugh and B. Pinkard, *Nmap in the enterprise: your guide to network scanning*. Elsevier, 2011.
- [40] Pallets, “Flask - python web framework,” <https://flask.palletsprojects.com/>, last accessed: April 27, 2024.
- [41] B. Peng, E. Alcaide, Q. Anthony, A. Albalak, S. Arcadinho, S. Biderman, H. Cao, X. Cheng, M. Chung, M. Grella *et al.*, “Rwkv: Reinventing rnns for the transformer era,” *arXiv preprint arXiv:2305.13048*, 2023.
- [42] R. Pigglin and I. Buffey, “Active defence using an operational technology honeypot,” in *11th International Conference on System Safety and Cyber-Security (SSCS 2016)*. IET, 2016, pp. 1–6.
- [43] PROFINET International, “Profinet technology,” <https://www.profibus.com/technology/profinet/>, accessed: April 27, 2024.
- [44] N. Provos, “Honeyd-a virtual honeypot daemon,” in *10th dfn-cert workshop, hamburg, germany*, vol. 2, 2003, p. 4.

- [45] P. H. N. Rajput, C. Doumanidis, and M. Maniatakos, “{ICSPatch}: Automated vulnerability localization and {Non-Intrusive} hot-patching in industrial control systems using data dependence graphs,” in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 6861–6876.
- [46] P. H. N. Rajput, P. Rajput, M. Sazos, and M. Maniatakos, “Process-aware cyberattacks for thermal desalination plants,” in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, 2019, pp. 441–452.
- [47] R. Sheldon, *A First Course in Probability*. Pearson, 2018.
- [48] Shodan, “Shodan - the search engine for internet-connected devices,” <https://www.shodan.io/>, last accessed: April 27, 2024.
- [49] I. Sinosoglou, G. Efstatopoulos, D. Pliatsios, I. D. Moscholios, A. Sarigiannidis, G. Sakellari, G. Loukas, and P. Sarigiannidis, “Neuralpot: An industrial honeypot implementation based on deep neural networks,” in *2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2020, pp. 1–7.
- [50] G. Team, M. Riviere, S. Pathak, P. G. Sessa, C. Hardin, S. Bhupatiraju, L. Husseidot, T. Mesnard, B. Shahriari, A. Ramé *et al.*, “Gemma 2: Improving open language models at a practical size,” *arXiv preprint arXiv:2408.00118*, 2024.
- [51] The OpenSSL Project, “Openssl - the open source toolkit for ssl/tls,” <https://www.openssl.org/>, last accessed: April 27, 2024.
- [52] The Tcpdump Group, “tcpdump - a powerful command-line packet analyzer,” <https://www.tcpdump.org/>, last accessed: April 27, 2024.
- [53] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, “Llama: Open and efficient foundation language models,” *arXiv preprint arXiv:2302.13971*, 2023.
- [54] D. Tychalas, H. Benkraouda, and M. Maniatakos, “[ICSFuzz]: Manipulating {I/Os} and repurposing binary code to enable instrumented fuzzing in {ICS} control applications,” in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2847–2862.
- [55] C. Villa, C. Doumanidis, H. Lamri, P. H. N. Rajput, and M. Maniatakos, “Icsquartz: Scan cycle-aware and vendor-agnostic fuzzing for industrial control systems,” in *Network and Distributed System Security (NDSS) Symposium*, 2025.
- [56] N. Weiler, “Honeypots for distributed denial-of-service attacks,” in *Proceedings. Eleventh IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*. IEEE, 2002, pp. 109–114.
- [57] K. Wilhoit and S. Hilt, “The gaspot experiment: Unexamined perils in using,” *blackhat*, 2015.
- [58] Wireshark, “S7 communication protocol,” <https://wiki.wireshark.org/S7comm>, accessed: April 27, 2024.
- [59] F. Xiao, E. Chen, and Q. Xu, “S7commtrace: A high interactive honeypot for industrial control system based on s7 protocol,” in *Information and Communications Security: 19th International Conference, ICICS 2017, Beijing, China, December 6–8, 2017, Proceedings 19*. Springer, 2018, pp. 412–423.
- [60] L. Xue, A. Barua, N. Constant, R. Al-Rfou, S. Narang, M. Kale, A. Roberts, and C. Raffel, “Byt5: Towards a token-free future with pre-trained byte-to-byte models,” *Transactions of the Association for Computational Linguistics*, vol. 10, pp. 291–306, 2022.
- [61] H. Ying, S. Zhang, L. Li, Z. Zhou, Y. Shao, Z. Fei, Y. Ma, J. Hong, K. Liu, Z. Wang *et al.*, “Internlm-math: Open math large language models toward verifiable reasoning,” *arXiv preprint arXiv:2402.06332*, 2024.
- [62] J. You, S. Lv, Y. Sun, H. Wen, and L. Sun, “Honeyvp: A cost-effective hybrid honeypot architecture for industrial control systems,” in *ICC 2021-IEEE International Conference on Communications*. IEEE, 2021, pp. 1–6.

## Appendix

### 1. Additional results on protocol emulation

In Figure 15a and Figure 15b, show the training and the validation loss plots. For smaller datasets, overfitting



(a) train/validation loss on Modbus (b) train/validation loss on s7comm

Figure 15: The BCA and RVA per epoch of the byt5-small model when using different dataset sizes and protocols to finetune. A patience value of 10 epochs was used to stop the finetuning in case the validation loss was not improving, thus the different ending epochs for each dataset size.

