iConPAL: LLM-guided Policy Authoring Assistant for Configuring IoT Defenses

Mahbub Alam, Siwei Zhang, Eric Rodriguez, Akib Nafis, and Endadul Hoque
Syracuse University, NY USA

2024 IEEE Secure Development Conference (SecDev) | 979-8-3503-9193-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/SecDev61143.2024.00014

***Abstract*—Safety and security concerns surrounding Internet-of-Things (IoT) platforms for smart homes have spurred the development of defense mechanisms to safeguard against unexpected behaviors in accordance with safety and security policies. However, the need to manually craft policies in tool-specific languages increases the burden on humans. Previous attempts to address this issue have fallen short, either lacking portability or requiring human intervention in other forms. Therefore, in this paper, we propose iConPAL, an automated policy authoring assistant for IoT environments. iConPAL accepts a policy description in natural language (English) and translates it into a specific formal policy language. iConPAL leverages the capabilities of modern large language models (LLMs), employs prompt engineering to automatically generate few-shot learning prompts for the LLM, and post-processes the LLM's response to ensure the validity of the translated policy. We implemented a prototype of iConPAL and evaluated it on our curated dataset of 290 policies. We observed that iConPAL successfully translated 93.61% policies, of which 93.57% were semantically correct. iConPAL's high accuracy makes it suitable for assisting ordinary users in drafting policies for smart homes.**

***Index Terms*—IoT Security, Policy Authoring and Enforcement**

I. Introduction

The rapid expansion of programmable Internet-of-Things (IoT) platforms has enabled affordable automation solutions for smart homes, utilizing a myriad of low-cost IoT devices [1–3]. Concurrently, safety and security concerns surrounding smart homes have taken center stage, prompting the development of defense mechanisms mitigating unexpected/insecure behaviors on these platforms [4–21]. Defenses vary in terms of their core mechanisms—static analysis, dynamic analysis, and runtime monitoring—yet their primary focus remains on safeguarding against unexpected behaviors in accordance with safety and security policies, each defining expected IoT system behavior for installed devices. However, the requirement to manually craft policies in tool-specific languages increases the burden on humans, diminishing the appeal and adoption of these defense tools. Therefore, it is crucial to automate this policy authoring process.

Prior work [9, 11, 13, 16, 17, 22–24] attempts to reduce the human burden by employing automated policy synthesis mechanisms. However, these approaches fall short for several reasons. Firstly, they require either a collection of execution traces (such as event logs or user activities) from the smart home, labeled with acceptable and unacceptable outcomes (*i.e.*, positive and negative examples) [9], or access to the source code of installed automation apps [13, 16, 17]. These requirements impose significant human effort, which may be outright impossible in some cases due to inaccessible source code or a lack of traces. Secondly, the synthesized policies are often limited in scope, as they can only express invariants—a subset of policies [11, 13]. Finally, the policy synthesizers are tightly coupled with their respective defense tools, and the synthesized policies are specific to the current smart home, lacking portability altogether.

On the other hand, prior work [25] explores the concept of training non-tech-savvy users to formulate their desired policies using a formal policy language through an online user study. Participants underwent gradual training through various tutorials and were evaluated on their ability to draft policies in the trained language. The overwhelmingly low success rate suggests that these policy languages are too complex for ordinary users to grasp and utilize effectively for policy composition.

In this paper, we propose iConPAL, an automated policy authoring assistant. iConPAL takes a natural language description of a policy from a user as input and produces the policy translated into an expressive formal policy language tailored for various IoT defense mechanisms. Unlike prior work, iConPAL does not have the same limitations: (a) It does not necessitate execution traces (*e.g.*, event logs) or source code of automation apps; (b) It uses a policy language that is not tied to any specific defense tools but is expressive enough to capture policies beyond invariants (*e.g.*, linear temporal logic); (c) It is highly portable and not dependent on any particular defense tools or smart home setup; and (d) It does not require users to learn the policy language. These features make iConPAL a user-friendly tool for policy creation in IoT environments.

At its core, iConPAL relies on a pre-trained large language model (LLM) like ChatGPT and Llama2. These modern LLMs demonstrate a nuanced ability to understand natural languages and perform various tasks, including translation. While iConPAL utilizes an LLM's capabilities, merely tasking it with translating a policy description into a specific formal policy language proves futile, as no LLMs are trained for our target policy language. Furthermore, re-training or fine-tuning an LLM presents challenges due to the substantial computational resources required and the need for a vast dataset containing pairs of $\langle \alpha, \beta \rangle$, where α represents the natural language description of a policy and β represents the translated policy expressed in the policy language.

Instead of re-training or fine-tuning, iConPAL utilizes the

979-8-3503-9193-0/24/$31.00 ©2024 IEEE
DOI 10.1109/SecDev61143.2024.00014

prompt-based in-context learning capability of LLMs. iConPAL employs *prompt engineering* to automatically generate *few-shot learning prompts* for the LLM by incorporating numerous types of contexts, such as samples of policy translation, the grammar of the policy language, and a tutorial to teach the language. Given a few-shot learning prompt, the LLM responds with the translated policy. While a context plays an important role in the translation, not all contexts are equally effective for the translation task, as we observed certain combinations yield better accuracy than others. iConPAL utilizes a subset of these contexts to form a few-shot learning prompt for translating each new policy description. Note that these contexts are pre-computed and remained unchanged over time. Samples of translations are derived from our small dataset (290 translations) curated through exploration and consultation of prior work in the IoT security landscape. Rather than creating a new policy language, we have extended an expressive policy language from [9].

To increase the success rate of the translation, iConPAL employs various methods, such as syntax validation and prompt refinement. Syntax validation enables iConPAL to identify syntax errors in the LLM's response, while prompt refinement allows it to adjust the prompt based on identified errors and query the LLM again to avoid repetition of the error. iConPAL iteratively refines the prompt until it receives a syntactically valid translation or reaches a query threshold. Consequently, iConPAL either produces a translated policy, which is syntactically valid, or fails to do so.

While syntactic validation of a translated policy is crucial, semantic validation is equally, if not more, important. Although a policy translated by iConPAL is syntactically valid, it might lack semantic validity. Manual semantic validation is possible but burdensome. To address this, we devised an automated approach using an LLM. Therefore, in evaluating iConPAL's effectiveness, we conducted both manual and automated semantic validation of the translated policies.

We developed a fully functional prototype of iConPAL using Python 3.10. For the syntax validator, we utilized ANTLR 4.13.1. Our prototype is not tied to any specific LLM like ChatGPT. Instead, we integrated a generic prompt template within iConPAL and equipped it with an LLM client. This client relays each prompt request to our offshore LLM gateway server, which is capable of interfacing with multiple LLM backends, such as ChatGPT, Llama2, and Mixtral. Notably, any change in the LLM backend will not disrupt the operation of iConPAL. When interfacing with an open-source, locally-deployable LLM like Llama2 and Mixtral, the gateway will operate on a GPU-backed server. However, for a cloud-based LLM like ChatGPT, our gateway simply implements a lightweight wrapper to communicate with the respective vendor's web API, which does not require any local GPUs.

We evaluated iConPAL on 290 policy descriptions from our dataset. Firstly, we assessed iConPAL's efficacy in translating a given policy description, measuring its success rate. iConPAL successfully translated policies at a rate of 93.61%. Among these successfully translated policies, 93.57% were semantically valid. Secondly, we conducted an ablation study to identify the contribution of iConPAL's components to its effectiveness, allowing us to determine the optimal configuration. Thirdly, we evaluated how iConPAL's performance varied with different LLMs. We found that OpenAI's GPT-4 produces the highest success rate of 93.61%, whereas Llama2 with 13B parameters had the lowest success rate 13.53%. Finally, while we manually analyzed the reported semantic validation rate in the aforementioned experiments, we also assessed the effectiveness of our automated semantic validation approach. We observed that the automated approach is 78% accurate and has the potential to serve as a first-stage filter for manual semantic validation, thus reducing the burdens on humans while incurring minimal financial cost. iConPAL is available as open-source at `https://github.com/syne-lab/iconpal`.

Contributions. This paper makes the following contributions:

- We proposed iConPAL, an automated policy authoring assistant designed to translate natural language policy descriptions into a specific formal policy language. iConPAL relies on the power of LLMs and employs several components to automate the translation process.
- We curated a small dataset comprising 290 policy translation examples and a tutorial for the policy language.
- We implemented a fully functional prototype of iConPAL and evaluated its effectiveness using policies from our dataset. iConPAL generated 93.61% syntactically valid policies, of which 93.57% were also semantically valid.
- To the best of our knowledge, iConPAL is the first automated policy authoring assistant for IoT defense solutions.

II. Preliminaries

IoT Devices and Platforms. Smart homes are composed of a myriad of IoT devices. A device can have one or more sensors or actuators or both. For example, a surveillance camera has motion sensors and actuators to turn on/off the camera for recording video and audio. A device with actuation capabilities can be operated through instructions (aka, *commands*) sent from a remote entity (*e.g.*, a mobile app, a cloud provider, an IoT platform). Similarly, a device with sensing capabilities senses a change in its surrounding environment (*e.g.*, a motion). Each device maintains the current status of its capability as an internal state and notifies the remote entity of any change in its internal state. These notifications are also referred to as events that the remote entity or the automation application can act upon.

Programmable IoT platforms (*e.g.*, SmartThings, OpenHAB, IFTTT) have paved the way for regular users to turn their traditional homes into full-fledged smart homes. While IoT devices are essentially part of the physical world, it is the platform that not only hosts the customized automation of the user's choice but also connects the IoT devices with the cyber world. Therefore, a platform plays the major role in orchestrating the automation and monitoring of IoT devices installed in a smart home.

Automation Apps. Customized automation are primarily encoded as short applications/programs (in short, *apps*). Plat-

forms like SmartThings, OpenHAB, and HomeAssistant allow users to install and edit directly the source code of the apps. On the other hand, platforms like IFTTT and Zapier allow users to configure apps using their graphical web interface. By and large, an app follows a *trigger-action* paradigm. Consider an automation app for an outdoor camera to be "`if motion_detected, turn on recording video and notify the user's phone app`". Upon sensing a motion, the sensor notifies the platform about this `motion_detected` event, which in turn will trigger the platform's app engine to execute this camera app to take actions. The execution of this app will result in sending a `recording.turn_on()` command to the camera and a notification to the user (*i.e.*, `notify_user()`). While these platforms and apps work together to facilitate customized automations for smart homes, they have also introduced a new attack surface to smart homes [26–28]. For instance, remote adversaries can exploit flaws in these apps [4, 18, 26] or third-party services [9, 13] to create security or safety issues.

