LLM-Based Fine-Grained ABAC Policy Generation

Khang Mai[1], Nakul Ghate[2], Jongmin Lee[1] and Razvan Beuran[1]
[1]*Japan Advanced Institute of Science and Technology (JAIST), Ishikawa, Japan*
[2]*NEC Corporation, Tokyo, Japan*

Keywords: Access Control, Fine-Grained Policy Generation, Large Language Models, ABAC, ICS, Security Guideline.

Abstract: The central practice in the development of Attribute-Based Access Control (ABAC) is policy generation, for which supervised machine-learning approaches can achieve state-of-the-art performance. However, the scarcity of training data poses challenges for supervised solutions, limiting their practical application. Recently, large language models (LLMs) have demonstrated extraordinary proficiency in various language processing tasks, offering the potential for policy mining in scenarios with only a few training examples. This paper presents an LLM-based generation of fine-grained ABAC policies. The approach utilizes multiple LLMs in a mixture-of-agents mechanism to consider the ABAC scenario from diverse perspectives. Multi-turn interaction and retrieval augmented generation are combined to generate and prepare adequate LLM prompting context. In the evaluation, we conduct experiments within an Industrial Control System (ICS) network, ensuring that the ABAC policies align with specific security guidelines. We explore the feasibility of utilizing policies generated by LLMs directly in the access control decision-making process. By leveraging ground truth data, we implement an optimization module that refines the priority values of these policies, ultimately achieving an impressive F1 score of 0.994, showing that LLMs have the potential to generate fine-grained ABAC policies for real IT networks.

1 INTRODUCTION

With its flexibility and granularity, Attribute-Based Access Control (ABAC) is an access control model suitable for complex and dynamic IT environments. A typical ABAC implementation begins with defining system attributes and access control policies. Many ABAC research studies aim to automate these labor-intensive and error-prone tasks with computer-based models. Training computer models in a supervised approach for policy generation from text involves annotating thousands of sentences, which is challenging. Furthermore, adopting an already-trained model to new systems commonly requires re-training with new data, which is not always available

The advent of large language models (LLMs), such as GPT-4, holds promise for addressing data scarcity. Leveraging their exceptional language comprehension and generalization abilities, LLMs can provide innovative solutions for unforeseen tasks with minimal examples. We propose an LLM-based solution for generating fine-grained ABAC policies for IT networks. Our approach mitigates challenges associated with LLMs, including context insufficiency and length limit. Multiple LLMs are utilized to capitalize on their diverse strengths. We employ an automated multi-turn prompt construction method to systematically integrate necessary information. Furthermore, we implement a flipped interaction pattern, allowing LLMs to request additional data. This method effectively utilizes retrieval-augmented generation (RAG) to gather required inputs. The synthesized policies undergo validation before being used in decision-making.

To showcase the effectiveness of our approach, we collaborated with a team of cybersecurity industrial experts to design a typical ICS network as a running example. The National Institute of Standards and Technology (NIST) security document, SP 800-82 r2, is the main guideline for LLMs to follow when designing ABAC policies. We evaluate the generated policies and discuss different methods to rectify the priority values assigned by LLMs to ensure proper decision-making with generated policies.

In the remainder of this paper, we first present the background and related work in Section 2. Section 3 describes the proposed approach in detail. Section 4 discusses the approach's experimental evaluation regarding a typical ICS network. Finally, we conclude the paper with a conclusion and references.

Mai, K., Ghate, N., Lee, J. and Beuran, R.
LLM-Based Fine-Grained ABAC Policy Generation.

In *Proceedings of the 11th International Conference on Information Systems Security and Privacy (ICISSP 2025) - Volume 2*, pages 204-212
ISBN: 978-989-758-735-1; ISSN: 2184-4356

2 BACKGROUND AND RELATED WORK

This section presents an overview of the background literature and relevant studies for this study.

2.1 Machine Learning for Attribute-Based Access Control Policy Generation

Attribute-Based Access Control (ABAC) is a flexible approach that involves granting or denying users access to resources based on the evaluation of policies against specific attributes. An ABAC policy is a statement that combines attributes to set restrictions and conditions for access control decision-making.