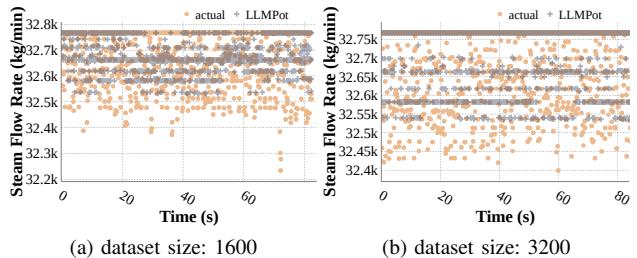


Figure 16: Fluctuation of steam flow rate over time using set-point value 65 and dataset sizes, 1600 and 3200.

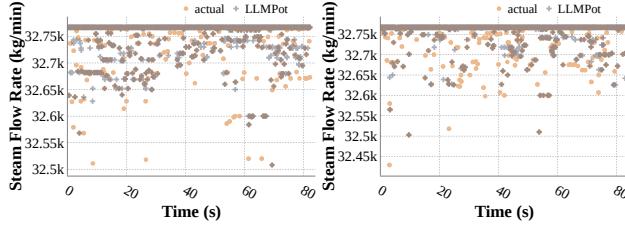
is evident as depicted by the validation loss surpassing the training loss. Additionally, losses for these smaller datasets remain relatively high, indicating inadequate fine-tuning.

### 2. Additional results on industrial testbed emulation

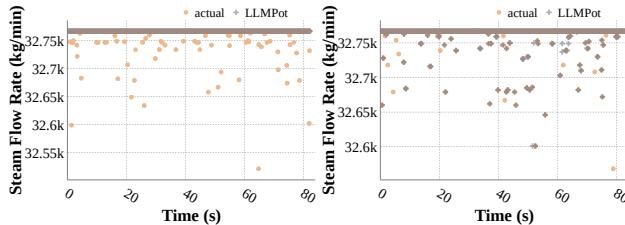
Figure 16 depicts the point where LLMPot fails to capture the disturbance fluctuation due to the vast amount of fluctuating points and instead averages the data points. This case is observed in all set-point values below 70 and both dataset sizes, 1600 and 3200.

Through experimentation, it was noted that with abnormally low set-point values, below 70, LLMPot starts to fail. It was observed that the high number of fluctuating points affected by low set-point values make it challenging for LLMPot to capture the disturbance fluctuations accurately and thus resulted in the model averaging the data points and losing the Ws steady-state value as shown in Figure 16. This figure depicts the point where LLMPot fails to capture the disturbance fluctuation due to the vast amount of fluctuating points. Since the ultimate objective is to emulate the physical process and to accurately reflect the values, a trade-off was made to make LLMPot accurately emulate and maintain the Ws steady-state value.

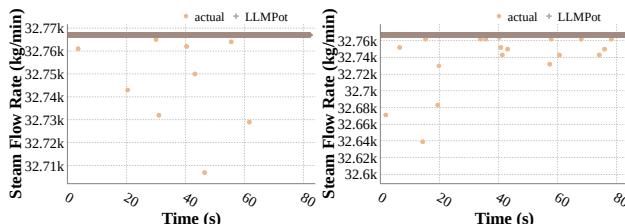
Figure 17 depicts the additional experiment made to capture the distribution of the data points collected from the real testbed and LLMPot’s emulation results. For this experiment, different dataset sizes were chosen, 1600 and 3200. It was observed that in dataset sizes below 1600, LLMPot will start to fail as there are not enough data points in the dataset for the model to learn from. On the



(a) setpoint: 75, dataset size: 1600 (b) setpoint: 75, dataset size: 3200



(c) setpoint: 80, dataset size: 1600 (d) setpoint: 80, dataset size: 3200



(e) setpoint: 85, dataset size: 1600 (f) setpoint: 85, dataset size: 3200

Figure 17: Fluctuation of Steam Flow Rate over Time using different PID controller setpoints.

TABLE 6: Statistical Analysis for Real and Predicted Distribution using Kolmogorov-Smirnov test and presenting the P-value and KS-statistic.

Run	Metric	sp: 75 size: 1600	sp: 75 size: 3200	sp: 80 size: 1600	sp: 80 size: 3200
1st	P	0.33717	0.05792	0.00008	0.99957
	KS	0.03295	0.03306	0.07938	0.00877
2nd	P	0.23519	0.09639	0.00008	1.00000
	KS	0.03619	0.03061	0.07938	0.00593
3rd	P	0.21852	0.00082	0.00008	0.99967
	KS	0.03682	0.05461	0.07938	0.01002
4th	P	0.37873	0.00000	0.00008	1.00000
	KS	0.03182	0.14043	0.07938	0.00687
5th	P	0.38359	0.03322	0.00008	0.99999
	KS	0.03170	0.03561	0.07938	0.00718
6th	P	0.51533	0.07030	0.00008	0.99999
	KS	0.02857	0.03217	0.07938	0.00721
7th	P	0.29466	0.05802	0.00008	1.00000
	KS	0.03420	0.03305	0.07938	0.00687
8th	P	0.62591	0.10248	0.00008	0.99996
	KS	0.02619	0.03029	0.07938	0.00781
9th	P	0.37877	0.10939	0.00008	0.99998
	KS	0.03182	0.02993	0.07938	0.00752
10th	P	0.37877	0.07498	0.00008	0.99990
	KS	0.03182	0.03186	0.07938	0.00815

other hand, with dataset size beyond 3200, the model was facing challenges in capturing the vast amount of fluctuating points. Thus, both 1600 and 3200 sizes were chosen as they provide the best performance of LLMPot. Moreover, various set-point values were chosen in the experiment, 75, 80, and 85, in order to better understand the behavior of the process.

Table 6 depicts the statistical results obtained from

the Kolmogorov-Smirnov test. The table outlines results obtained from 10 runs for different set-point values, 75 and 80, on two different dataset sizes, 1600 and 3200.