Policies and Defense Mechanisms. Several defense mechanisms have been proposed to address the threats from apps and mitigate the safety and security issues of smart homes [4–11, 13, 16, 18]. These defenses often revolve around enforcing countermeasures based on policies, where a policy expresses a certain behavioral condition of a smart home that must be satisfied at all times (*e.g.*, a safety property [29]). While static analysis based defenses [4, 18] proactively identify any app that could potentially violate a given policy, runtime monitoring based approaches [6, 7, 10, 11, 13] dynamically prevent any app's contemplated actions that would violate a given policy at runtime.

Consider "`the outdoor camera must not be turned off when the user is in vacation`" be a policy. At first glance, this policy may appear to follow the trigger-action paradigm of apps. However, policies are different from apps, because policies express conditional relationships among the behavior of smart home devices, unlike apps that dictate actions when a triggering condition is satisfied. This policy can also be interpreted as a logical implication like "`UserAway` $\implies$ `Camera_is_ON`", which will be evaluated as `false` in a smart home execution scenario when `UserAway` becomes `true` but `Camera_is_ON` turns `false`. Now if an app issues a command to turn off the camera when the user is away, a defense must recognize that this command will violate the stated policy and therefore take appropriate measures – either notifying of such potential violations [10] or blocking the actions in question [6, 7].

Policy Languages. Unlike apps, writing policies has been daunting and complicated for regular smart home users (who are mostly non-tech-savvy). Apps are relatively straightforward to program because of their simplified structures, which align with how humans think, and the support of visual programming editors offered by platforms. On the contrary, policies are complex as they are conditioned on the current state or the execution history (traces) of a smart home. Prior work attempted to address the trade-off between usability and functionality, thereby resulting in many types of domain- and tool-specific policy languages, inspired by propositional logic, quantifier-free first-order logic, temporal logic, and so on. While some languages can only express system invariants (*i.e.*, policies conditioned on the current state of the home), others can express variants of temporal logic (*i.e.*, policies conditioned on the execution history). In any case, writing a policy demands users to deal with the idiosyncrasies of the tool's policy language and to possess a nuanced understanding of the IoT system's corner cases, which is almost impossible for non-tech-savvy users.

Large Language Models (LLMs). The recent success of pre-trained large language models (LLMs), such as ChatGPT, has demonstrated LLMs' ability to comprehend natural language instructions and carry out domain-specific tasks across diverse topics, without re-training the entire model. LLMs have been used for many common tasks, including, text translation. Given an instruction, also known as *prompt*, to translate a text from one language (*e.g.*, English) to another (*e.g.*, French), LLM can respond with the translated text. For example, with a prompt like "`Translate the following text into French: "LLMs are powerful"`", ChatGPT responds "`Les LLM sont puissants`". Such translations are possible because LLMs like ChatGPT were trained on a substantially large dataset of both languages and therefore LLMs learned the general rules and dependencies within each language.

However, translating a policy from its natural description in English to a policy language is challenging because LLMs are not trained on the target policy language. To overcome this challenge, re-training or fine-tuning the model is a huge undertaking as it requires a large labeled dataset and significant computing resources. Instead, LLMs can be taught a domain-specific task by including a few labeled examples of the task in the prompt, also known as a *few-shot prompting* technique. By carefully crafting and employing such examples in prompts, LLMs can be successfully adapted to carry out this domain-specific task.

III. Our Approach: iConPAL

We present our problem definition and a high-level workflow of iConPAL, along with its critical components.

A. Problem Definition

Let Σ be a set of policy descriptions written in English (a natural language) and α be a policy description such that $\alpha \in \Sigma$. Consider $\mathbb{L}$, a domain-specific policy language and $\mathbf{G}$, a grammar to produce each $\beta \in \mathbb{L}$. In other words, β is a policy written in the policy language $\mathbb{L}$. $\mathbf{T}$ represents a tutorial written for humans teaching how to write policies in $\mathbb{L}$. $\mathbf{E}$ denotes a set of sample pairs of manually translated policies, *i.e.*, $\mathbf{E} = \{\langle\alpha, \beta\rangle | \alpha \in \Sigma \text{ and } \beta \in \mathbb{L}\}$. Existing information pertaining to the policy language and the translation comprise our *knowledge base*, $\mathsf{KB} = \{\mathbf{G}, \mathbf{T}, \mathbf{E}\}$. Now, we can define the policy translation operation as $\mathcal{T} \subseteq \Sigma \times 2^{\mathsf{KB}} \times \mathbb{L}$. In other words, $\langle\alpha_i, \mathcal{C}, \beta_j\rangle \in \mathcal{T}$ means that β_j is the translated policy of α_i with respect to a configuration of the knowledge base, $\mathcal{C} \in$

```
Input: When the smoke is detected, the alarm must sound
Output:
If SmokeDetected Then AlarmSound
SmokeDetected = (SmokeSensor.status == "Detected")
AlarmSound = (Alarm.status == "Sound")
```

Fig. 1: An example of a policy translation using iConPAL

2^{KB}. Implementing a $\mathcal{T}$ is undecidable as the completeness and soundness cannot be achieved simultaneously.

Instead, iConPAL is designed to realize $\mathcal{T}_{iConPAL}$, a subset of $\mathcal{T}$, while striving to achieve soundness. To formulate $\mathcal{T}_{iConPAL}$, we need to first define two functions. Let SynValid be a function to check syntactic validity and defined as $\mathsf{SynValid} : \mathcal{S} \times \mathbf{G} \mapsto \{\texttt{true}, \texttt{false}\}$, where $\mathcal{S}$ is a set of all possible strings (*i.e.*, $\mathbb{L} \subset \mathcal{S}$). Given a string $s \in \mathcal{S}$ and a grammar $\mathbf{G}$, SynValid can determine if s is syntactically valid with respect to $\mathbf{G}$. Similarly, let SemValid be a function to check semantic validity and defined as $\mathsf{SemValid} : \Sigma \times \mathbb{L} \mapsto \{\texttt{true}, \texttt{false}\}$, where Σ and $\mathbb{L}$ are defined as above. If α is a policy text and β is a translated policy, SemValid can determine if β is semantically valid with respect to α.

Definition. *Given a policy text $\alpha \in \Sigma$ and a configuration of the knowledge base $\mathcal{C} \in 2^{\mathsf{KB}}$, $\mathcal{T}_{iConPAL}$ can be defined as $\mathcal{T}_{iConPAL} = \{\langle \alpha, \mathcal{C}, \beta \rangle | \mathsf{SynValid}(\beta, \mathbf{G}) = \texttt{true}\}$, where $\beta \in \mathbb{L}$.* iConPAL develops $\mathcal{T}_{iConPAL} \cup \{\langle \alpha, \mathcal{C}, \perp \rangle\}$. This means when iConPAL succeeds in translating α, it outputs β, but when iConPAL fails to translate, it outputs $\perp$ (*i.e.*, a message like `"translation failed"`). A syntactically valid policy (*e.g.*, β) is not necessarily semantically valid. To ensure semantic validity, iConPAL utilizes SemValid to check if β is semantically valid, *i.e.*, $\mathsf{SemValid}(\alpha, \beta) = \texttt{true}$.

B. Overview of iConPAL's Design

Given a policy description α, iConPAL translates it to a format (β) written in the desired policy language. Instead of designing a new policy language, iConPAL leverages an existing policy language [9], which is expressive and generic enough to capture different policies common for IoT systems. We have extended the grammar ($\mathbf{G}$) of this language ($\mathbb{L}$) with some operators (see Appendix A).

Example. iConPAL takes the description of a policy in English as an input and outputs the corresponding translated policy written in $\mathbb{L}$, as shown in Figure 1. A user wants to ensure that the fire alarm must sound whenever the smoke sensor detects smoke. The user can utilize iConPAL to obtain the translated policy, where `SmokeDetected` and `AlarmSound` are predicates and defined as conditions on the state of the device.

iConPAL's Workflow. Figure 2 presents the architecture of iConPAL. Internally, iConPAL consists of multiple individual modular components that work in concert to translate a given policy description. The user writes a description of the intended policy, referred to as a policy text α, in the iConPAL's interface (❶). iConPAL forwards α to its prompt generator (❷). The prompt generator utilizes a knowledge base (KB) consisting of a collection of example translations ($\mathbf{E}$), the grammar ($\mathbf{G}$) of the desired policy language $\mathbb{L}$, and a tutorial ($\mathbf{T}$) for humans to write policies using $\mathbf{G}$. While iConPAL allows the user to select any preferred combination $\mathcal{C} \in 2^{\mathsf{KB}}$ as a prompt config, the default configuration of iConPAL uses the combination that yielded the best result in our evaluation (see § V). Given a prompt configuration, the prompt generator constructs an intermediate few-shot prompt (ρ) composed of α and information from KB according to $\mathcal{C}$ (❸). Note that ρ follows a generic template, not tied to any particular LLM.

Our LLM client takes in ρ, converts it into a full-fledged prompt (γ), and sends γ to our off-shore LLM server that interfaces with several LLMs (*e.g.*, ChatGPT, Llama2) (❹). Note that γ follows a customized prompt template (shown in Figure 3) that is devised for our LLM client and server and is expressive enough to encode information required for actual prompts to query all 7 LLMs we used in our evaluation. Upon the receiving the query, the LLM backend generates a response (λ) containing the translated policy (❺). Often, λ is surrounded by explanations or comments for the user, and therefore, our response extractor extracts the policy formula (φ) from λ (❻).

iConPAL employs a syntax validator (*i.e.*, a parser for $\mathbf{G}$) to check if φ is syntactically valid. If so, iConPAL outputs the valid translated policy β (❼). If not valid, iConPAL is equipped with a refinement feature to guide the LLM by using the validator generated error message (R). If this option is enabled, the prompt refiner creates a refined prompt (Ψ) by encapsulating the last γ, the last received λ, and R and forwards Ψ to the LLM client (❽) to initiate the next round of query. This refinement loop continues until φ becomes valid or the iteration reaches a user-provided threshold. Upon reaching the threshold, iConPAL outputs $\perp$ (*i.e.*, `translation failed`).