Recently, Machine Learning (ML)–a branch of artificial intelligence (AI)–has been widely recognized as an advanced approach in ABAC, especially in policy mining and generation. Policy can be generated from various data sources. For example, Narouei et al. (Narouei et al., 2017) use real-world documents to develop a dataset of 2660 annotated sentences for policy generation. Heaps et al. (Heaps et al., 2021) extract access control information from user stories written by software developers, which requires the identification of actors, data objects, and operations. Access logs are also a great resource for policy mining, as in (Cotrini et al., 2018) and (Alohaly et al., 2019).

After being generated, policies are optimized to minimize the unnecessary complexity of access control policies. For example, policies are clustered decision effects (Ait El Hadj et al., 2017) to reduce the redundancy. The recent work by Mitani et al. (Mitani et al., 2023) introduces QI-ABAC, which leverages the intentions driving the policy manager's decision-making to enhance neural network-based policy refinement using a limited set of initial policies. While the initial findings are encouraging, there are notable challenges associated with generating initial policies, intentions, and training data in real-world settings.

2.2 Large Language Models

Large language models (LLMs) are transformer-based models (Vaswani et al., 2017) with billions of parameters. With remarkable language understanding capabilities, they have been recruited to replace the role of experts in various domains.

Recently, LLMs have been applied in various cybersecurity applications, especially security control (Ahmed et al., 2024; Tarek et al., 2024). For example, SoCureLLM (Tarek et al., 2024) is a framework designed for system-on-chips (SoCs) security verification and policy generation. The policies generated by SoCureLLM require manual fidelity checking performed by security experts. Despite this, SoCureLLM's coarse-grained security policies are presented in natural language form and potentially serve as security guidelines for relevant SoC applications.

This paper proposes an LLM-based fine-grained ABAC policy generation. We address challenges associated with working with LLMs, such as context length limit and context sufficiency.

2.3 ICS and Related Security Guidelines

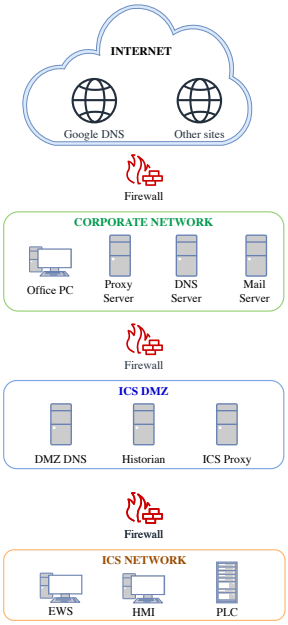


Figure 1: An example of a typical ICS network with main segments and devices.

Industrial Control Systems (ICS) are automated control systems that manage industrial and critical infrastructure, such as manufacturing. In addition to standard devices found in a typical computer network, ICS includes specialized components (e.g., Programmable Logic Controllers (PLCs), Human-Machine Interfaces (HMIs)), and communication protocols like Modbus, DNP3. To secure the ICS network, NIST developed a security guideline, Special Publication (SP) 800-82 r2 (Stouffer et al., 2015), for protecting ICS environments from cybersecurity threats. In this paper, we consider an ICS network (as shown in Figure 1) as a running example for our approach that follows the NIST 800-82 r2 guidelines.

Table 1: Main data elements in the knowledge base for ICS network and their relationship with specific tasks.

Name	Data Type	Attribute Refinement	Policy Generation
Introduction of ICS concept	text	✓	✓
A brief introduction of NIST SP 800-82 r2	text	✓	✓
ABAC's related concepts	text	✓	✓
NIST SP 800-82 r2 specific guidelines	list (JSON file)		✓
The ICS network overview	text	✓	✓
System information (devices, protocols, etc.)	dataframe (CSV files)	✓	✓
System initial attributes	list (JSON file)	✓	
Task description and template, rules	text	✓	✓
Task few-shot examples (for few-shot learning)	text	✓	✓
Database schema	text	✓	
Refined attributes	list (JSON file)		✓

3 PROPOSED APPROACH

This section describes the proposed methodology for LLM-based fine-grained ABAC policy generation as shown in Figure 2, with five main components:

1. **Knowledge Base Construction:** This component constructs a knowledge base from initial data to support the following tasks.
2. **Prompt Construction:** This component aims to create a task-specific prompt to provide LLM with sufficient context.
3. **Attribute Refinement:** This component aims to refine attributes from a list of initial attributes.
4. **Policy Generation:** This component utilizes multiple LLMs as generators to generate policies aligned with security guidelines.
5. **Priority Optimization:** This optional component aims to optimize the policy priority values generated by LLMs for policy conflict resolution.

