Translating Natural Language Specifications into Access Control Policies by Leveraging Large Language Models

Sherifdeen Lawal
Institute for Cyber Security (ICS)
University of Texas at San Antonio, Texas, United States
sherifdeen.lawal@utsa.edu

Xingmeng Zhao
Department of Information Systems and Cyber Security
University of Texas at San Antonio, Texas, United States
xingmeng.zhao@utsa.edu

Anthony Rios
Department of Information Systems and Cyber Security
University of Texas at San Antonio, Texas, United States
anthony.rios@utsa.edu

Ram Krishnan
Department of Electrical & Computer Engineering
Institute for Cyber Security (ICS)
University of Texas at San Antonio, Texas, United States
ram.krishnan@utsa.edu

David Ferraiolo
National Institute of Standards and Technology, Gaithersburg,
Maryland, United States
dferraiolo@nist.gov

***Abstract*—This paper investigates the application of large language models (LLMs) for the automated translation and information extraction of access control policies from a natural language source. Prior research in this domain have predominantly relied on manual methods, traditional natural language processing (NLP), or a hybrid approach involving machine learning and artificial neural networks combined with NLP techniques. We demonstrate a significant advancement by leveraging the power of LLMs to achieve improved efficiency and accuracy in these tasks. Our study focuses on applying cutting-edge prompt engi- neering techniques designed to optimize LLM performance in the specific context of access control policy information extraction. The findings highlight the potential of LLMs to streamline the process of converting human-readable requirements into formal, machine-interpretable policies, ultimately contributing to the automation and security of access control systems.**

***Index Terms*—Natural Language Specification, Access Control Policies, Large Language Models (LLMs), Prompt Engineering, Attribute-Based Access Control (ABAC)**

I. Introduction

Translating NLACPs to formal models presents challenges, primarily due to the ambiguity and variability inherent in natural language. Central to this translation is extracting ACP entities, such as subjects, objects, actions, and conditions. Historically, approaches to address these challenges have included manual extraction methods [1], the application of Natural Language Processing (NLP) techniques for named entity recognition (NER) and relationship extraction (RE) [2]–[4], and advanced methods combining NLP with machine learning or deep learning to enhance accuracy and scalability [5], [6]. This study aims to extend the body of work on automating the extraction of access control policy entities [2], [3], [5]–[9] in natural language specification by leveraging Large Language Models (LLMs) to translate policies explicitly or implicitly stated in natural language text documents. These policies are often found across a myriad of documents, including but not limited to agile user stories, configuration management documentation, compliance documents, data classification policies, security policies, and standard operating procedures (SOPs). These documents collectively articulate access control policies necessary to maintain robust security systems. However, the traditional manual process of sifting through these diverse sources to extract relevant policy information is laborious, inefficient, and error-prone. As organizations strive for better governance and quicker responsiveness in a highly regulated environment, there is a pressing need to streamline this process.

Herein lies the potential of leveraging large language models (LLMs) to automate the extraction and translation of access control policy expressions embedded within these disparate documents. This approach represents a paradigm shift that promises enhanced efficiency and accuracy over previous methodologies that heavily relied on manual efforts coupled with traditional Natural Language Processing (NLP) techniques and Deep Neural Networks (DNNs). The shift towards employing LLMs can significantly reduce the limi- tations encountered in earlier methodologies, paving the way for more intelligent and reliable access control mechanisms. Emerging advancements in artificial intelligence, particularly large language models (LLMs) such as OpenAI's GPT, offer a transformative solution to this problem. Capable of understanding and generating human-like text, LLMs present

an unprecedented opportunity to automate the extraction and translation of natural language access control policies. These models can parse diverse document types swiftly, discerning contextual cues and translating them into structured access control rules with remarkable precision.

This work investigates the use of in-context learning, specifically prompting, for the translation of Access Control Policies (ACPs) from natural language text documents. In-context training involves providing a few examples (the "prompt") alongside the input text to guide the large language model (LLM) towards the desired output. This contrasts with fine-tuning, where the LLM's weights are adjusted on a large dataset of labeled examples, improving its performance on a specific task.

For instance, a prompt such as, "Given the policy statement, extract subjects and actions," can effectively direct the LLM to identify and contextualize critical entities and relationships within natural language text. Prompting is particularly advantageous in this context because it allows for rapid adaptation to new document formats and terminologies without the extensive data requirements and computational costs associated with fine-tuning. This makes it an efficient approach for dynamically extracting entities and relation that we studied.

This study proposes the use of state-of-the-art large language model (LLM) prompting techniques, specifically Program-of-Thought (PoT), to efficiently extract policy enti- ties and relations from natural language specifications to an Attribute-Based Access Control. Program-of-Thought incorpo- rates programming logic paradigms directly into the prompting process, allowing the LLM to interpret and manipulate the text with an understanding akin to computer programming. These methodologies offer significant advantages in understanding complex dependencies and relationships within policy texts, making them ideal for extracting detailed entities and their interrelations.

The Program-of-Thought prompting technique can be readily adapted for other access control models like Role-Based Access Control (RBAC) and Resource-Based Access Control (ReBAC). For RBAC, the prompts can be modified to focus on extracting roles, permissions, and their assignments. Similarly, in ReBAC, the prompts can be adapted to extract resources, actions, and their associated permissions.

To present the contributions of this work, we outline the key advancements as follows:

1) We pioneer the application of large language models (LLMs) to extract access control policies articulated in natural language and convert them into formal access policy specifications.
2) We employ cutting-edge prompting techniques to effectively identify and extract entities and relationships within policy texts.
3) We develop a synthesized access control corpus generated by LLMs.

II. Related Work

This related work section is divided into two parts: one reviewing earlier studies that use previous approaches to extract access control policy information (entities and relationships), and the other examining research inspired by large language models to extract entities and relationships.