What iConPAL outputs (*i.e.*, β) is only syntactically valid. However, semantic validity of β is not part of this pipeline and is conducted separately, as shown in Figure 4. For semantic validation of the translated policy (β), we introduce an *automated* approach, `AutoSemVal`, by harnessing the power of an existing LLM and *differential checking*. Let β be the translated policy of the text α. `AutoSemVal` queries the LLM for a natural language description of β, which we refer to as α'. Next, `AutoSemVal` constructs another prompt to ask the LLM to assess if both α and α' are equivalent. In other words, `AutoSemVal` checks if two descriptions of the policy (β) have the same semantic meaning. The response of the LLM dictates whether β is semantically valid or not. Furthermore, for the assessment of iConPAL's efficacy, we performed a manual semantic validation of β in parallel (as shown in Figure 4).

C. Knowledge Base

iConPAL utilizes a knowledge base to construct an effective learning prompt for guiding the LLM to translate the given policy description. Currently, iConPAL incorporates three types of the policy-translation related information into the knowledge base: a grammar of the policy language, a tutorial to use the grammar for translation, and a collection of sample policy translations. While more types of information can be added to the knowledge base, we observed these three

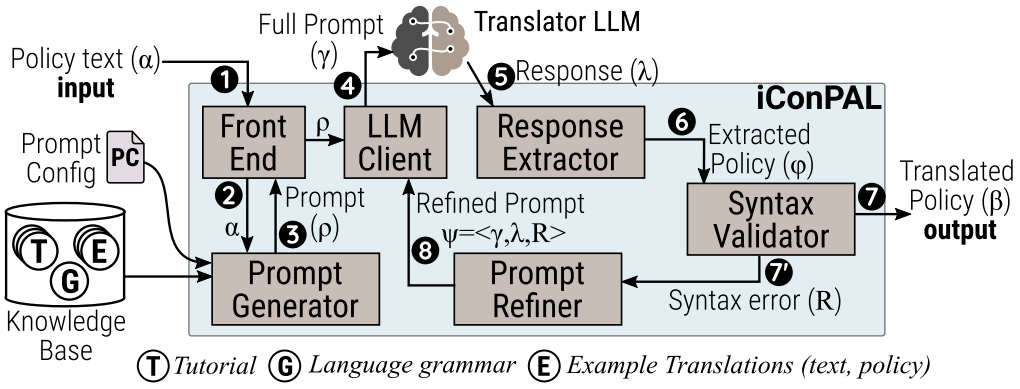

Fig. 2: iConPAL's architecture and workflow

```
Role: You are a natural text to policy translator.

Contexts:
{Grammar information}
{Tutorial information}
{Example_1: translation of text to policy}
...
{Example_n: translation of text to policy}

Task:
Translate the following natural text to policy language.
"When the smoke is detected, the alarm should sound."
```

Fig. 3: The structure of a full prompt (γ). Texts in braces {...} are placeholders

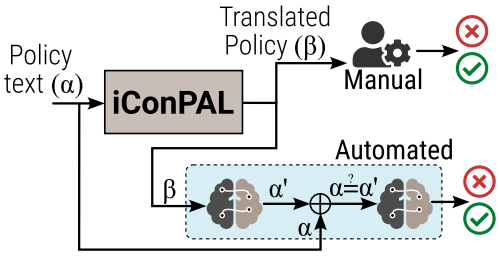

Fig. 4: Semantic Validation of iConPAL's translations

types are readily available and enough to translate policies with high accuracy.

Grammar (G). iConPAL uses the grammar **G** of Maverick's policy language [9]. We have encapsulated **G** using the syntax of Extended Backus-Naur Form (EBNF) [30]. Based on the prompt configuration, some of our generated learning prompts include **G** to help the LLM learn the underlying policy grammar. We decide to include **G** in the learning prompts because **G** is a context-free grammar and modern LLMs understand context-free grammars.

Tutorial (T). We developed a *full tutorial* to help humans learn the nuances of the grammar **G** and write a policy in this language. We decide to include this tutorial in the learning prompts so that the tutorial can help LLM learn how to construct a policy in this language. However, we quickly realized the length of the full tutorial was becoming a limiting factor as each LLM has a maximum allowed length for a prompt in terms of tokens (words) and the closed-source LLMs charge financially (USD) based on the prompt size. If a prompt crosses this limit, the LLM will raise an error, hindering the translation process. To address this issue, we devised two more variations of the tutorial: *simplified* and *summarized*. While the simplified tutorial has less information than the full tutorial, the summarized tutorial is essentially a short gist. Both the simplified and the summarized tutorials were generated with the help of ChatGPT (using GPT-3.5).

Example translations (E). Pre-trained LLMs perform demonstrably well if a prompt includes some task-specific examples for the LLM to learn [31], eliminating the need for re-training or fine-tuning LLM's parameters. To harness this power of LLMs fueled by few-shot prompts, we need a collection of policy translation examples, from their English description to the policy language format. Unfortunately, no such publicly available dataset exists. Therefore, we manually curated a small *dataset* of policy translations. We collected a set of policy descriptions from the recent IoT literature and manually converted each description into a policy using **G**.

We varied the number of examples in the prompt to identify the minimum number of examples required to achieve the best accuracy. Our evaluation reveals that 3 examples are enough (§V-B2). Furthermore, not all the examples are relevant for the policy text in question. To identify relevant examples, we devise a *categorical* approach to classify each policy text. We systematically classified the policy texts of our dataset into *eight* categories. While doing so, we also formulated a pattern-based definition for each category. Although our dataset includes the manual categorization of each policy text, iConPAL employs an automated approach to classify a policy text into one of these 8 categories by using an LLM. iConPAL encapsulates these categorical definitions in the prompt, asking the LLM to classify the policy text in question. Additional details on the dataset will be discussed in section V.

Algorithm 1: Prompt Generator

Input : The prompt configuration $\mathcal{C}$, policy text α, category list $\mathcal{L}$, policy to category map $\mathcal{M}$, and knowledge base KB

Output: The prompt

```
tutorial ← retrieveTutorial (KB, C)
grammar ← retrieveGrammar (KB, C)
if C.categorizer = "manual" then
    category ← getPolicyCategory (α, M)
else if C.categorizer = "llm" then
    category ← askLLMToCategorize (α, L)
else
    category ← random (L)
cnt ← getExampleCount (C) // cnt: Example Count
examples ← retrieveSimilarExamples (KB, category, cnt)
prompt ← concat (tutorial, grammar, examples, α)
return prompt
```

D. Prompt Generator

The prompt generator in iConPAL is a pivotal component, which we will describe here. While space restricts detail on other components, understanding it offers insights into iConPAL's inner workings.

Algorithm 1 outlines how the prompt generator operates. It takes in a prompt configuration $\mathcal{C}$, the knowledge base KB, the list of category definitions $\mathcal{L}$, the policy-to-category map $\mathcal{M}$, and a policy text α as input. It retrieves the desired tutorial and the grammar from KB as dictated by $\mathcal{C}$. However, depending on the configuration $\mathcal{C}$, `tutorial` and `grammar` can also be $\varnothing$ (empty).

Now, the policy text α will be categorized based on the selection strategy (*i.e.*, $\mathcal{C}$.categorizer) specified in $\mathcal{C}$. It supports three variations: `manual`, `llm`, and `random`. In the `manual` approach, it utilizes the map $\mathcal{M}$ to categorize α. In the `llm` approach, it queries the LLM to categorize α according to the definitions in $\mathcal{L}$. In the `random` approach, it randomly selects a category for α.

Next, the prompt generator focuses on the learning examples. It determines the number of examples (*i.e.*, `cnt`) mentioned in $\mathcal{C}$ and extracts `cnt` examples from the same category as α. Again, if `cnt` is 0, then `examples` will be $\varnothing$. Finally, it concatenates `tutorial`, `grammar`, `examples`, and α to construct the prompt, which will be returned as its output. Appendix J presents the actual LLM prompts and responses while translating two policy texts as case studies.

IV. Implementation

We now discuss the implementation of some components of iConPAL and how we addressed some technical challenges.

Prompt Generator. We implemented the prompt generator in Python 3.10 (118 LoC). Based on a prompt configuration written in JSON and the input policy text, the prompt generator creates a learning prompt in a customized format that will include a combination of one of the tutorials, the grammar, and some example translations from our knowledge base.

Syntax Validator. We implemented a policy language parser for the syntax validator using Python 3.10 and ANTLR 4.13.1. We used ANTLR to define the grammar ($\mathbf{G}$) of the policy language $\mathbb{L}$ and generate the parser. The grammar is 50 lines. The custom code added to the ANTLR generated parser is only 56 LoC. This parser can validate the syntax of a policy with respect to $\mathbf{G}$.

Response Extractor. Despite requesting only the translated policy, responses from some LLMs include additional comments and explanations. To extract the policy from the response, we implemented a response extractor in Python 3.10 (32 LoC). Technically, we leveraged our policy parser to validate the syntax of each line of the response, one at a time, and reconstructed the translated policy by combining the valid lines only.

LLM Gateway Server and Client. iConPAL uses an LLM solely for inference, with each LLM having its own interface. Instead of direct interfacing, we connect iConPAL to an LLM gateway via a client-server approach. The client, integrated into iConPAL, remains LLM-independent, while the gateway server interfaces with each LLM. To handle model variations, the server provides a unified HTTP interface, operating offshore. For details, see Appendix F.

V. Evaluation

In this section, we present a comprehensive evaluation of iConPAL. Our assessment covers both the effectiveness of its end-to-end operation and the performance of its individual components.

Dataset and Categorization. To prepare few-shot learning prompts for LLMs, we need to embed a few examples of policy translations along with the policy description in question. We manually curated a dataset of policy translations. We collected 290 natural language (English) descriptions of IoT policies that describe how IoT systems should behave. These policies are sourced from 16 recent IoT security literature [4–9, 13, 16, 18, 21–23, 32–35] spanning 2015–2023. Most of these sources proposed IoT defense systems and used a set of policies for their evaluation. We manually converted each description into a policy using the language grammar ($\mathbf{G}$).