3.1 Knowledge Base Construction

To provide LLMs with the necessary context to solve complex tasks, we set up a knowledge base for storing and managing data for subsequent analysis. Table 1 presents an example knowledge base for policy generation for the ICS network. Each data element contains three essential pieces of information: name, description, and data content. Different functions are implemented to transform the initial data (provided by the user) to an LLM-friendly format with useful information for subsequent tasks, including:

1. We split the security guideline document (e.g., NIST SP 800-82 r2 document) into sub-sections and paragraphs to avoid the error of context length limits and reduce the complexity of the task.
2. System information is transformed into an SQL database and then a database schema. The database schema is inputted into the prompt instead of the system information (see section 3.3).
3. A list of example access requests is generated from the system information. An access request contains a sequence of attributes and their specific values, whose usage can be seen in Section 3.4.
4. We create embedding vectors for data names and descriptions to support RAG.

3.2 Prompt Construction

As shown in the blue dashed box of Figure 2, the prompt construction requires a task configuration and interacts with the knowledge base and LLM to construct a task-specific structured prompt in a multi-turn format. As shown in Figure 3, the prompt starts with a system message to utilize the persona pattern (White et al., 2023), asking the LLM to act as an ICS and ABAC expert. The prompt body contains various chat turns; each can be a knowledge-recalling prompt (to retrieve commonly encountered types of knowledge, e.g., ICS, ABAC) or a knowledge-injecting prompt (to inject novel knowledge). The prompt ends with a task-triggering message notifying the LLM to start its work. Additionally, a specialized prompt (the flipped interaction pattern (White et al., 2023)) enables the LLM to actively request new knowledge. Via similarity search, we retrieve this knowledge from the database in a fashion similar to RAG for integrating into the chat session appropriately.

3.3 Attribute Refinement

The example workflow of attribute refinement is shown in the red dashed box of Figure 2. The required data for this task (see Table 1) is encapsulated inside a task-specific configuration and sent to prompt construction. We start with a basic list of attributes containing minimal attribute information such as name and description. We then employ an LLM to refine the attributes using system data, such as system infrastructure and other information. However, we avoid inputting the detailed information of the system to circumvent the context length limit by using the database schema (mentioned in Section 3.1). This task requires the LLMs to refine attributes to include more helpful information and SQL SELECT commands (as shown

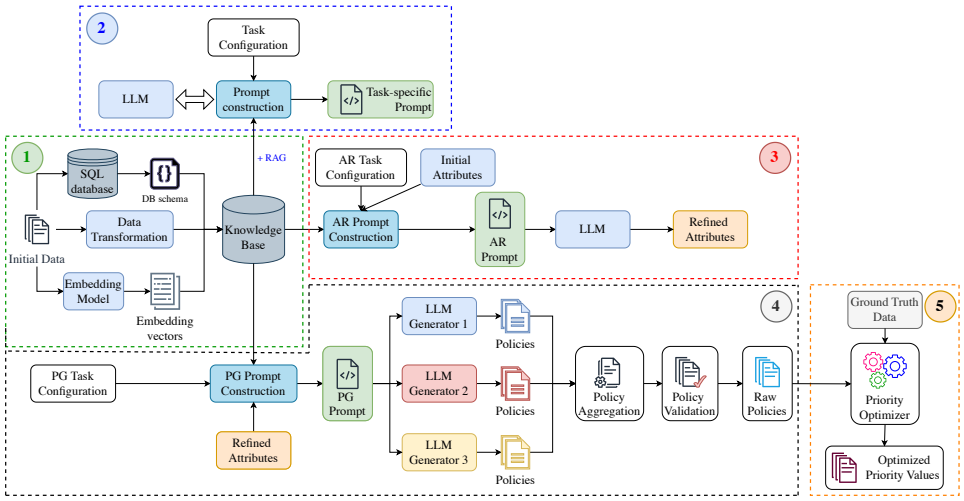


Figure 2: System architecture with main components: 1. Knowledge Base Construction, 2. Prompt Construction, 3. Attribute Refinement (AR), 4. Policy Generation (PG) and 5. Priority Optimization.