A. Pre-LLMs: Extracting Entities and Relations

Slankas et al. [7] introduced Access Control Relation Extraction (ACRE), a machine learning approach designed to extract elements of Access Control Policies (ACP) from natural language documents. The ACRE process can be divided into two main phases: identifying ACP sentences and extracting ACP elements. In the identification phase, the authors explored the potential of using words, their synonyms, parts of speech (POS), and named entities as markers to detect ACP sentences. For the elements extraction phase, they employed a bootstrapping method that relies on patterns derived from the dependency tree representation of sentences to extract instances of ACP accurately. An extension of the previous study, Slankas et al. [8] have tested the suggested method on extensive datasets gathered from five different policy data sources that have been referenced in earlier studies. During the identification phase, they utilized the K-nearest neighbor (K- NN) learning algorithm to distinguish between ACP sentences and other forms of sentences.

Turner et al. [1] developed an assistance tool for authoring Attribute-Based Access Control (ABAC) policies that enables a security architect to set up an ABAC expression using a series of fields such as subject, subject attributes, object, and object attributes, among others. The tool accepts natural lan- guage ABAC policies that conform to these predefined fields. The goal is to offer a user-friendly, business-level environment for writing ABAC policies, while the application handles the conversion of these inputs into machine-readable ABAC rules. In contrast to our proposal, the authors method requires the manual extraction of information relevant to the ABAC rule from the natural language access control policy (NLACP). They took a manual approach and noted that automation of this task is possible.

Narouei et al. [2] proposed a top-down role engineering approach to extract access control policies from unrestricted natural language requirements documents. They leverage natural language processing techniques, specifically Semantic Role Labeling (SRL), to automate the extraction of these policies. This method aims to define roles and build a Role-Based Access Control (RBAC) system effect. Further study by Narouei et al. [5] present a policy development framework for Attribute-Based Access Control (ABAC) designed to automatically derive policies from freely structured natural language texts. Their approach involves a method for extracting policy-specific information using advanced deep neural networks, deep recurrent neural network (RNN), on a compilation of annotated dataset containing sentences sourced from actual policy documents. The trained deep recurrent neural network

(RNN) distinguishes between sentences that include access control policies (ACP) and those that do not.

Abdelgawad et al. [4] approach includes algorithms that utilize spaCy, a natural language processing library, to identify and extract entities and relationships from Access Control Policy (ACP) sentences, subsequently transforming them into the Next Generation Access Control (NGAC) model. For analysis purposes, they further translate this NGAC model into a Neo4j representation.

B. Code-LLMs: Extracting Entities and Relations

A Large Language Model (LLM) trained on extensive datasets of code and natural language documents can convert natural language instructions into structured code [10]–[12]. [13] shows that this ability to translate between language and code can serve as a bridge to connect language and semantic structure, which is crucial for tasks like semantic parsing and information extraction in NLP. Intuitively, mapping a semantic relation structure to code is more straightforward compared to natural language, which often requires careful prompt engineering [13]–[17]. Programming languages inherently represent complex and interdependent structures more effectively [18]. [19] propose CodeKGC, a type of Code- LLM that leverages code language models for generating Knowledge Graphs, to obtain better performance.

In our CODE4POLICY prompt, we first translate the entities and relations into Python class definitions. Conditioned on these class definitions and the input sentence, we prompt LLMs to generate an instance of the relation class. From this instance, we can extract the predicted relation node in JSON format. This allows CODE4POLICY to handle multiple relationships between the same pair of entities as well as multiple entities within a single sentence.

III. Approach

Processing Access Control Policy expressions start with identifying relationships between core entities and extracting these entities. Relationships expressed in Access Control Policy can be direct relationship such as ownership or parent- child (e.g., an employee can view their own payroll record) or indirect relationship such as through common attribute or role (e.g., a user is permitted to view a document if they belong to the same department as the document).

Our research shows that using prompt methods, without altering parameters or retraining large language models (LLMs), is effective in recognizing and extracting entities and their relationships as described in an access control policy. This study employs zero-shot and few-shot prompt strategies, guided by a framework inspired by Wang et al. [13], and supplemented with guidelines adapted from the work of Sainz et al. [11]. Large language models (LLMs) have significantly improved the precision of information extracted from natural language documents. Nonetheless, it is imperative that we do not depend exclusively on these models' outputs. It remains essential for a security administrator or another domain expert to validate the predictions made by the LLMs.

```
1  @dataclass
2  class CyberEvent:
3    '''
4    Defines a general entity within the
     cybersecurity context and categorizes entities
     based on their roles within relationships.
5    '''
6    Attributes:
7    '''
8        input_text: str - Textual description or
     identification of the cybersecurity event.
9        user_attributes: List[str] - Roles or titles
      of users involved in the event.
10       object_attributes: List[str] -
     Characteristics of data or objects involved.
11       individual_users: List[str] - Identifiers or
      names of individual users mentioned in the
     context of the event.
12       individual_objects: List[str] - Identifiers
     or names of individual objects or resources
     mentioned.
13   '''
14
15   input_text: str
16   user_attributes: List[str]
17   object_attributes: List[str]
18   individual_users: List[str]
19   individual_objects: List[str]
```

Listing 1. Prompt Pattern for Extracting Access Control Policy Entities

This method adopts a Python code-based format for both the input and output of the model. This technique offers a clear and human-readable structure while addressing many common challenges associated with natural language instructions. It allows any information extraction task to be represented in a consistent format. Inputs can be automatically standardized using Python code formatters like Black, resulting in well-organized output that is easy to parse. Additionally, most modern LLMs have been pre-trained on datasets that include code, indicating that these models are already familiar with this form of representation.