Recall that our prompt generator relies on policy categorization to decide which examples to be embedded into a prompt. We manually classified the policy texts in our dataset into *eight* categories. This categorization was based on the structure of their English descriptions. For instance, policies like "*In any situation, surveillance cameras must remain on.*" and "*In any situation, front doors must remain locked.*" are grouped together in category G1, as they assert a state that must be true with/without any triggering event/condition. Appendix H presents the rest of the categories along with their policy count.

While these 8 categories may be influenced by our dataset's policies, drawn from recent IoT literature, we consider them

to offer a broad representation. Additionally, new policy types can be seamlessly integrated into iConPAL's configuration without requiring modifications to its implementation.

Dataset creation (policy collection and translation) took 26 person-hours and the categorization task took 7 person-hours. One author curated and translated policies while two others verified translations independently to ensure accuracy.

In evaluation, our policy dataset served as both the knowledge base and the test set. For each experiment, we maintained a disjoint knowledge base and test set, with the latter also functioning as a holdout dataset.

Experimental Setup. We employed both closed- and open-source LLMs, 7 models in total. For closed-source models, we used OpenAI's GPT-3.5 [36] and GPT-4 [37]. For open-source models, we used Llama2-13B [38], Llama2-70B (Quantized) [39], Mistral [40], Mixtral [41], and Yi-34B [42]. We used our in-house developed LLM gateway server and client to interact with these models. We deployed the Llama2-70B model in its quantized version on a server with 2 NVIDIA A100 GPUs and 250 GB RAM, while other open-source models ran on a server with 1 NVIDIA A100 GPU and 125 GB RAM. GPT-3.5 and GPT-4 were utilized through OpenAI's API endpoints for inference.

We primarily used GPT-3.5 to evaluate iConPAL's efficacy, unless stated otherwise. Each inference was conducted on an LLM with no chat history to prevent bias. Results represent the average of 3 runs due to financial constraints from GPU server rentals and API usage. Initially, we tried free-shared GPU servers, but encountered delays due to resource availability.

Research Questions. We seek to answer the following research questions:

- **RQ1.** How effective is iConPAL in translating policies from natural description (text)? (§ V-A)
- **RQ2.** How do different components contribute to iConPAL's effectiveness? (§ V-B)
- **RQ3.** How do different LLMs contribute to iConPAL's effectiveness? (§ V-C)
- **RQ4.** How effective is our LLM-powered automated semantic validation? (§ V-D)
- **RQ5.** What is the incurred overhead of iConPAL in terms of cost and time? (§ V-E)

A. RQ1: Effectiveness of iConPAL

We evaluated the effectiveness of iConPAL in translating policies from their descriptions in natural language (we refer to them as *policy texts*).

For this assessment, we used the optimal prompt configuration identified in our ablation study (§V-B). The optimal configuration utilizes GPT-4 as the translation LLM, a model temperature of 0.5, the full tutorial, and three similar examples. If the LLM produces a syntactically invalid policy, we refined the learning prompt by embedding the syntax error as additional context, repeating this process up to twice. Even after refining the prompt twice, if the LLM produces a syntactically invalid policy, iConPAL outputs $\perp$ (*i.e.*, "translation failed").

TABLE I: Reference of Notations

Symbol	Description
Succ.	Successful
Sem.	Semantic
C_T	Total policy texts of the test data set
C_{tr}	Count of Successful translation
C_{sm}	Count of Semantically valid policies
R_{tr}	Rate of successful translation w.r.t. C_T
R_{sm}	Rate of semantically valid policies w.r.t. C_T
R_{smtr}	Rate of semantically valid policies w.r.t. C_{tr}
C_{Ex}	Count of policy examples in a learning prompt
Sel_{Ex}	The example selection strategy used for a prompt

TABLE II: Effectiveness of iConPAL (using the optimal prompt configuration). Notations are explained in Table I.

Total (C_T)	**Successful Translation**		**Semantic Validation**		
	C_{tr}	R_{tr}	C_{sm}	R_{sm}	R_{smtr}
266	249	93.61%	233	87.59%	93.57%

Out of 290 policy descriptions in our dataset, we manually translated 24 policy texts, 3 from each of the eight categories, and used them as our knowledge base. The remaining 266 policy texts are used as our test data set (a holdout data set) to assess iConPAL's effectiveness. For each test policy text, our prompt generator selected 3 similar examples using the manual selection strategy from our knowledge base.

To ensure reliability, we conducted the experiment three times with different example sets. Table II shows the average of the three runs. We observed that iConPAL successfully translated 93.61% policies (249 out of 266). Among these, 93.57% (233 policies) are semantically valid (*i.e.*, the translated policy matches its natural description). Policies that lack semantic validity fail to capture the nuances of the natural text. Appendix B shows such an example. Appendix I provides additional insights on translation validity and consistency.

B. RQ2: Ablation Study

We performed an ablation study on iConPAL to assess the impact of its components on effectiveness. The study comprised three segments: (i) the effect of three knowledge base components, (ii) component-specific variations, and (iii) prompt refinement effects. Using GPT-3.5 as the model and a temperature of 0.5, we conducted each experiment three times with different example sets to account for variance.

1) Effect of three knowledge base components: Our knowledge base comprises three components: examples (**E**), tutorial (**T**), and grammar (**G**). We evaluated the individual and combined effects of these components on iConPAL's efficacy, as reported in Table III.

For experiments involving examples in the prompt, we manually translated 8 policy texts (1 from each of the eight categories) out of 290 to serve as our knowledge base, leaving 282 for testing. For each test policy text, our prompt generator selected 1 similar example using the manual selection strategy from the knowledge base. In other experiments, where examples were not included in the prompts, we used all 290 texts as the test dataset.

TABLE III: Effect of three components of the knowledge base (examples (**E**), the tutorial (**T**), the grammar (**G**)). ✓: *present*, ✗: *not present*. Notations are explained in Table I

Components			Total(C_T)	Succ. Translation		Sem. Validation	
E.	T.	G.		C_{tr}	R_{tr}	C_{sm}	R_{sm}
✓	✓	✓	282	229	81.21%	179	63.48%
✓	✓	✗	282	228	80.85%	180	63.83%
✓	✗	✓	282	127	45.04%	104	36.88%
✓	✗	✗	282	141	50.00%	114	40.43%
✗	✓	✓	290	211	72.76%	150	51.72%
✗	✓	✗	290	227	78.28%	155	53.45%
✗	✗	✓	290	0	0.00%	0	0.00%
✗	✗	✗	290	0	0.00%	0	0.00%

Without examples, tutorial, or grammar in the prompt, the translation rate (R_{tr}) was 0.0%. Even after adding grammar alone, R_{tr} remained at 0.0%, indicating the LLM's struggle with complex grammar and syntax errors in policy creation. Adding just one example increased R_{tr} to 50.0%, while using only the full tutorial resulted in R_{tr} of 78.28%. Combining both tutorial and examples proved most effective, yielding an R_{tr} of 80.85% and a semantic validity rate (R_{sm}) of 63.83%.

Combining **G** with **T** and **E** slightly increases R_{tr}, but decreases R_{sm}, demonstrating a negative impact. Our EBNF-format grammar, though concise (25 lines, 524 tokens), overwhelms the LLM, hindering its ability to generate syntactically correct translations. This aligns with findings from [43], where increasing grammar complexity led to LLM unreliability. Natural language descriptions of grammar yielded better results, as observed with our tutorial. Thus, **T** and **E** together represent the most effective combination for iConPAL's performance.

2) Effect of component-specific variations: We studied the impact of various example selection strategies, prompt sizes, and tutorial types.

• **(a) Impact of example selection.** We assessed how different example selection strategies in learning prompts affect policy generation quality. Recall that we had three selection strategies: *manual*, *LLM*, and *random*. 8 policy examples, one for each of 8 categories, were chosen as the knowledge base from 290 policies, leaving 282 for testing.

For the manual strategy, we utilized the manually assigned category of each test policy text and included the corresponding category's example. For the LLM strategy, GPT-3.5 categorized the policy text, and we included the corresponding example. Finally, the random strategy involved randomly selecting one of the eight examples to include in the prompt.

Table IV displays iConPAL's average rates for each example selection strategy. The manual strategy had the highest translation rate (R_{tr}) at 78.01%, followed by the LLM strategy at 77.30%. The LLM strategy also achieved the highest semantic validity rate (R_{sm}). Given its comparable performance to manual selection but without additional human effort, the LLM-powered strategy is recommended as iConPAL's default.

• **(b) Impact of the number of examples.** We evaluated the impact of varying the number of examples in the prompt (C_{Ex}) on iConPAL's efficacy, ranging from 0 to 3 examples. Table V illustrates the corresponding rates.

We selected policy examples (pairs of policy text α and translated policy β) for each category from our dataset to comprise our knowledge base. The remaining policy texts served as the test dataset. For instance, to prepare prompts with 3 examples, we utilized 24 translated examples (3 for each category) out of 290. iConPAL was then tested with the remaining 266 policy texts. Prompts with 3 examples showed superior performance, with a translation rate of 83.08% and semantic validity of 67.67%.

TABLE IV: Impact of example selection strategies. Notations are explained in Table I

Strategy (Sel_{Ex})	Total(C_T)	Succ. Translation		Sem. Validation	
		C_{tr}	R_{tr}	C_{sm}	R_{sm}
Random	282	215	76.24%	143	50.71%
Manual	282	220	78.01%	171	60.64%
LLM	282	218	77.30%	174	61.70%

TABLE V: Impact of the number of examples in a prompt. Notations are explained in Table I

Count (C_{Ex})	Total(C_T)	Succ. Translation		Sem. Validation	
		C_{tr}	R_{tr}	C_{sm}	R_{sm}
0	290	204	70.34%	138	47.59%
1	282	220	78.01%	171	60.64%
2	274	221	80.66%	171	62.41%
3	266	221	83.08%	180	67.67%

While theoretically possible to increase the number of examples, we limited our experimentation to a maximum of three due to constraints on prompt length and cost. Furthermore, we found that employing two rounds of refinement on a prompt with three examples yielded only marginal performance improvement compared to the first refinement (more in § V-B3).

• **(c) Impact of the tutorial type.** We assessed how different variants of the same tutorial affect iConPAL's efficacy. Recall that we had three variations: *full*, *simplified*, and *summarized*. The full tutorial offers detailed step-by-step instructions and examples. The simplified version, generated by GPT-3.5 using the full tutorial, has simplified instructions and examples. The summarized version, also generated by GPT-3.5, includes simplified instructions without examples. Token counts, as per OpenAI's Tokenizer, are: full (1212 tokens), simplified (888 tokens), and summarized (296 tokens).