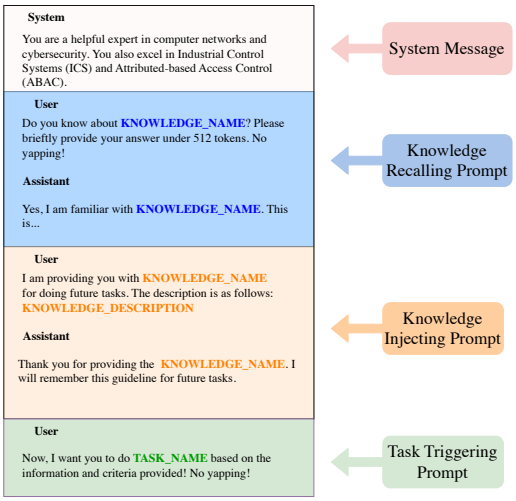


Figure 3: Main structure of a chat session.

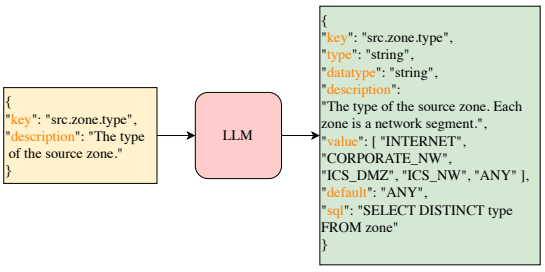


Figure 4: The input and output for the attribute refinement.

in Figure 4). We then execute the SQL command to extract the attribute's valid values from the SQL database. The output will be a list of refined attributes ready for use in fine-grained policy generation.

3.4 Policy Generation

A workflow for policy generation can be seen in the black dashed box of Figure 2. The primary information for this policy generation process is the security guideline and the system attributes, which have been refined using attribute refinement. Similar to attribute refinement, all the required data for this task (see Table 1) is first encapsulated into a single task-specific configuration before sending to prompt construction to create a suitable prompt.

Since this task is complicated, we craft a list of generation rules to guide LLMs on generating policies in an expected format (e.g., Python function with docstring). LLMs are also required to generate the priority values for the policies based on the guideline and their reasoning. This is a proactive step since we aim to use a priority-based combining algorithm for policy conflict resolution. The example policy generated to follow the mentioned criteria and rules can be seen in Figure 5. As depicted in this figure, the policies are structured as Python functions where each condition in the policy specifically denies or allows access. The policy function returns "None" as default for cases where it does not have enough information to make the decisions. These policies can be directly imported into a Python runtime environment to facilitate access control decision-making.

Mixture of Agents. The mixture-of-agents (Wang et al., 2024) approach allows multiple LLMs to work collaboratively to solve a complicated task. Inspired by this paradigm, we develop a similar solution to incorporate multiple LLMs in the policy generation which is demonstrated in Algorithm 1. The **prompt-**

```
def SP800_82_51_Rule_No0_1(effect, policy, dictdat):
    """
    **Fact**: Network segmentation and segregation is one of the
most effective architectural concepts to protect ICS.
    **Reasoning**: This policy function checks if the source and
destination zones are different and if the service is essential for
cross-domain communication. If not, deny access.
    **Condition 1**: If `service.essential` is False and the source
and destination zones are different, deny access.
    **Default**: None
    **External Knowledge**: None
    **New Attribute Requirement**: None
    **Attributes**: `service.essential`, `src.zone.type`,
`dst.zone.type`
    **Potential Error**: None
    **Priority Value**: 70
    """
    if (dictdat.get("service.essential") == 0 and
        dictdat.get("src.zone.type") != dictdat.get("dst.zone.type")):
        return "deny"
    return None
```

Figure 5: An example of the generated policy.

construction() is a function that constructs a prompt tailored for the policy generation task (see Section 3.2). Each LLM generator is then prompted to generate fine-grained policies. When the generators finish their work, the results are gathered and passed on to a subsequent aggregation process.