Although for simplicity we have segmented the code4policy prompt into two sections corresponding to the Python classes for policy entities and relations, these two program fragments are combined into a single prompt payload for the model API call. The code listing 1 illustrates the three principal components of this format: schema definition, input text, and output annotations.

The schema definition encompasses labels depicted as Python class (CyberEvent line 2), guidelines articulated within docstrings (line 4), and representative class member that includes the input are annotated as Python string (lines 8 - 12). Output is generated by the model and annotations are represented as a list (lines 15 - 19) of instances of the class defined on the schema definition part.

The guidelines for the CyberRelation class provide an overview of the task, which involves identifying various types of relationships between policy elements (entities) within the context of a CyberEvent. These guidelines also list a predefined set of relationships between these policy elements. In accordance with the access control model, compliant class

```
@dataclass
class CyberRelation:
    '''
    Defines relationships between two entities
    within the cybersecurity event context by
    leveraging the detailed context provided by the
    associated CyberEvent class.
    Each relationship is categorized based on
    predefined interaction types, which include:

    - 'r' for read: Accessing information without
    modification.
    - '!r' for deny read: Prohibiting read access to
     information.
    - 'w' for write: Modifying or adding information
    .
    - '!w' for deny write: prohibiting the right to
    write operation
    - 'x' for execute: Performing operations or
    executing commands.
    - '!x' for deny execute: Disallowing the
    permission to execute a resource or an entity
    - 'c' for create: Creating new entities or
    resources.
    - '!c' for deny create: denying the authority to
     create new  entities or resources
    - 'd' for delete: Removing or deleting entities
    or resources.
    - '!d' for prohibiting delete: Revoking the
    right to delete an entity or a resource.
    '''
    Attributes:
    '''
    cyber_event: CyberEvent - Instance of CyberEvent
     from which this relation inherits context.
    relations: List[Dict[str, str]] - List of
    dictionaries, each representing a specific
    relationship instance:
    - 'subject' (str): Entity initiating the
    interaction.
    - 'subject_type' (str): Type of the subject
    entity (e.g., 'user_attribute').
    - 'relationship' (str): The type of interaction
    (characterized by the codes 'r', 'w', 'x', 'c',
    'd','!r', '!w', '!x', '!d').
    - 'object' (str): Entity that receives the
    action.
    - 'object_type' (str): Type of the object entity
     (e.g., 'object_attribute').
    '''

    cyber_event: CyberEvent
    relations: List[Dict[str, str]]
```

Listing 2. Prompt Pattern for Extracting Access Control Policy Relations

members are formatted as strings, and the model's output is presented as lists of relationships.

The schema definition guidelines for the CyberRelation class, as detailed in listing 2, guide the model by specifying how to determine authorization and prohibition of CRUD operations through relationships extracted between policy entities (lines 7 - 16). Within this context, class members defined under CyberRelation are linked to instances of the CyberEvent class, from which CyberRelation inherits entities relevant to a particular relationship (lines 20 - 26). The outputs delineated by the CyberRelation class encompass the lists of entities sourced from the CyberEvent class and a list of dictionary

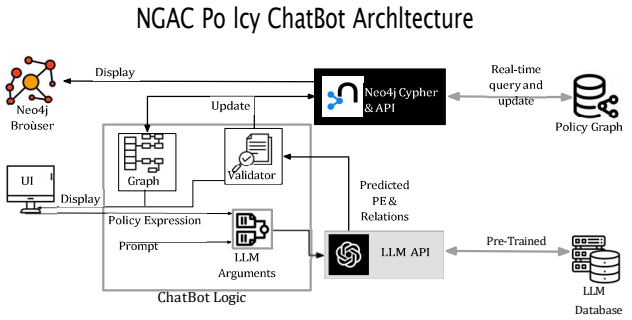

Fig. 1. An Architecture for Translating Natural Language Policy Expressions to NGAC Specification

of relations generated by the CyberRelation class (lines 30 & 29).

The prompt structures showcased in listings 1 and 2 follow a zero-shot approach. This means no illustrative examples of policy expressions and expected results were provided to further guide the language model in identifying entities and relations. This initial prompt pattern will serve as a baseline for comparison, while a scaled version incorporating illustrative examples will be developed in our implementation.

IV. Implementation

In the previous section, we introduced our proposed prompt technique, Code4Policy, to extract access control policy entities and relations. This section details our implementation process for translating these natural language expressions into National Institute of Standards and Technology (NIST) Next Generation Access Control (NGAC) policy specification. Our implementation focuses on utilizing a chatbot's capabilities within a use case scenario, allowing an administrator to input natural language policy expressions into a web-based GUI. These inputs are then translated directly into NGAC policy specification.

The chatbot representation for translating natural language policy expressions into NGAC specification was inspired by the Neo4j GraphAcademy [20], providing a robust foundation for our system's architecture. To implement the graphical user interface (GUI), we leveraged Streamlit, an open-source Python framework tailored for data scientists and AI/ML engineers to craft dynamic data applications with minimal development overhead. Within this interface, users input Natural Language Access Control Policy text, and the system outputs an NGAC specification graph in the form of a Python Dictionary and Neo4j browser graphical dashboard. The extraction of NGAC policy elements and their relationships is handled by utilizing the OpenAI API. Once extracted, the information is translated into a policy graph, which is stored in a Neo4j database and simultaneously displayed on the chatbot interface as shown in figure 1. The following subsections presents a brief overview of NIST Next Generation Access Control,

delves into the Neo4j integration and chatbot logic used in this implementation.