We chose 8 policy examples, one for each category, from 290 policies to comprise the knowledge base, leaving 282 for testing. For each test policy text, our prompt generator utilized the manual selection strategy to select 1 similar example. Table VI shows the achieved rates for each tutorial type. Prompts generated with the full tutorial had the highest translation rate (80.85%) and semantic validity (63.83%).

The success of prompts with the full tutorial is due to its detailed instructions with examples. However, this option is the most costly due to its longer token length. Conversely, the simplified tutorial can be a more economical choice, albeit with a slight reduction in iConPAL's efficacy.

3) Effect of Prompt Refinement: We assessed how prompt refinement affects iConPAL's efficacy. Recall that when iConPAL's translation LLM generates a syntactically invalid policy, we refine the learning prompt using the syntax error and ask

TABLE VI: Impact of the tutorial type. Notations are explained in Table I

Tutorial	Total(C_T)	Succ. Translation		Sem. Validation	
		C_{tr}	R_{tr}	C_{sm}	R_{sm}
Full	282	228	80.85%	180	63.83%
Simplified	282	220	78.01%	171	60.64%
Summarized	282	148	52.48%	119	42.20%

TABLE VII: Effect of prompt refinement. Notations are explained in Table I

Refinement Limit	Total(C_T)	Succ. Translation		Sem. Validation	
		C_{tr}	R_{tr}	C_{sm}	R_{sm}
0	266	230	86.47%	192	72.18%
1	266	238	89.47%	197	74.06%
2	266	241	90.60%	198	74.44%

LLM to regenerate the policy. In this experiment, we varied the number of prompt refinements from 0 to 2.

Out of 290 policies, we chose 24 policy examples, 3 from each category, to form our knowledge base, leaving 266 for testing. For each test policy text, our prompt generator used the manual selection strategy to select 3 similar examples from the knowledge base.

Table VII displays the achieved rates, notably, both rates consistently improved with each refinement. Following the 2nd refinement, iConPAL's average translation and semantic validity rates reached 90.60% and 74.44% respectively. We anticipate further refinements would enhance policy quality, albeit at increased translation costs. However, prompt length limitations in LLMs constrain indefinite refinement.

C. RQ3: Variation in LLMs

We evaluated the effect of the model (LLM) and its temperature on iConPAL's efficacy.

1) Effect of the model temperature: The model temperature influences the randomness of a LLM's responses. We assessed how adjusting the model temperature parameter influences iConPAL's efficacy by varying the temperature (0.0, 0.5, 1.0, and 1.5) of GPT-3.5. We conducted the experiment three times with different example sets to account for variance.

As before, we selected 8 policy examples, 1 from each category, as our knowledge base, leaving 282 for testing. For each test policy, our prompt generator chose 1 similar example using the manual selection strategy. Table C1 displays the achieved rates. Notably, a temperature of 0.5 yielded the highest translation and semantic validity rates at 78.37% and 61.70% respectively. Higher temperatures increase randomness, resulting in more diverse but potentially invalid policies.

2) Effect of the model (LLM): We evaluated the impact of seven different language models (LLMs) on iConPAL's efficacy: GPT-3.5, GPT-4, Llama2-13B, Llama2-70B (Quantized), Mistral, Mixtral, and Yi-34B. Our study utilized a consistent learning prompt configuration (full tutorial with 3 examples), allowed prompt refinement twice, and maintained a temperature of 0.5 across all LLMs.

Out of 290 policies, we selected 24 policy examples, 3 from each category, as our knowledge base, leaving 266 for testing. For each test policy text, our prompt generator selected 3 similar examples using the manual selection strategy.

Table C2 presents the performance rates for each LLM. GPT-4 notably outperformed other models, achieving translation (R_{tr}) and semantic validity (R_{sm}) rates of 93.61% and 87.59% respectively. GPT-3.5 closely followed with rates of 90.60% for R_{tr} and 74.44% for R_{sm}. Among the remaining models, Llama2-70B (Quantized) surpassed all, while Llama2-13B exhibited the lowest performance.

The varied performance of different LLMs in the same task is influenced by factors like size, architecture, and training datasets [44]. These underlying details are often undisclosed. The impact of LLM selection on iConPAL's performance is evident in Table C2. GPT-4's superior performance aligns with its market dominance. Larger models like Llama2-70B tend to outperform smaller ones like Llama2-13B, highlighting the significance of model size.

D. RQ4: Automated Semantic Validation

Recall that not all policies translated by iConPAL are semantically valid, and the rates (R_{sm}) we reported so far were manually assessed. Three authors spent a total of 83 person-hours validating 13,964 translated policies, averaging 0.36 minutes per policy.

To streamline the semantic validation of a translated policy, iConPAL developed `AutoSemVal`, an LLM-powered automated approach (see Figure 4). `AutoSemVal` was supplied with the full tutorial and 3 similar example translations for each policy. By using GPT-3.5 as the translation LLM, it achieved 85% precision, 88% recall, 78% accuracy, and 86% F1 score, effectively identifying semantically valid policies while minimizing false positives and negatives (see Table G4). However, it struggled to identify semantically invalid policies, recognizing only 42% of them. We also observed that `AutoSemVal`'s efficacy remained almost identical for both GPT-3.5 and GPT-4. Appendix G presents this comparison, along with insights on `AutoSemVal`'s misclassified policies.

E. RQ5: Performance Overhead

Having addressed the efficacy of iConPAL in previous research questions, we now discuss its performance overhead in terms of time and financial cost. Due to space constraints, we provide only the total overhead here: 40 hours of time and a total cost of 169.05 USD for all conducted experiments. Further breakdown is available in Appendix D.

VI. Discussion

Automated Semantic Validation. iConPAL's automated semantic validation currently achieves a 78% accuracy rate. Nonetheless, as LLMs improve, it holds the potential to serve as an initial filter, further reducing the already minimal human effort required for manual semantic validation in comparison to manual policy translation (see Appendix E).

Grammar Dependent. The current prototype of iConPAL is specific to the policy language ($\mathbb{L}$) and the grammar ($\mathbf{G}$) proposed by [9]. We picked a language that is powerful enough

to express many types of policies. Yet, it is possible that the translated policies in $\mathbf{G}$ may not be directly utilized for some specific language (say, $\mathbb{L}'$) based on a different grammar (say, $\mathbf{G}'$). For example, IoT defense solutions like [5, 8, 45] use a different policy language other than `Maverick`'s [9]. We can address this concern in two different approaches: (a) we can replace $\mathbf{G}$ with $\mathbf{G}'$ and update the collection of policy translation examples for this new $\mathbf{G}'$; and (b) we can employ a source-to-source parser-based translator to convert iConPAL's output β written in $\mathbf{G}$ to β' written in $\mathbf{G}'$. The latter approach is relatively easier and less cumbersome as these grammars are straightforward and unambiguous, unlike natural languages, and writing parser-based translators is not challenging.

Implications for Practice and Research. Our prototype of iConPAL can translate policies for `Maverick` [9], a recent IoT defense solution, out-of-the-box. Additionally, iConPAL is adaptable to other IoT defense solutions like `IoTGuard` [5] and `IOTSAFE` [8], as previously noted. This adaptability will streamline the comparative evaluation of IoT defense solutions using testing platforms like VetIoT [46] against IoT policy benchmarks. Despite promising results, iConPAL's semantic validation accuracy is currently 87.59%, indicating room for improvement. While this accuracy warrants caution in real-world deployment of the translated policies, iConPAL substantially advances automated policy authoring for IoT defense solutions. We believe it will spur further research in the IoT field, ultimately leading to enhanced accuracy.

Threats to Validity. The main threat is the generalizability of our findings, addressed by using a diverse set of IoT policies from existing literature. We conducted experiments 3 times to address result variance and used multiple independent authors for manual validation to minimize validation risks.

VII. Related Work

Security and safety concerns in smart homes have led to numerous policy-enforcing mechanisms [4–21] to protect users from misconfigurations and vulnerabilities. However, these mechanisms often require users to write complex policies in specific formats. iConPAL addresses this issue by automating policy authoring from natural language descriptions.

In an attempt to reduce the burden on end-users, prior work adopted several approaches: (a) template-based policy forms for the users to fill out [8, 14, 15], (b) temporal property synthesis based on static analysis of IoT apps and the users' interactions with the apps [16, 17, 24], (c) invariant synthesis based on supplied positively and negatively labeled execution traces [9], and (d) feedback-based systems that block all sensitive scenarios and ask users to resolve at runtime [10, 47]. Unfortunately, they fall short in reducing human burden. Both template-based and property-synthesis still require users to have a decent knowledge of IoT apps and temporal logic. The invariant synthesizer imposes the additional requirement of various execution traces. Finally, feedback-based systems can render users vulnerable to fatigue attacks, consequently leading to security failures. On the contrary, iConPAL requests the policy description from end-users in English and then automatically translates the policy into a domain-specific policy language, thereby eliminating the steep learning curve associated with writing correct IoT policies.

Natural language processing and speech recognition have been employed in prior work. Goffinet *et al.* [48] proposed a speech assistant that poses questions to users and recognizes their intentions, which are then used to generate policies. Hϵlion [22] developed a statistical smart home model based on user-supplied smart home descriptions. This model is utilized to generate realistic test events rather than policies directly. In practice, policy creation involves human intervention. While these methods necessitate multiple interactions with users, iConPAL offers an automated policy generation process that requires no user involvement during translation.

Prior work [49–53] have utilized LLMs to translate natural language sentences into Linear Temporal Logic (LTL) formulas. However, these systems typically handle only simple descriptions, often limited to coarse-grained commands. `Efficient-Eng-2-LTL` [51] and `Lang2LTL` [52] generate LTL expressions from structured commands to guide robot actions. `nl2spec` [53] is a human-in-the-loop translator that utilizes LLMs for translating sub-formulas of the given natural description. `NL2LTL` [50] fills in existing templates, and `NL2TL` [49] processes specialized command structures but cannot handle simple free-form texts like "*If A, then not B*." While `NL2LTL` can identify devices involved in a policy description, it fails to recognize each device's status, crucial for IoT policies. In contrast, iConPAL comprehends fine-grained, device-specific information within free-form policy descriptions and encodes necessary details in the translated policy.