Algorithm 1: Policy Generation.

Data: Task Configuration T
Result: A list of generated policies
$G \leftarrow$ list of LLM generators;
$t \leftarrow$ threshold value;
$F \leftarrow \{\}$ /*empty frequency dictionary*/ ;
$P \leftarrow []$ /*empty policy list*/;
$S \leftarrow prompt\_construction(T)$;
/*start retrieving responses from generators*/
foreach *generator in G* **do**
    Prompt the generator with S;
    Extract the policies from the response;
    Deduplicate the policies;
    /*start counting the votes for policies*/
    foreach *policy p in generated policies* **do**
        if *p is in F* **then**
            Increase count for p by 1;
        else
            Add p into F with count 1;
        end
    end
end
/*keep policies with a high number of votes*/
foreach *policy p in F* **do**
    if $(p.count/len(G)) \geq t$ **then**
        Add p to P;
    end
end
Return P;

Policy Aggregation. This module is to combine policies generated from different LLM generators into a single list. In the original mixture-of-agents approach (Wang et al., 2024), the aggregator, which takes the responses from other LLM generators for synthesizing, is also an LLM. In our approach, we use a deterministic aggregator with a major voting mechanism (lines 12 to 25 of Algorithm 1) to determine the output policies from the generator's responses. We maintain a frequency dictionary to count the votes of generators with respect to a specific policy. When a new policy appears, its frequency is assigned to one and will increase by one each time a generator produces a similar policy. We use an equivalent operator to deduplicate policies and check policy similarity (in line 10 and line 13 of the algorithm, respectively).

Policy Equivalence. Because the policies generated are in Python function format, we first check the similarity in their abstract syntax tree (AST) using the Python library named code_diff (Smith and Johnson, 2024). Additionally, we compare the decisions made by the two policies against the list of potential access requests. Two policies are considered equivalent if their decisions (e.g., allow, deny, None) are identical for all access requests in the list. This behavior is enforced by setting the threshold value of 1.

Policy Validation. As shown in Figure 5, each output policy is a Python function with a docstring. The docstring provides us with useful information to validate the policy. We implement various functions to validate these generated policies in a deterministic and non-deterministic manner, as follows.

1. Deterministic validation: We employ the Bandit (OpenStack Security Group (OSSG), 2024) library to identify prevalent security vulnerabilities in the generated code.
2. Non-deterministic validation: We use LLMs to examine the correlation among components of each policy to detect inconsistencies and errors.

3.5 Priority Optimization

Using LLM-generated fine-grained ABAC policies for access control may lead to policy conflicts. To address these conflicts, we employ a priority-based combining algorithm for its explainability, flexibility, and scalability. The workflow for priority optimization is illustrated in the orange dashed box of Figure 2. We assume access to ground truth data to optimize the priority values of the generated policies, formu-

lated as a continuous mathematical problem to identify the optimal solution among various alternatives.

Solution Space. We define the solution space as $S \subseteq \mathbb{R}^n$. Each solution, formulated as $\mathbf{s} = (s_1, s_2, \ldots, s_n)$ where $0 \leq s_i \leq 100$ for $i = 1, 2, \ldots, n$, is a list of n priority values where each value is a real number from 0 to 100. Here s_i is the priority value for the policy p_i in a list of policies $\mathbf{p} = (p_1, p_2, \ldots, p_n)$.

Objective Function. The objective function of priority optimization is denoted as $f : S \to \mathbb{R}$ in which the function $f(s)$ will output a real value when inputted with solution $s \in S$. The objective function $f(s)$ calculates the F1 score by comparing the output access control decisions using the policies with priority values (in a priority-based combining algorithm) and ground truth decisions (see Section 4.1).

We conceptualize the objective function as a black-box function, indicating that although we can observe or measure the output for a given input, the underlying relationship between inputs and outputs remains unclear or too complicated to model directly. In the context of black-box function optimization problems, a variety of established gradient-free approaches exist that can effectively identify optimal solutions, which are well documented in the literature and are readily implemented within the Python framework known as Nevergrad (Rapin and Teytaud, 2018). This paper tests the feasibility of priority optimization using the optimizers implemented in Nevergrad, such as CMA, DE, TBPSA, etc.