A. NIST Next Generation Access Control

The NIST Next Generation Access Control (NGAC) framework, an instantiation of the Attribute-Based Access Control (ABAC) model, utilizes a Directed Acyclic Graph (DAG) to represent system authorization states. This DAG, referred to as the authorization graph, consists of five node types representing different access control policy entities: Users (U), User Attributes (UA), Objects (O), Object Attributes (OA), and Policy Classes (PC). These entities, collectively known as Policy Elements (PE), are connected through directed edges representing relationships defined by the access control policy expressions. These edges, termed Assignment, Association, and Prohibition, establish the logical connections between PEs, thereby defining the authorization state within the NGAC framework.

Assignment relations provide a hierarchical structure by connecting these entities through conforming ordered pairs, establishing relationships like user to user attribute, object to object attribute, and attribute to policy class. Association relations, represented as weighted edges, define permissions, specifying which actions a user attribute can perform on an object attribute based on the assigned weight. Prohibition relations, also weighted edges, express denials by specifying which actions are forbidden from a user attribute on an object attribute. Finally, obligation relations, unlike other relations, are not part of the authorization graph but are dynamically triggered upon specific events, requiring actions to be performed.

B. Neo4j Graph Database

Neo4j is a graph database that stores and manages data in a natural, connected state. It utilizes a property graph approach, which optimizes both data traversal speed and operational efficiency. Beyond being a core database, Neo4j has evolved into a comprehensive ecosystem encompassing tools, applications, and libraries, enabling seamless integration of graph technologies within various working environments.

Neo4j Cypher is a graph query language designed for modifying and retrieving data from Neo4j graphs. Inspired by SQL, Cypher emphasizes clarity and ease of use, allowing users to focus on what data they want rather than the technical details of retrieval. Users employ Cypher for all CRUD operations on their graphs, making it the primary interface for Neo4j.

Cypher's unique visual aspect allows users to match patterns and relationships through an ASCII-art-inspired syntax, where nodes are represented by rounded brackets and relationships by arrows labelled with squared brackets. This visual approach enables users to effectively draw graph patterns within their data. For example, the cypher command:

$CREATE\ (ua_i{:}admin){-}[{:}\{r,\ w\}]{->}(oa_j{:}backupfiles),$

creates the nodes admin (user attributes) and backup files (object attribute), and an association relation between them with the read (r) and write (w) permissions are the weight of the relation.

C. Chatbot Logic

The chatbot's core functionality revolves around translating natural language specifications into formal access control policies. This process utilizes a carefully crafted pipeline that leverages the power of large language models (LLMs) and integrates it with our own domain-specific knowledge.

The chatbot acts as a wrapper for the OpenAI API, providing a seamless interface for users. When a user enters a policy specification in natural language, the chatbot combines this input with a specially designed in-context learning prompt. This combined input serves as the argument for the OpenAI API call. The prompt is crafted using techniques aimed at guiding the LLM to extract relevant policy entities and their relationships. These entities correspond to the four NIST NGAC relations – Assignments, Associations, Prohibitions, and Obligations.

The API call returns predicted entities and their relationships extracted from the user's input. These predictions are then subjected to a rigorous validation process by an NGAC specification-checker within the chatbot logic. This checker ensures that extracted entities conform to the formal definitions of each relation. For instance, if the predicted relation is an association and the entities forming this relation are a pair of object attributes, the checker would flag a specification violation. Such violations are communicated back to the user through an informative exception message on the chatbot GUI.

Upon successful validation, the predicted policy entities are integrated into the system. This integration occurs on two levels: an in-memory authorization graph is updated using Python modules, and concurrently, the Neo4j database graph is updated using Cypher queries. This simultaneous update ensures consistency and allows for real-time visualization. Users can observe the updated authorization graph directly on the chatbot GUI and can access the Neo4j database graph through the Neo4j browser. This visual feedback provides a clear and intuitive representation of the evolving access control policy, empowering users to understand and manage their security configurations.

TABLE I
SELF-SYNTHESIZED NATURAL LANGUAGE ACCESS CONTROL POLICY DATASET

Dataset	**Relations**			
	Assignment	***Association***	***Prohibition***	***Obligation***
Finance	32	26	11	8
&	**Policy Elements**			
Human	***U***	***UA***	***OA***	***PC***
Resources	12	17	33	2

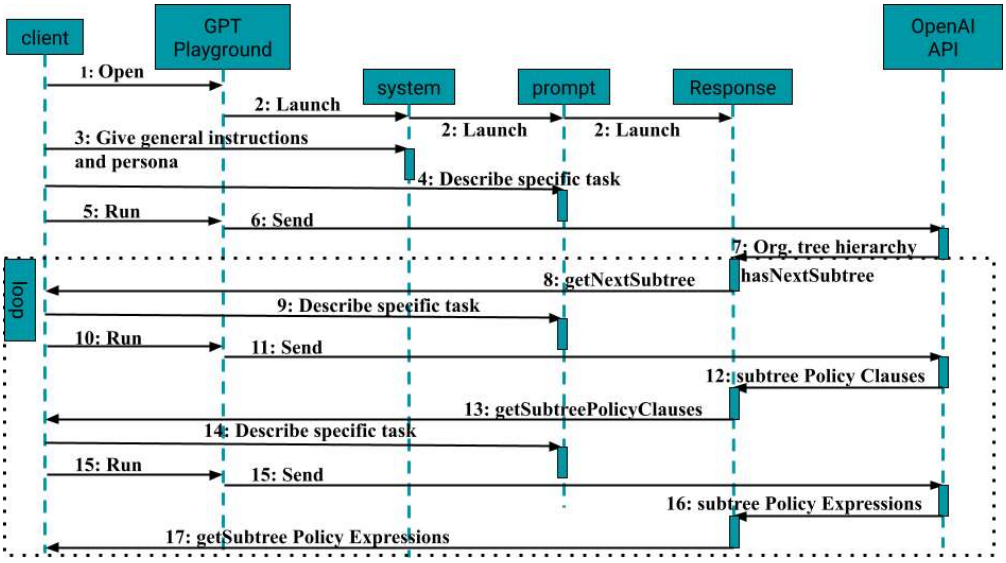

Fig. 2. A Sequence Diagram for a Large Language Model Synthesized Natural Language Policy Expressions