Additionally, recent work has also demonstrated the use of LLMs for various translation-related tasks: extracting invariants (in natural language) from hardware design specifications [54], identifying contradictions in natural-language statements [55], and generating code from specifications [56]. Similarly, iConPAL uses LLMs for natural language processing but focuses on translating IoT system policies, which require handling intricate device-specific details and fine-grained conditions unique to IoT systems. This necessitates a specialized in-prompt learning campaign with IoT-specific examples, a tailored policy-language tutorial and refinement.

VIII. Conclusion

We introduced iConPAL, an automated assistant for crafting safety and security policies in smart homes. By translating natural language descriptions into a formal policy language without human intervention, iConPAL addresses a key challenge in traditional IoT defenses. Leveraging modern LLMs, iConPAL achieved a 93.61% successful translation rate, with 93.57% of these being semantically valid, highlighting its potential to advance research in the field.

Acknowledgments

We thank the reviewers and our shepherd for their insightful comments and suggestions. This research was supported by the National Science Foundation under grant CNS-2007512.

References

[1] OpenHAB, https://www.openhab.org.

[2] SmartThings, https://www.smartthings.com/.

[3] IFTTT, https://ifttt.com/.

[4] Z. B. Celik, P. McDaniel, and G. Tan, "Soteria: Automated IoT safety and security analysis," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, 2018, pp. 147–158.

[5] Z. B. Celik, G. Tan, and P. D. McDaniel, "Iotguard: Dynamic enforcement of security and safety policy in commodity iot." in *NDSS*, 2019.

[6] M. Yahyazadeh, P. Podder, E. Hoque, and O. Chowdhury, "Expat: Expectation-based policy analysis and enforcement for appified smart-home platforms," in *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies*, 2019.

[7] M. Yahyazadeh, S. R. Hussain, E. Hoque, and O. Chowdhury, "Patriot: Policy assisted resilient programmable iot system," in *International Conference on Runtime Verification*. Springer, 2020.

[8] W. Ding, H. Hu, and L. Cheng, "Iotsafe: Enforcing safety and security policy with real iot physical interaction discovery," in *the 28th Network and Distributed System Security Symposium (NDSS 2021)*, 2021.

[9] M. H. Mazhar, L. Li, E. Hoque, and O. Chowdhury, "Maverick: An app-independent and platform-agnostic approach to enforce policies in iot systems at runtime," in *Proc. of ACM Conference on Security and Privacy in Wireless and Mobile Networks 2023 (WiSec '23)*, 2023.

[10] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, and A. Prakash, "Contexlot: Towards providing contextual integrity to appified iot platforms." in *NDSS*. San Diego, 2017.

[11] K. Kafle, K. Jagtap, M. Ahmed-Rengers, T. Jaeger, and A. Nadkarni, "Practical integrity validation in the smart home with homeendorser," in *ACM WiSec*, 2024.

[12] R. Sahay, W. Meng, D. S. Estay, C. D. Jensen, and M. B. Barfod, "Cybership-iot: A dynamic and adaptive sdn-based security policy enforcement framework for ships," *Future Generation Computer Systems*, vol. 100, pp. 736–750, 2019.

[13] H. Chi, Q. Zeng, X. Du, and L. Luo, "Pfirewall: Semantics-aware customizable data flow control for smart home privacy protection," in *Network and Distributed System Security Symposium*, 2021.

[14] P. H. Nguyen, P. H. Phung, and H.-L. Truong, "A security policy enforcement framework for controlling iot tenant applications in the edge," in *Proceedings of the 8th International Conference on the Internet of Things*, 2018, pp. 1–8.

[15] A. Alkhresheh, K. Elgazzar, and H. S. Hassanein, "Daciot: Dynamic access control framework for iot deployments," *IEEE Internet of Things Journal*, vol. 7, no. 12, pp. 11 401–11 419, 2020.

[16] Y. Tian, N. Zhang, Y.-H. Lin, X. Wang, B. Ur, X. Guo, and P. Tague, "SmartAuth:User-Centered authorization for the internet of things," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 361–378.

[17] L. Zhang, W. He, J. Martinez, N. Brackenbury, S. Lu, and B. Ur, "Autotap: Synthesizing and repairing trigger-action programs using ltl properties," in *2019 IEEE/ACM 41st international conference on software engineering (ICSE)*. IEEE, 2019, pp. 281–291.

[18] H. Chi, Q. Zeng, X. Du, and J. Yu, "Cross-app interference threats in smart homes: Categorization, detection and handling," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2020, pp. 411–423.

[19] D. T. Nguyen, C. Song, Z. Qian, S. V. Krishnamurthy, E. J. Colbert, and P. McDaniel, "Iotsan: fortifying the safety of iot systems," in *ACM CoNEXT*, 2018.

[20] Q. Wang, P. Datta, W. Yang, S. Liu, A. Bates, and C. A. Gunter, "Charting the attack surface of trigger-action iot platforms," in *ACM CCS*, 2019.

[21] Q. Wang, W. U. Hassan, A. Bates, and C. Gunter, "Fear and logging in the internet of things," in *Network and Distributed Systems Symposium*, 2018.

[22] S. Manandhar, K. Moran, K. Kafle, R. Tang, D. Poshyvanyk, and A. Nadkarni, "Towards a natural perspective of smart homes for practical security and safety analyses," in *2020 IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2020.

[23] C. Fu, Q. Zeng, and X. Du, "Hawatcher: Semantics-aware anomaly detection for appified smart homes," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[24] C. Nandi and M. D. Ernst, "Automatic trigger generation for rule-based smart homes," in *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, 2016, pp. 97–102.

[25] M. H. Mazhar, "Improving the safety of iot systems with usable policy enforcement," Ph.D. dissertation, The University of Iowa, 2023.

[26] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *IEEE S&P*, 2016.

[27] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "Sok: Security evaluation of home-based iot deployments," in *2019 IEEE symposium on security and privacy (sp)*. IEEE, 2019, pp. 1362–1380.

[28] B. Hammi, S. Zeadally, R. Khatoun, and J. Nebhen, "Survey on smart homes: Vulnerabilities, risks, and countermeasures," *Computers & Security*, vol. 117, p. 102677, 2022.

[29] L. Lamport, "Proving the correctness of multiprocess programs," *IEEE transactions on software engineering*, no. 2, pp. 125–143, 1977.

[30] "Extended backus-naur form," https://en.wikipedia.org/wiki/Extended_Backus-Naur_form.

[31] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[32] R. Liu, Z. Wang, L. Garcia, and M. Srivastava, "Remediot: Remedial actions for internet-of-things conflicts," in *Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, 2019.

[33] W. Ding and H. Hu, "On the safety of iot device physical interaction control," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 832–846.

[34] C.-J. M. Liang, B. F. Karlsson, N. D. Lane, F. Zhao, J. Zhang, Z. Pan, Z. Li, and Y. Yu, "Sift: building an internet of safe things," in *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*, 2015, pp. 298–309.

[35] Y. Yu and J. Liu, "Tapinspector: Safety and liveness verification of concurrent trigger-action iot systems," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 3773–3788, 2022.

[36] O. GPT-3, https://platform.openai.com/docs/models/gpt-3-5-turbo.

[37] O. GPT-4, https://platform.openai.com/docs/models/gpt-4-and-gpt-4-turbo.

[38] Llama2-13b, https://huggingface.co/meta-llama/Llama-2-13b-chat-hf.

[39] Llama2-70B, https://huggingface.co/TheBloke/Llama-2-70B-Chat-GGUF.

[40] Mistral, https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.1.

[41] Mixtral, https://huggingface.co/mistralai/Mixtral-8x7B-v0.1.

[42] Yi-34B, https://huggingface.co/01-ai/Yi-34B-Chat.

[43] "Craft domain-specific grammars," https://www.linkedin.com/pulse/craft-domain-specific-grammars-marie-chatfield-rivas/.

[44] "Even llms need education–quality data makes llms overperform," https://stackoverflow.blog/2024/02/26/even-llms-need-education-quality-data-makes-llms-overperform/.

[45] M. O. Ozmen, X. Li, A. Chu, Z. B. Celik, B. Hoxha, and X. Zhang, "Discovering iot physical channel vulnerabilities," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2415–2428.

[46] A. J. Nafis, O. Chowdhury, and E. Hoque, "Vetiot: On vetting iot defenses enforcing policies at runtime," in *2023 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2023, pp. 1–9.

[47] H. J. Kang, S. Q. Sim, and D. Lo, "Iotbox: Sandbox mining to prevent interaction threats in iot systems," in *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2021, pp. 182–193.

[48] S. Goffinet, D. Schmitz, I. Zavalyshyn, A. Legay, and E. Riviere, "Controlling security rules using natural dialogue: an application to smart home care," in *Proc. of UbiComp/ISWC '21*, 2021, pp. 194–197.

[49] Y. Chen, R. Gandhi, Y. Zhang, and C. Fan, "Nl2tl: Transforming natural languages to temporal logics using large language models," *arXiv preprint arXiv:2305.07766*, 2023.

[50] F. Fuggitti and T. Chakraborti, "Nl2ltl–a python package for converting natural language (nl) instructions to linear temporal logic (ltl) formulas," in *AAAI Conference on Artificial Intelligence*, 2023.

[51] J. Pan, G. Chou, and D. Berenson, "Data-efficient learning of natural language to linear temporal logic translators for robot task specification," *arXiv preprint arXiv:2303.08006*, 2023.

[52] J. X. Liu, Z. Yang, I. Idrees, S. Liang, B. Schornstein, S. Tellex, and A. Shah, "Grounding complex natural language commands for temporal tasks in unseen environments," in *Conference on Robot Learning*. PMLR, 2023, pp. 1084–1110.

[53] M. Cosler, C. Hahn, D. Mendoza, F. Schmitt, and C. Trippel, "nl2spec: interactively translating unstructured natural language to temporal logics with large language models," in *International Conference on Computer Aided Verification*. Springer, 2023, pp. 383–396.

[54] M. Hassan, S. Ahmadi-Pour, K. Qayyum, C. K. Jha, and R. Drechsler, "Llm-guided formal verification coupled with mutation testing," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2024, pp. 1–2.

[55] A. E. Gärtner and D. Göhlich, "Automated requirement contradiction detection through formal logic and llms," *Automated Software Engineering*, vol. 31, no. 2, p. 49, 2024.