Priority Optimization Problem. Using the above definitions, we can mathematically present our priority optimization problem as follows:

$$\begin{aligned} \text{Maximize} \quad & f(\mathbf{s}) \\ \text{Subject to} \quad & 0 \leq s_i \leq 100, \quad i = 1, 2, \ldots, n \\ \text{with} \quad & \mathbf{s} = (s_1, s_2, \ldots, s_n) \end{aligned}$$

4 EXPERIMENTAL EVALUATION

This section details the experiments conducted to assess the LLM-based policy generation approach for a typical ICS network as our running example.

4.1 Ground Truth for the Running Example

For the typical ICS network shown in Figure 1, we generate the list of 26616 access requests by creating valid combinations of system attributes with their valid values. To create the access control decisions for generated access requests, we collaborate with cybersecurity experts who possess extensive knowledge of both the ICS environment and ABAC to identify 17 guidelines from the provided NIST guidelines to formulate fine-grained ABAC policies for the ICS network. To further refine these expert-generated policies, experts also develop qualitative intentions—a novel concept introduced in the QI-ABAC paper (Mitani et al., 2023). Both the expert-generated policies and these qualitative intentions are used to create a neural network-based classifier that can yield "allow" or "deny" decisions for input access requests. Ultimately, the decisions made by the trained classifier are considered ground truth. As a result, there are 18815 "allow" decisions and 7,801 "deny" decisions within the ground truth data.

Table 2: Preliminary performance of LLM-generated policies in access control decision-making.

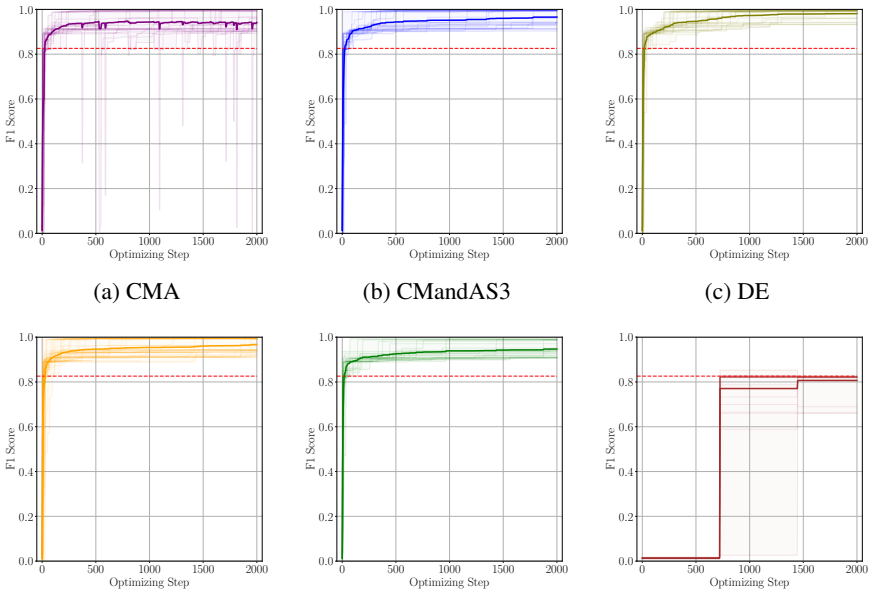
Combining Algorithm	TP	FP	TN	FN	Precision	Recall	F1
Deny-overrides	0	0	7801	18815	0	0	0.0
Allow-overrides	18729	7801	0	86	0.706	0.995	0.826
Priority-based	127	0	7801	18688	1	0.007	0.013
Weight-based	16	0	7801	18799	1	0.001	0.002

4.2 Preliminary Evaluation

In this section, we evaluate the feasibility of utilizing all fine-grained ABAC policies generated by LLMs for decision-making at a Policy Decision Point (PDP). There is a total of 181 generated policies. For policy conflict resolution, we utilize combining algorithms such as **Deny-overrides**, **Allow-overrides**, **Priority-based** and **Weight-based algorithm**. Note that the LLM-generated priority values are used as weights in the weight-based combining algorithm.