V. Experimental Setup

In the previous section, we discussed the architectural components designed for translating natural language policies into NIST Next Generation Access Control (NGAC) specifications using a chatbot user interface. However, a notable challenge in this domain is the scarcity of publicly available natural language policy datasets. Such datasets are often kept proprietary due to the sensitive nature of authorization requirements. Moreover, existing datasets frequently require significant augmentation to form valid Attribute-Based Access Control policy expressions. Additionally, we compared our method with traditional plain language prompting, which simply converts the CyberEvent and CyberRelation classes into plain language descriptions and definitions.

Previous studies [4], [6] have consistently highlighted these challenges and have resorted to augmenting existing datasets or creating self-synthesized datasets. In this work, we tested our proposed implementation on self-synthesized datasets and further evaluated its performance using datasets synthesized by large language models (LLMs). The following subsections detail our process for generating both self-synthesized and LLM-synthesized datasets, offering insights into our methodology and the specific considerations undertaken to ensure robust experimental evaluation.

A. Self-Synthesized Dataset

The basis of our self-synthesized dataset is an organization chart, closely resembling the hierarchical structure of the NIST authorization graph, which is articulated through as- signment relations. To streamline this corpus, we concentrated on synthesizing access policy expressions specifically for two departments commonly found in any organization: the Legal and Finance Departments.

To construct meaningful policy expressions, we inferred user and resource attributes from job descriptions associated with each hierarchical position within the two departments. These attributes include roles, responsibilities, and resources specific to each job function.

By decoding these attributes and considering the routine activities performed by individuals in given roles, we were able to generate precise policy expressions. These expressions systematically encapsulate the NIST relations - Assignment, Association, Prohibition, Permission, and Obligation.

A precise breakdown of policy entities and relations derived from 45 policy expressions in our synthesized dataset is presented in Table I. This table illustrates the distribution and intricacies of each relation type, providing an empirical foundation for further exploration into synthesizing policy expressions using Large Language Models (LLMs).

B. LLM-Synthesized Dataset

The primary advantage of using an automated approach with LLMs is twofold. Firstly, it generates policy expressions that are free from the biases introduced by human authorship. By utilizing the expansive datasets on which LLMs are trained, we ensure that the synthesized policies contain natural context and relevant policy entities of interest. Secondly, it obviates the necessity of recruiting domain experts for manual creation of an extensive dataset of natural language policy expressions, thereby saving both time and resources.

For this study, we have utilized the ChatGPT Playground to generate policy expressions. Notably, similar outcomes were observed when employing Google Gemini via Vertex AI Studio, underscoring the robustness and reliability of LLMs across different platforms. The sequence diagram in Figure 2 encapsulates the comprehensive process for generating this LLM-synthesized natural language policy expressions dataset.

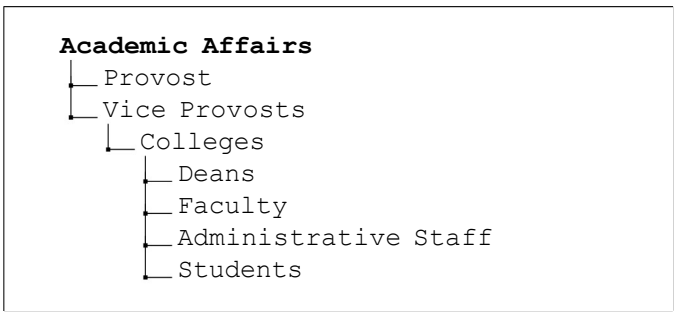

Fig. 3. Academic Affairs: A Sub-tree of LLM Generated University of Texas at San Antonio, Texas, Employee Tree Hierarchy

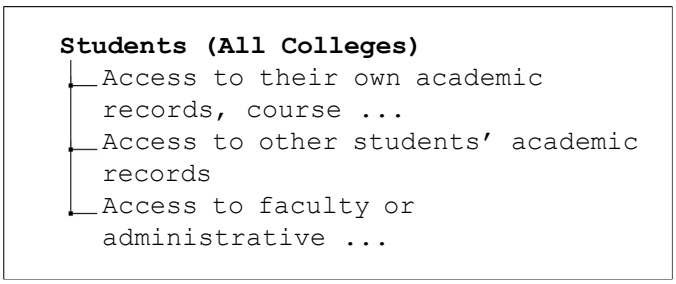

Fig. 4. LLM-Synthesized Access Control Policy Fragments For A User Attribute

In the sequence diagram, interactions amongst three primary objects—client, GPT Playground, and OpenAI API—culminate in the creation of the LLM-synthesized dataset. A User (client) commence by accessing the web-based GPT Playground interface. Upon accessing it, the Playground provides three text fields: system prompt and user prompt fields for input instructions, and an assistant message field displaying the LLM-generated response. Through these editable prompts, clients can precisely guide the output of the large language model API call to obtain desired policy expressions.

To illustrate the sequential process of using GPT Playground to generate natural language access control policy sentences, we outline the following steps:

We first configured the large language model (LLM) with a system prompt to establish its role and context. The provided text was: *You are a Senior database and security administrator for the University of Texas at San Antonio (UTSA). You will assist in the process of writing access control policy expressions for various units and departments the Institution*

The initial user prompt tasked the LLM with creating an organizational hierarchy for employees across all the departments at UTSA. The text of the user prompt was: *Create a tree hierarchy of the University of Texas at San Antonio employees across all departments/units.* In response, OpenAI produced a simplified organizational tree hierarchy divided into major functional units. A subtree for the Academic Affairs Unit demonstrates this structure shown in figure 3.