[56] P. Kogler, A. Falkner, and S. Sperl, "Reliable generation of formal specifications using large language models," in *SE 2024-Companion*. Gesellschaft für Informatik eV, 2024, pp. 141–153.

APPENDIX A. ABSTRACT GRAMMAR

According to the policy language of [9], a policy is $\Omega ::= \bigwedge_j I_j$, where an invariant I_j can be written in using the

```
Input:
Fire sprinkler should only be on when there's fire
detected in the home.
Output:
If FireDetected Then FireSprinklerOn
  FireDetected = (FireDetector.status == "Detected")
  FireSprinklerOn = (FireSprinkler.status == "On")
```

Fig. B1: Semantically invalid policy generated by iConPAL

TABLE C1: Effect of the model temperature. Notations are explained in Table I

Temperature	Total(C_T)	Succ. Translation		Sem. Validation	
		C_{tr}	R_{tr}	C_{sm}	R_{sm}
0.0	282	219	77.66%	173	61.35%
0.5	282	221	78.37%	174	61.70%
1.0	282	205	72.70%	162	57.45%
1.5	282	155	54.96%	127	45.04%

following grammar:

$$I ::= \textbf{If } \Phi \textbf{ Then } \Psi$$
$$\Phi ::= p \mid \Phi 1 \textbf{ and } \Phi 2 \mid \Phi 1 \textbf{ or } \Phi 2 \mid \textbf{not } \Phi \mid \textbf{true}$$
$$\Psi ::= \Phi \mid \Psi 1 \textbf{ since } \Psi 2 \mid \textbf{yesterday } \Psi$$
$$p ::= t1 \oplus t2 \mid p1 \textbf{ and } p2 \mid p1 \textbf{ or } p2 \mid \textbf{not } p \mid \textbf{true}$$
$$t ::= x \ \epsilon \ V \mid \langle const. \rangle \mid \text{func.}(t1, \ldots, t_n)$$

The policy invariant *I* follows the pattern: "**If** Φ **Then** Ψ" where Φ and Ψ are logical expressions. Φ can be a predicate p, the constant **true**, or logical combinations of them. Ψ includes standard past temporal operators like **since** and **yesterday**, along with the logical expression Φ. This flexibility lets us express more complex invariants by considering past events. The predicate p can be a relational operator $\oplus$ (such as $\leq$, $\neq$) applied to a pair of terms or logical combinations of multiple predicates. A term t can be a variable x from the set of variables V, a constant $\langle const. \rangle$ ("ON", "OFF", 5), or a function $func.$ applied to one or more terms (*e.g.*, $timer(x)$).

Appendix B. An Example of a Semantically Invalid Policy

Figure B1 shows an example. The correct policy should be "`If FireSprinklerOn Then FireDetected`", capable of flagging a violation of the policy when `FireSprinkler` in `ON` even if there is no fire. But the generated policy is semantically incorrect because iConPAL's LLM failed to recognize that the given text essentially expresses an '`only if`' relationship. We observed that for some different texts, the LLM accurately translated when '`only if`' was used in the description.

Appendix C. Results of RQ3

The impact of the model selection on iConPAL's performance is shown in Table C2, and the impact of the model temperature on GPT-3.5 is presented in Table C1.

TABLE C2: Effect of the model (LLM). Notations are explained in Table I

Model	Total(C_T)	Succ. Translation		Sem. Validation	
		C_{tr}	R_{tr}	C_{sm}	R_{sm}
GPT-4	266	249	93.61%	233	87.59%
GPT-3.5	266	241	90.60%	198	74.44%
Llama2-70B	266	231	86.84%	181	68.05%
Mixtral	266	202	75.94%	176	66.17%
Mistral	266	187	70.30%	131	49.25%
Yi-34B	266	93	34.96%	72	27.07%
Llama2-13B	266	36	13.53%	24	9.02%

TABLE D3: Policy translation cost and duration for different models (Duration in minutes and Cost in USD)

	GPT 3.5	GPT 4	Llama2 13B	Llama2 70B	Mistral	Mixtral	Yi 34B
Duration	35[1]	50[1]	20	100	20	20	200
Cost	2	13	0.8	8	0.8	0.8	8

[1]To address rate limit error, 2-6 seconds of delay was added using exponential backoff retry mechanism, so the duration may vary based on usage tier.

Appendix D. A Breakdown of Performance Overhead

For a one-to-one comparison, we reported the average overhead across different models in Table D3. We collected this data during our experiment of measuring the effect of the models on iConPAL's efficacy (§ V-C2). While we utilized OpenAI's web API endpoints for GPT-4 and GPT-3.5, we rented GPU servers from a third-party GPU cloud provider to deploy the open-source LLMs (*e.g.*, Llama2, Mistral, Mixtral, Yi-34B).

GPT-4 required 50 minutes to translate 266 policies using the optimal prompt configuration, averaging 11 seconds per translation and resulting in a cost of 13 USD ($0.05 per translation). In comparison, GPT-3.5 completed the same task in 35 minutes (8 seconds per translation) at a cost of 2 USD ($0.008 per translation). Llama2-13B finished this task in 20 minutes (4.5 seconds per translation), costing 0.80 USD ($0.003 per translation). Llama2-70B (Quantized) took 100 minutes (22.5 seconds per translation), costing 8 USD ($0.03 per translation). Mistral and Mixtral incurred the same cost and time, 20 minutes and 0.8 USD per model ($0.003 per translation). Since each experiment with a model was repeated thrice, it required a total cost of 100.20 USD and a cumulative time of 22 hours and 15 minutes.

The exploration of the optimal prompt configuration during our ablation study with GPT-3.5 incurred a cost of 27 USD and took 11 hours. Across all experiments, 13,964 translated policies were syntactically valid. We performed automated semantic validation for all syntactically valid translated policies with GPT-3.5. We used 10 parallel connections to OpenAI's API endpoint to speed up the validation process. It took about 6 hours and 12 minutes (1.6 seconds per validation), costing 11.51 USD ($0.00082 per validation). We also performed

automated semantic validation for the transations produced by the optimal prompt configuration using GPT 4. GPT 4 took about 35 minutes (2.8 seconds per validation) to validate 748 translations, costing 30.34 USD ($0.04056 per validation).

Grand total. All the experiments we conducted for iConPAL required 40 hours and incurred a cost of 169.05 USD.

Appendix E. Comparison of manual efforts in translation vs. semantic-validation

Manual translation of policy-texts requires meticulous attention to three sub-tasks: designing the policy structure, writing the policy, and checking semantic correctness. This process is time-consuming and burdensome. In contrast, our semantic validation focuses solely on checking the semantic correctness of translated policies, which are already syntactically correct, making it a significantly faster process.

While we lack precise data on the time needed for manual translation of 290 policies, we measured that 26 person-hours were spent on both collecting and manually translating these policies. Note that the collection task, which involves copying policy texts from 16 IoT papers, is significantly simpler compared to the manual translation task. Assuming equal time allocation for both collection and translation tasks (a rough conservative estimate), manual translation costs approximately 2.69 minutes per policy (= 13 person-hours / 290 policies). In contrast, semantic validation of 13,843 policies consumed 83 person-hours, costing approximately 0.36 minutes per policy.

Appendix F. LLM Gateway Server and Client

Our LLM gateway server supports three backend engines: llama.cpp, pytorch-based transformers, and OpenAI endpoints. We used the models for inference only. By default, if the model is supported by llama.cpp, the server uses this engine; otherwise, it falls back to pytorch-based transformers. For ChatGPT (GPT-3.5 and GPT-4 models), the server uses a lightweight wrapper to communicate with the OpenAI's proprietary endpoints. Furthermore, the server was designed to effectively address the disparity among these three backends and different models, providing a unified stateless HTTP interface to the client. As LLMs differ in their prompt templates and documentations, we resolved ill-documented prompt templates by exploring the code base of some LLMs to figure out the correct format.

We used Python 3.10 to implement the server (1907 LoC) and the client (780 LoC). We also utilized some popular python packages, such as, `fire`, `sentencepiece`, `gguf`, `transformers`, `openai`, `fastapi`, `torch`, `llama-cpp-python`, and `protobuf`.

While interacting with OpenAI, we noticed that the translation time differs from one policy to another. If a smaller request timeout is used, the OpenAI client retries too frequently, thereby incurring more cost. We addressed this by calibrating the timeout and retry settings to reduce the incurred cost. Furthermore, OpenAI has a rate limit on tokens per minute (*e.g.*, 80,000 tokens per minute for Tier-2). To handle rate limit errors, we calibrated the retry settings.

TABLE G4: Semantic validation confusion matrix for `AutoSemVal` (our automated approach to semantically validate the translated policy). `TP`: True Positive, `FP`: False Positive, `TN`: True Negative, `FN`: False Negative.

		Predicted by `AutoSemVal`		
		Valid=`Yes`	Valid=`No`	Total
Manually checked (**Ground truth**)	Valid=**Yes**	`TP`=9,664	`FN`=1,372	11,036
	Valid=**No**	`FP`=1,690	`TN`=1,238	2,928
	Total	11,354	2,610	13,964

TABLE G5: Effectiveness of `AutoSemVal` with respect to GPT 3.5 and GPT 4 in terms of Precision, Recall, Accuracy, Specificity, and F_1 Score. Sample size: 748.

Model	Precision	Recall	Accuracy	Specificity	F_1 Score
GPT 3.5	96%	88%	85%	36%	92%
GPT 4	95%	91%	87%	19%	93%

Appendix G. Automated Semantic Validation

Misclassification by `AutoSemVal` vs. Policy Categories. We assessed the translated policies misclassified by `AutoSemVal` (see Table G4) and examined any correlation with their categories. Appendix H details the categories of our dataset. Policies from G2 had the highest misclassification rate, with an average of 4 out of 6 policies misclassified, followed by G5 (6 out of 13) and G8 (8 out of 21). Policies from these categories often involve the phrase "`only if`" (or its variants, such as `except`) with the `deny` keyword (for G2), temporal operators like `lastly` (for G5), and the `allow` keyword (for G8). We found that LLM struggles to interpret the meaning of `only if` when combined with other logic. Misclassification rates for other categories were relatively low, below 10%.