The preliminary evaluation of generated policies compares output decisions to ground truth decisions. We calculate the metrics using binary classification, with "allow" decisions classified as positive. The findings illustrated in Table 2 indicate that the overall performance is unsatisfactory. Although the allow-overrides algorithm performs better than the other algorithms with an F1 score of 0.826, the abundance of positive cases in the ground truth data raises concerns about the reliability of these output metrics.

Our analysis identifies several contributing factors to these adverse outcomes: (1) The high volume of LLM-generated policies may lead to conflicts. (2) The guidelines recommend various security strategies (e.g., whitelisting and blacklisting), which can con-



(a) CMA (b) CMandAS3 (c) DE

(d) NGOpt (e) Random Search (f) TBPSA

Figure 6: Optimization results with respect to eight algorithms. Each chart illustrates the testing F1 scores across 30 independent runs (in lighter lines) and the mean score (in darkest lines). The horizontal red dashed line denotes the baseline score.

flict within a specific ICS network. (3) The NIST guidelines are inherently more restrictive, prioritizing "deny" policies over "allow" ones. (4) LLMs can only address one guideline at a time, lacking a comprehensive understanding of all guidelines.

4.3 Priority Optimization Evaluation

In this section, we evaluate the performance of optimizers to optimize the policy priority values. The ground truth data is divided into 80% training and 20% testing data using stratified sampling to enhance the generalizability of the output solutions derived from this optimization process. This approach ensures that the distribution of positive and negative samples is maintained in both sets. We run each algorithm 30 times for 2000 optimization steps each. We regenerate the training and testing data for every run. The baseline score is the highest F1 score sourced from the allow-overrides algorithm (see Table 2) of the preliminary evaluation.

Optimization Results for Testing Data. Figure 6 shows the optimization process results for six observed algorithms, including CMA, CMandAS3, DE, NGOpt, RandomSearch, and TBPSA. From our observation of this figure, we can conclude various points:

1. In general, most of the observed algorithms yield F1 scores that surpass the baseline, demonstrating that priority optimization is viable when ground truth data is accessible. Additionally, the effectiveness of the algorithm exhibits variability across different runs.
2. In evaluating the mean performance of the algorithms across eight different charts, it is evident that **TBPSA** has the poorest results, whereas **DE** achieves the highest performance. Notably, **CMandAS3, DE, NGOpt**, and **Random Search** demonstrate an upward trend in F1 scores as the number of optimization steps increased. Additionally, **CMA** displays inconsistent performance in F1 scores during several optimization stages.

Overall Comparison. In this section, we assess the optimized solutions generated through the optimization process concerning all ground truth access requests. For each algorithm, the list of priority values that achieves the highest F1 score throughout the optimization process is its optimal solution. The comprehensive evaluation is displayed in Table 3.

From the table presented, the optimized priority values generated by the various optimizers are significantly superior to those produced by the LLM. Notably, the highest F1 score of 0.994 can be achieved with several optimizers, including **CMA, CMandAS3, DE** and **NGOpt**. However, **NGOpt** demonstrates a marginally better performance, achieving the highest Recall value of 0.989. In our system, **NGOpt** is currently used as the default optimization algorithm for refining the priority values of policies.

Table 3: Overall comparison between the performance of the LLM-generated and optimized priority values for access control decision-making. The bold values are the best results among the methods that use the priority-based combining algorithms, which are marked in gray.

	TP	FP	TN	FN	Precision	Recall	F1
Allow-overrides	18729	7801	0	86	0.706	0.995	0.826
LLM-based priority	127	0	7801	18688	1	0.007	0.013
CMA	18597	0	7801	218	1	0.988	**0.994**
CMandAS3	18597	0	7801	218	1	0.988	**0.994**
DE	18587	0	7801	228	1	0.988	**0.994**
NGOpt	**18609**	0	7801	**206**	1	**0.989**	**0.994**
Random Search	18504	0	7801	311	1	0.983	0.992
TBPSA	18462	6150	1651	353	0.75	0.981	0.85

4.4 Discussion

The experimental evaluation of a typical ICS network yields several noteworthy observations, along with limitations that require further consideration. Firstly, the proliferation of LLM-generated policies often leads to conflicts and redundancies. Detecting and correcting policy conflicts and redundancies represent critical future tasks to enhance overall efficiency. Secondly, priority optimization evaluations show that using the raw policies with optimized priority values in a priority-based combining algorithm can yield access control decisions comparable to those derived from the ground truth data. However, access to ground truth data is crucial for precisely adjusting the priorities of generated policies.