Next, we used iterative inputs for each subtree, directing the LLM to generate access control policies. The user prompt included instructions such as: *For each employee position in the given unit tree hierarchy, write access control policy in terms of allowed and denied access to resources.* The LLM responded with policy clauses formatted under headings for each employee position, listing specific allowed and denied resources.

In the final step, we requested that the LLM generate coherent sentences integrating both policy expression fragments and employee positions. The user prompt was structured to elicit this output: *For each policy expression fragment and corresponding employee position, create a sentence.*

The resulting output transformed technical policy fragments into natural language statements, rendering them more accessible for human review and implementation. This methodical approach ensures that access control policies are precisely articulated while leveraging advanced language model capabilities for clarity and cohesion.

Some may suggest that instead of using a series of steps, we could combine everything into one comprehensive prompt to achieve similar outcomes. However, it has been observed that this approach significantly reduces the number of policy expressions generated, and those that are produced tend to be excessively generalized. Rather than utilizing a programmatic method to execute the full sequence of LLM-Synthesized access control policies directly via the OpenAI developer interface, we opted to use the GPT Playground. This approach enables a user (client) to validate the LLM's response at each step in the sequence for creating synthesized access control policies. Even with meticulously designed prompt messages, the LLM can occasionally produce suboptimal results. By involving a user mediator in this process, we can ensure that our evaluation of prompt techniques for extracting policy entities and relationships isn't compromised by faulty data.

C. NIST NGAC Prompt Constructs

In Section V, we discussed adaptable, generic prompts for CoT and PoT approaches, applicable to any security model. The prompt constructs for these approaches include specific definitions to identify NGAC policy elements and extract their relationships. For example, the prompt pattern detailed in

Students (All Colleges) cannot access other students' academic records.
Faculty or administrative staff personal records or internal decision-making documents cannot be accessed by Students (All Colleges).
Students (All Colleges) can access their own academic records, course materials, schedules, and grading information.

Fig. 5. LLM-Synthesized Access Control Policy Expressions After Integrating User Attribute and Policy Fragments

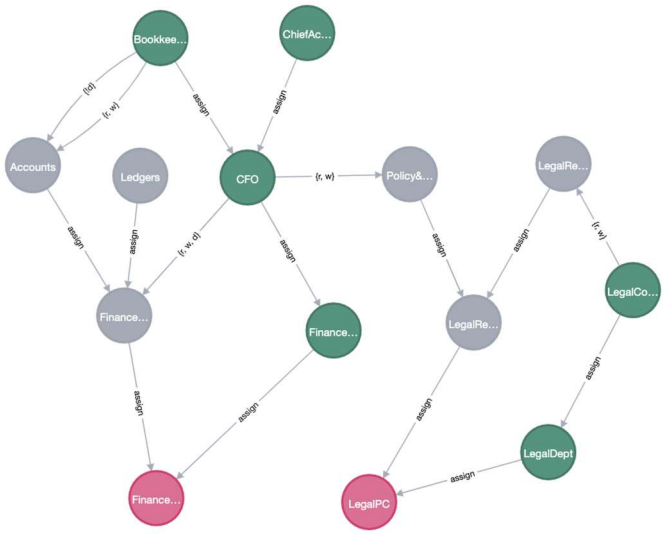

Fig. 6. Graphical Representation of Self-Synthesized Natural Language Access Control Policy Translated to NIST Next Generation Access Control Specification

Section V is sufficient to extract an association relationship and its entities, such as user and object/resource attributes.

Moreover, the prompt pattern was adapted to define prohibitions explicitly: a prohibited or disallowed action between two entities (user and object attributes), instructing the LLM to prefix the prohibited relation with an exclamation mark (!). Consequently, prohibitions like reading, writing, creating, deleting, and executing are represented as !r, !w, !c, !d, and !x, respectively. An obligation relation is defined as a conditional clause that triggers an event response, consisting of an event pattern followed by an independent clause. These definitions help the LLM accurately extract entities and relationships, which Neo4j then uses to translate natural language access control policy expressions into NGAC specifications.

VI. Results

An implementation of the proposed architecture to translate a self-synthesized dataset into access control policies adhering to the NIST Next Generation Access Control (NGAC) specification has been visualized using the neo4j browser, as displayed in Figure 6. This visualization offers a clear graphical representation of the authorization state and structured policy expressions, stored within a graph database via neo4j. The two lower nodes in the graph represent the policy classes for the Legal and Finance departments, with assignment relations from user attributes depicted by greenish cyan nodes and gray nodes for object attributes. The user nodes are the upper shade of pink assigned to user attributes.

We conducted an evaluation focused on extracting entities—user attributes and object attributes—and relations, including association, prohibition, and obligation, from the LLM-synthesized dataset using our proposed approach. It's notable that the assignment relation was deliberately excluded from this evaluation, as it is inherently established through the tree hierarchy while synthesizing the dataset.

TABLE II
LLM-Synthesized Natural Language Access Control Policy Dataset for the Experiment

Dataset	NLACP	Attributes		Associations	Prohibitions	Obligations
		User	Object			
BoA, NA	220	70	359	127	46	47
NHS	323	90	369	196	104	23
UTSA	374	76	579	210	134	30
Total	917	236	1,307	533	284	100

TABLE III
Results of the Proposed (Scaled) Code-of-Thought Policy Entities and Relations Extraction Approach on the LLM-Synthesized Natural Language Access Control Policy Dataset

Dataset	Model	Relations								
		Association			Prohibition			Obligatior		
		P	R	F1	P	R	F1	P	R	F1
Bank of America, NA	Baseline	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	code4policy	1.00	0.98	0.99	1.00	1.00	1.00	1.00	0.99	0.99
		Attributes								
		User			Object					
	Baseline									
	code4policy	1.00		1.00		1.00		1.00		1.00
		Relations								
		Association			Prohibition			Obligation		
		P	R	F1	P	R	F1	P	R	F1
National Health Service	Basline	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	code4policy	0.98	0.98	0.98	0.99	0.99	0.99	1.00	1.00	1.00
		Attributes								
		User			Object					
		P		R	F1	P		R		F1
	Baseline	0.88		0.86	0.87	0.32		0.27		0.29
	code4policy	1.00		1.00	1.00	0.99		0.99		0.99
		Relations								
		Association			Prohibition			Obligation		
		P	R	F1	P	R	F1	P	R	F1
University of Texas at San Antonio	Baseline	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	code4policy	0.99	0.98	0.98	1.00	1.00	1.00	1.00	1.00	1.00
		Attributes								
		User			Object					
		P		R	F1	P		R		F1
	Baseline	0.94		0.92	0.92	0.34		0.28		0.30
	code4policy	1.00		0.99	0.99	0.99		0.99		0.99

P, R, F1 represent precision, recall and F1-score, respectively

In evaluating the zero-shot baseline prompt for translating natural language specifications into access control policies, our observations yielded notable contrasts in performance metrics across different categories of entities and relations. Specifically, the prompt demonstrated commendable proficiency in extracting user attribute entities, with an accuracy rate averaging over 90% as shown in Table III . This high success rate reflects the model's robust capability in identifying user-related information within policy statements without prior training or

contextual fine-tuning.

However, the model's performance markedly declined in the extraction of object attribute entities and the delineation of the three critical relations (Associations, prohibitions, and obligations). Analysis revealed that this discrepancy largely arises from the inherent complexity and structure of object attribute expressions within policy language.

A significant portion of these expressions includes qualifiers and descriptors that provide contextual nuances to the object attributes. For instance, a natural language policy might describe an object with elaborate compound phrases such as "Financial records unrelated to medical treatments" where "unrelated to medical treatments" functions as a descriptor and "Financial records" as the core object attribute. The zero-shot model, however, frequently misinterprets these compound phrases, often extracting unexpected text fragments that amalgamate both qualifiers and descriptors as singular object attributes. This leads to a proliferation of inaccurate and semantically inappropriate object entities in the output.

Furthermore, the complexities inherent in relational constructs within natural language policies exacerbated the model's challenges. Relations among entities are dependent on subtle syntactic and contextual indicators. The zero-shot prompt failed to make any correct predictions for these three relations. The model not only struggled to reliably identify and categorize these relations but also produced inconsistent and incorrect extractions. As a result, this significantly compromised the overall reliability of the zero-shot approach when applied to the task of translating comprehensive access control policies.

The zero-shot performance results were extremely poor, but when employing the scaled version of the model (code4policy) with a few-shot learning approach that included three illustrative examples of prohibition relations, we observed significant improvements across all three types of policy relations. This nearly flawless performance of the scaled model can likely be attributed to the fact that the datasets were synthesized by a large language model (LLM), while the in-context training examples were drawn from our self-synthesized dataset.

VII. Limitations

While this study represents a significant step towards developing a robust chatbot for translating natural language access control policies through the use of large language models (LLMs), several limitations must be acknowledged.

Firstly, the architecture proposed in this work supports only single-turn tasks. This means it focuses on translating one or more natural language access control policy expressions in a single interaction to the NIST Next Generation Access Control (NGAC) specification. Single-turn interactions are less dynamic and do not capture the iterative nature of real-world policy management scenarios. Future iterations need to exploit the multi-turn dialogue capabilities of platforms like the OpenAI API. This would enable more complex interactions such as refining initial translations, handling follow-up questions, and managing iterative updates based on historical context.

Another limitation is the current approach's restriction to translating policies with distinct relations among policy elements. Attribute-Based Access Control (ABAC) is highly expressive and often involves complex policy expressions that embed multiple types of relations within a sentence, such as assignments and associations among policy elements. Although our proposed method offers a foundation for trans- lating these relationships, it does not fully accommodate the interwoven and multifaceted relations described in more complex access control policies. This constraint could limit its applicability in more sophisticated or nuanced scenarios.

Moreover, there is a necessity to enhance the mechanism for real-time policy queries and updates. The future work aims to leverage chat history for maintaining context across multiple interactions, which would facilitate real-time adjust- ments and queries concerning policy statuses. However, the current system lacks this dynamic adaptability, potentially constraining its effectiveness in environments where policy conditions frequently change or need real-time verification.

Finally, another important limitation is the comprehensiveness of failure handling mechanisms. When policy translations fail or produce ambiguous results, there is currently no robust system to manage these exceptions autonomously. Future work would need to incorporate algorithms that not only identify failed translations but also attempt automated corrections or seek clarifications from users to ensure higher accuracy and reliability.

VIII. Conclusion

This paper investigates the application of Large Language Models (LLMs) in translating Natural Language Access Control Policies (NLACPs) into formal access control specifications. The inherent ambiguities and variability of natural language have historically posed significant challenges in accurately extracting ACP entities.

Through our study, we make notable contributions to this domain by pioneering an advanced automated method leveraging LLMs. We implemented state-of-the-art prompting techniques tailored to effectively parse policy texts for en- tity and relationship extraction. Furthermore, we developed a synthesized access control corpus generated through LLMs to support our approach. Our empirical results reveal that the code4policy (scaled) models achieve remarkable precision in entity prediction within policy expressions—each attaining an accuracy of 99.9%. However, differences emerge when evaluating relationships between entities. Specifically, the baseline model underperformed in zero-shot settings for association, prohibition, and obligation due to insufficient context within prompts.

Remarkably, enhancing the baseline model with few-shot learning by incorporating illustrative examples substantially improved relationship extraction accuracy across all pol- icy types. This improvement underscores the importance of context-rich prompts when dealing with complex natural language expressions, particularly those related to prohibitions

that exhibit conflicting segments suggesting both access denial and conditional approvals.

Our findings highlight the efficacy of LLMs in navigating the intricacies of NLACPs and present a significant step toward fully automating the translation of natural language policies into formal access control models. Future research should focus on refining contextual understanding within these models to further enhance relationship extraction accuracy and address lingering issues of ambiguity in complex policy expressions.

Through continued advancements in this field, we aim to equip organizations with robust tools for more efficient governance and regulatory compliance in managing access control policies articulated in everyday language.

REFERENCES

[1] R. C. Turner, "Proposed model for natural language abac authoring," in *Proceedings of the 2nd ACM Workshop on Attribute-Based Access Control*, ser. ABAC '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 61–72. [Online]. Available: https://doi.org/10.1145/3041048.3041054

[2] M. Narouei and H. Takabi, "Automatic top-down role engineering framework using natural language processing techniques," in *IFIP International Conference on Information Security Theory and Practice*. Springer, 2015, pp. 137–152.

[3] ——, "Towards an automatic top-down role engineering approach using natural language processing techniques," in *Proceedings of the 20th ACM Symposium on Access Control Models and Technologies*, ser. SACMAT '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 157–160. [Online]. Available: https://doi-org.libweb.lib.utsa.edu/10.1145/2752952.2752958

[4] M. Abdelgawad, I. Ray, S. Alqurashi, V. Venkatesha, and H. Shirazi, "Synthesizing and analyzing attribute-based access control model generated from natural language policy statements," in *Proceedings of the 28th ACM Symposium on Access Control Models and Technologies*, 2023, pp. 91–98.

[5] M. Narouei, H. Khanpour, H. Takabi, N. Parde, and R. Nielsen, "Towards a top-down policy engineering framework for attribute-based access control," in *proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies*, 2017, pp. 103–114.

[6] M. Alohaly, H. Takabi, and E. Blanco, "A deep learning approach for extracting attributes of abac policies," in *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*, 2018, pp. 137–148.

[7] J. Slankas and L. Williams, "Access control policy identification and extraction from project documentation," *Science*, vol. 2, no. 3, pp. 145–159, 2013.

[8] J. Slankas, X. Xiao, L. Williams, and T. Xie, "Relation extraction for inferring access control rules from natural language artifacts," in *Proceedings of the 30th annual computer security applications conference*, 2014, pp. 366–375.

[9] M. Narouei, H. Khanpour, and H. Takabi, "Identification of access control policy sentences from natural language policy documents," in *Data and Applications Security and Privacy XXXI: 31st Annual IFIP WG 11.3 Conference, DBSec 2017, Philadelphia, PA, USA, July 19-21, 2017, Proceedings 31*. Springer, 2017, pp. 82–100.

[10] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[11] O. Sainz, I. Garc´ıa-Ferrero, R. Agerri, O. L. de Lacalle, G. Rigau, and E. Agirre, "Gollie: Annotation guidelines improve zero-shot information-extraction," in *The Twelfth International Conference on Learning Representations*.

[12] J. Gao, H. Zhao, W. Wang, C. Yu, and R. Xu, "Eventrl: Enhancing event extraction with outcome supervision for large language models," *arXiv preprint arXiv:2402.11430*, 2024.

[13] X. Wang, S. Li, and H. Ji, "Code4struct: Code generation for few-shot event structure prediction," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023, pp. 3640–3663.

[14] I.-H. Hsu, K.-H. Huang, E. Boschee, S. Miller, P. Natarajan, K.-W. Chang, and N. Peng, "Degree: A data-efficient generation-based event extraction model," in *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2022, pp. 1890–1908.

[15] X. Xu, Y. Zhu, X. Wang, and N. Zhang, "How to unleash the power of large language models for few-shot relation extraction?" in *Proceedings of The Fourth Workshop on Simple and Efficient Natural Language Processing (SustaiNLP)*, N. Sadat Moosavi, I. Gurevych, Y. Hou, G. Kim, Y. J. Kim, T. Schuster, and A. Agrawal, Eds. Toronto, Canada (Hybrid): Association for Computational Linguistics, Jul. 2023, pp. 190–200. [Online]. Available: https://aclanthology.org/2023.sustainlp- 1.13

[16] Z. Wan, F. Cheng, Z. Mao, Q. Liu, H. Song, J. Li, and S. Kurohashi, "Gpt-re: In-context learning for relation extraction using large language models," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023, pp. 3534–3547.

[17] S. Wadhwa, S. Amir, and B. Wallace, "Revisiting relation extraction in the era of large language models," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, A. Rogers, J. Boyd- Graber, and N. Okazaki, Eds. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 15 566–15 589. [Online]. Available: https://aclanthology.org/2023.acl-long.868

[18] "Natural language programming: Styles, strategies, and contrasts," *IBM Systems Journal*, vol. 20, no. 2, pp. 184–215, 1981.

[19] Z. Bi, J. Chen, Y. Jiang, F. Xiong, W. Guo, H. Chen, and N. Zhang, "Codekgc: Code language model for generative knowledge graph construction," *ACM Transactions on Asian and Low-Resource Language Information Processing*, vol. 23, no. 3, pp. 1–16, 2024.

[20] Neo4j GraphAcademy. (2024, Jul.) Neo4j GraphAcademy - Build a Neo4j-backed Chatbot using Python. Accessed jul. 26, 2024. [Online]. Available: https://graphacademy.neo4j.com/courses/llm-chatbot-python/