GPT-3.5 vs. GPT-4. To measure the effect of the LLM on `AutoSemVal`, we ran the experiment of assessing `AutoSemVal`'s efficacy twice: once using GPT-4 and next using GPT-3.5. Since GPT-4 is more expensive than GPT-3.5, we utilized a smaller sample size for a one-to-one comparison. We used a small sample set (748) of translated policies instead of 13,964 policies. This sample set was the collection of successfully translated policies over 3 runs of the experiment conducted in Section V-A (see Table II). We found that the efficacy of `AutoSemVal` stays almost identical for GPT-3.5 and GPT-4 (see Table G5), indicating a little to no effect of the LLM on `AutoSemVal`.

Appendix H. Dataset Categorization

We manually classified the policy texts in our dataset into *eight* categories. This categorization was based on the structure of their English descriptions. Table H6 shows the categories along with some examples. It also includes the number of policies that belong to each category.

Appendix I. More on Validity and Consistency

In this section, we present additional results on translation validity and consistency observed during our experiment for RQ1 in Section V-A.

TABLE H6: Dataset Categorization

Category	Description	Count
G1	Assert a state that must be true with/without any triggering event/condition. Examples: - In any situation, surveillance cameras must remain on. - In any situation, front doors must remain locked.	11
G2	Indicate blocking an action when a condition has been met. Examples: - Deny all HTTP requests. - Deny turning on the coffee machine only if the user is not at home.	9
G3	Indicate a straightforward correlation between the states of two devices. Examples: - If fire is detected in the home then fire sprinkler should be on. - If the door is open then the light should be on.	57
G4	Indicate a correlation between the states of two devices with a time constraint. Examples: - If the door is open for more than 5 minutes then the light should be on. - If the temperature is above 30 degrees for more than 10 minutes then the air conditioner should be on.	23
G5	Indicate a correlation between current state and past state of one or more devices Examples: - Allow light to be turned off only if lastly it was on. - Allow hallway light to be turned on only if the hallway motion sensor has tripped since the hallway light was off.	17
G6	Indicate a comparison between device state and other factors. Examples: - The heater should be turned on if the temperature is below 40. - The window should be open when the room temperature is above the threshold and there are people in the room.	57
G7	Indicate a correlation among more than two device states. Examples: - If user is away, on vacation, or sleeping then door should be locked. - When sink water leakage is detected and motion sensor is inactive, a text message should be sent.	91
G8	Indicate allowing an action when a condition has been met. Examples: - Allow the light to be turned on only if the user is at home. - Allow the heater to be turned on only if the temperature is below 40.	25
	Total	290

Translation failure vs. Policy Categories. In our experiment, we observed that iConPAL failed to translate on average 17 out of 266 policies. Further inspection revealed that policies from category G6 posed the greatest challenge for LLM. Appendix H details the categories of our dataset. For example, a G6 policy states: "`When the temperature is above 60° and no one is present, the heater should not be turned on.`" We found that LLM struggled with policies involving device-state-specific constraints and comparisons with abstract or predefined values. The translation failure rate for G6 was about 22% on average, while failure rates for other categories remained below 5%.

Semantic Validity vs. Policy Categories. We assessed the translated policies that were syntactically correct but semantically invalid based on our manual analysis (see Table G4) and examined any correlation with their categories. Translated policies from G2 had the highest rate of semantic invalidity (33%), with an average of 2 out of 6 policies being semantically invalid. Policies from G2 often involve the phrase "`only if`" (or its variants, such as `except`) with the `deny` keyword. We found that LLM struggles to interpret the meaning of `only if` when combined with other logic. The rates of semantically invalid translated policies for other categories were relatively low, below 10%.

LLM's Consistency. We assessed GPT-4's consistency in generating valid or invalid policies. In each run of this experiment, we randomly selected 24 policies from the dataset as our knowledge base and used the remaining ones as our test dataset. While the size of the test dataset remained constant, its composition varied across runs. We found 229 policies common to all 3 runs. Among these, 215 policies (93.88%) had consistent results, either consistently successful or consistently failed in translation across all three runs. Additionally, 192 out of 229 policies (83.84%) were semantically consistent across all three runs.

APPENDIX J. A JOURNEY OF A POLICY TEXT TRANSLATION

We outline the journey of translating a policy text by demonstrating an actual LLM prompt constructed by iConPAL for the policy text, the received LLM response, and further refinement iConPAL applied as needed. We present case studies with two policy texts as follows.

- Study1 (translation required no refinement): *`If smoke is detected, then gas stove should be turned off.`*
- Study2 (translation required refinement): *`In any situation, room temperature should never be over 100.`*

Each LLM prompt consists of several key components: **Role**, **Tutorial**, **Example Translations**, and **Instruction**. The role outlines the expectations for the LLM. Both the tutorial and example translations comprise the context of the learning prompt, as the tutorial serves as a guide for translating natural language text into policy language and the example translations demonstrate this process in action. The instruction points out the translation task for the LLM.

For the LLM prompts generated to query the policy texts of our case studies, we will show the common parts below and the policy text specific parts in their respective subsections (Appendix J-A and Appendix J-B).

Role. iConPAL includes the role component in each LLM prompt as follows:

```
Role:
You are a plain text to formal policy translator.
I will teach you how to translate a plain text to
formal policy with a tutorial and some examples.
```

Tutorial. The inclusion of the tutorial in each prompt depends on the specific prompt configuration. The tutorial remains fixed for our selected policy language and grammar ($\mathbf{G}$) and does not vary with the policy texts being tested. An excerpt of the tutorial is provided below due to space constraints:

```
Tutorial:
In IoT defenses, natural language can be structured into
a format that is easily parseable. This tutorial
explores how to convert English statements into a policy
language using logical expressions and variables.

Logical Expressions and Variable
Consider the statement: ''If Fire Sprinkler is on, then
Water Valve is on." This can be expressed as following.

'''
If FireSprinkler.status =='ON' Then WaterValve.status=='ON'
'''

However, for improved readability, we can assign the
logical expression into a variable and use the variable
in policy statement. The above policy statement can be
re-written as follows.

'''
If FireSprinklerOn Then WaterValveOn

FireSprinklerOn = (FireSprinkler.status == 'ON')
WaterValveOn = (WaterValve.status == 'ON')
'''

Device attributes
In the above example, we used 'status' attribute of
'FireSprinkler' and 'WaterValve' devices. You can
use other device attributes as well.

Functions
Besides device attibutes, function calls can be employed
in logical expressions. For example, we can use
'timer(logical_expression)' function. We assume that the
'timer' function returns the elapsed time since the logical
expression became true.

Text:
If water leak sensor sensed wet within 1 minute, then
turn off water sprinkler.

Policy:
'''
If (timer(WaterLeakSensorWet) > 0 and
    timer(WaterLeakSensorWet) < 60) Then WaterSprinklerOn
WaterLeakSensorWet = (WaterLeakSensor.status == 'Wet')
WaterSprinklerOn = (WaterSprinkler.status == 'On')
'''

This example demonstrates the use of the timer function to
determine if a condition occurred within a specified time
frame.
---- Snipped ----
```

A. Case Study 1: Translation required no refinement

For this study, our policy text under test was "`If smoke is detected, then gas stove should be turned off.`".

Example Transitions. Depending on the prompt configuration, iConPAL includes some examples in the prompt. For this case study, we used a configuration that made iConPAL select 3 similar example translations from our dataset as follows:

```
Text:
If the door is open, then the light should be on.

Policy:
If DoorOpen Then LightOn
DoorOpen = (Door.status == ''Open")
LightOn = (Light.status == ''On")

Text:
If fire is detected in the home, then fire sprinkler
should be on.

Policy:
If FireDetected Then FireSprinklerOn
FireDetected = (FireSensor.status == ''Detected")
FireSprinklerOn = (FireSprinkler.status == ''On")

Text:
If heater is on, then air conditioner should be off.

Policy:
If HeaterOn Then AirConditionerOff
HeaterOn = (Heater.status == ''On")
AirConditionerOff= (AirConditioner.status == ''Off")
```

Instruction. This component of the prompt expresses what policy text the LLM is expected to translate given the context. The instruction is shown below:

```
Instruction:
Translate the following natural text to formal policy.
Only provide the translated policy itself without any
explanations or extra words.
''If smoke is detected, then gas stove should be
turned off."
```

LLM Response. The received response from the translation LLM is shown below:

```
If SmokeDetected Then GasStoveOff
SmokeDetected = (SmokeSensor.status == ‘‘Detected”)
GasStoveOff = (GasStove. status == ‘‘Off”)
```

Since the translated policy was syntactically correct, iConPAL did not apply any refinement. iConPAL provided the translated policy as an output.

B. Case Study 2: Translation required refinement

For this study, our policy text under test was “`In any situation, room temperature should never be over 100.`”.

Example Transitions. Depending on the prompt configuration, iConPAL includes some examples in the prompt. For this case study, we used a configuration that made iConPAL select 3 similar example translations from our dataset as follows:

```
Text:
In any situation , surveillance camera must remain on.

Policy:
If true Then SurveillanceCamera. status == ‘‘On”

Text:
In any situation , front door must remain locked.

Policy:
If true Then FrontDoor. status == ‘‘Locked”

Text:
In any situation , TV speaker volume should never be
more than 90.

Policy:
If true Then not(TVSpeakerVolumeOver90)
TVSpeakerVolumeOver90 = TVSpeaker.volume > 90
```

Instruction. This component of the prompt expresses what policy text the LLM is expected to translate given the context. The instruction is shown below:

```
Translate the following natural text to formal policy.
Only provide the translated policy itself without any
explanations or extra words.
‘‘In any situation , room temperature should never
be over 100.”
```

LLM Response. The received response from the translation LLM is shown below:

```
If true Then not(RoomTemperatureOver100)
RoomTemperatureOver100 = RoomTemperature > 100
```

Since the translated policy was not syntactically correct, iConPAL refined the prompt and appended the following instruction for the LLM:

```
Refinement Instruction :
Your translation is invalid due to following reason.
Reason: ‘RoomTemperature’ variable used but not defined.
Please try again.
```

LLM’s Refined Response. The received response from the translation LLM is shown below:

```
If true Then not(RoomTemperatureOver100)
RoomTemperatureOver100 = (RoomTemperature.status > 100)
```

This translated policy was syntactically correct, as this version used a device’s attribute which does not require a definition unlike a variable, and therefore iConPAL displayed this translation as the output.