5 CONCLUSION

This study introduced a novel LLM-based methodology for developing fine-grained ABAC policies and addressing key challenges of LLMs, such as context insufficiency, and length limit. The approach combines various components, including data management and transformation, prompt construction with RAG-like knowledge integration and multi-turn template, attribute refinement, mixture-of-agents policy generation, and priority optimization.

We utilized a typical ICS network as a running example to generate 181 fine-grained ABAC policies. We discussed several reasons why directly applying these policies in decision-making processes yields undesirable results. Our experiments with various optimization algorithms indicated that refining the priority values greatly enhances the effectiveness of the generated policies, resulting in an F1 score of 0.994.

While priority optimization improves the access control decision-making of LLM-generated policies, its effectiveness is limited by reliance on ground truth data. In future work, we aim to reduce this dependence while optimizing policy priorities and exploring methods to identify optimal guideline/policy subsets and resolve policy conflicts.

REFERENCES

Ahmed, M., Wei, J., and Al-Shaer, E. (2024). Prompting LLM to Enforce and Validate CIS Critical Security Control. In *Proceedings of the 29th ACM Symposium on Access Control Models and Technologies*, pages 93–104. ACM.

Ait El Hadj, M., Benkaouz, Y., Freisleben, B., and Erradi, M. (2017). ABAC Rule Reduction via Similarity Computation. In *Networked Systems*, volume 10299, pages 86–100. Springer International Publishing.

Alohaly, M., Takabi, H., and Blanco, E. (2019). Towards an Automated Extraction of ABAC Constraints from Natural Language Policies. In *ICT Systems Security and Privacy Protection*, volume 562, pages 105–119. Springer International Publishing.

Cotrini, C., Weghorn, T., and Basin, D. (2018). Mining ABAC Rules from Sparse Logs. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 31–46. IEEE.

Heaps, J., Krishnan, R., Huang, Y., Niu, J., and Sandhu, R. (2021). Access Control Policy Generation from User Stories Using Machine Learning. In *Data and Applications Security and Privacy XXXV*, volume 12840, pages 171–188. Springer International Publishing.

Mitani, S., Kwon, J., Ghate, N., Singh, T., et al. (2023). Qualitative Intention-aware Attribute-based Access Control Policy Refinement. In *Proceedings of the 28th ACM Symposium on Access Control Models and Technologies*, pages 201–208. ACM.

Narouei, M., Khanpour, H., Takabi, H., et al. (2017). 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*, pages 103–114. ACM.

OpenStack Security Group (OSSG) (2024). Bandit: Security analyzer for python code. https://github.com/PyCQA/bandit. Version 1.7.10.

Rapin, J. and Teytaud, O. (2018). Nevergrad - A gradient-free optimization platform. https://GitHub.com/FacebookResearch/Nevergrad.

Smith, A. and Johnson, B. (2024). `code_diff`: Fast ast based code differencing in python. https://github.com/username/code_diff. Version 1.0.

Stouffer, K., Pillitteri, V., et al. (2015). Guide to Industrial Control Systems (ICS) Security.

Tarek, S., Saha, D., Saha, S. K., Tehranipoor, M., and Farahmandi, F. (2024). SoCureLLM: An LLM-driven Approach for Large-Scale System-on-Chip Security Verification and Policy Generation.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., et al. (2017). Attention is All you Need. In *Proceedings of the 31st International Conference on Neu-*

ral Information Processing Systems (NIPS'17), volume 30, pages 6000–6010. Curran Associates, Inc.

Wang, J., Wang, J., et al. (2024). Mixture-of-Agents Enhances Large Language Model Capabilities.

White, J., Fu, Q., Hays, S., Sandborn, M., et al. (2023). A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT.