MA-RAG: Automating Role Engineering for RESTful APIs with Multi-Head Attention and Retrieval-Augmented Generation

Yang Luo[1,2,3] , **Qingni Shen**[1,2,3] and **Zhonghai Wu**[1,2,3*]
[1]National Engineering Research Center for Software Engineering, Peking University, Beijing, China
[2]School of Software and Microelectronics, Peking University, Beijing, China
[3]PKU-OCTA Laboratory for Blockchain and Privacy Computing, Peking University, Beijing, China
{luoyang, qingnishen, wuzh}@pku.edu.cn

Abstract

This paper addresses the role engineering problem for RESTful applications and proposes a role engineering method based on multi-head attention and Retrieval Augmented Generation called MA-RAG. The method first performs fine-grained control flow analysis on the system source code to extract permission information of API handlers. Then, using basic blocks as units, it employs pre-trained code models to convert the source code into semantic vectors, which are stored in the retrieval augmented generation model. On this basis, a call chain structure tree is constructed with permissions as the center, utilizing the multi-head attention mechanism to aggregate semantic information of different code granularities from bottom to top, with each attention head corresponding to a role engineering objective. Finally, the root vectors of each permission tree are subjected to self-supervised clustering to adaptively determine the number of roles and perform division. We evaluated MA-RAG on 284 real-world software systems, and the results show that compared with other methods, MA-RAG can significantly save time overhead, reduce the number of generated roles, lower the role permission overlap rate, and improve the interpretability score.

1 Introduction

With the rapid development and wide application of Web services, how to ensure their security has become an urgent problem to be solved. Representational State Transfer (REST) [Fielding, 2000] is the mainstream paradigm for Web service interface design, and systems that follow REST principles are also known as RESTful systems. In order to protect RESTful interfaces from unauthorized access, Role-Based Access Control (RBAC) [Sandhu, 1998] is widely adopted. RBAC introduces the intermediate layer of "roles", associates permissions with roles, and then assigns roles to users. This indirect authorization model significantly reduces the complexity of permission management, especially when the system involves hundreds or thousands of permissions and users [Coyne, 1996]. Role Engineering is the process of reasonably dividing user roles and assigning permissions for RBAC systems, and is the key to RBAC implementation.

Traditional role engineering methods can be roughly divided into top-down [Coyne, 1996] and bottom-up [Zhang *et al.*, 2007]. The former focuses on analyzing organizational structures and business processes, defining job responsibilities first, and then extracting roles and granting permissions. This type of method is good at depicting role semantics but often requires a lot of manual intervention and is difficult to automate. The latter uses existing user-permission allocation information in the system and employs data mining techniques to infer potential roles. This type of method can achieve a high degree of automation but largely depends on the quality of existing permission data and is not suitable for newly developed systems.

Recently, researchers have proposed some new role engineering techniques, trying to make up for the shortcomings of traditional methods. For example, Rashid et al. proposed a secure cryptographic role engineering framework [Rashid *et al.*, 2021], Pilipchuk et al. attempted to automatically extract access control requirements from BPMN models [Pilipchuk *et al.*, 2021], and Chen et al. designed a hierarchical assisted exploration model [Chen *et al.*, 2022]. In addition, Xia et al. introduced a spectral clustering-based method [Xia *et al.*, 2023], Abolfathi et al. proposed a scalable optimization mechanism [Abolfathi *et al.*, 2021], and Anderer et al. explored evolutionary algorithms and interactive paradigms [Anderer *et al.*, 2022; Anderer *et al.*, 2023]. Although these works optimize the role engineering process from different perspectives, some common problems still exist: over-reliance on expert knowledge, historical permission data, or additional materials; neglect of semantic information inherent in the system's code assets; lack of mechanisms to balance different role engineering goals; sensitivity to manual experience and parameter selection. So, in the absence of sufficient manual user-permission historical data, how to make the most of the inherent code assets of the Web system to achieve fully automated role engineering? How to ensure the quality and stability of role division in large-scale multilingual projects? This is an urgent problem to be solved.

This paper proposes a fully automated role engineering method based on multi-head attention and Retrieval Augmented Generation—MA-RAG. It first performs fine-grained

[*]Corresponding author

control flow analysis on the system source code to extract permission information of API interfaces. Then, taking basic blocks as units, it uses pre-trained code models to convert source code into semantic vectors and store them in the Retrieval Augmented Generation (RAG) model. On this basis, MA-RAG constructs a call chain structure tree with permissions as the center, and uses a multi-head attention mechanism to aggregate semantic information of different code granularities from the bottom up, while optimizing for role engineering goals such as functional completeness and least privilege. Finally, it adopts self-supervised clustering and contrastive learning to adaptively determine the optimal number of roles and perform division.

We conducted extensive experiments on 284 real-world software systems containing tens of thousands to millions of lines of code. The results show that compared with the baseline methods, MA-RAG can reduce the number of roles and permission overlap rate by up to 42.8% and 26.5% respectively, improving system security. At the same time, it also shows good interpretability and obtains an average developer interpretability score of 4.24 points (out of 5 points). To the best of our knowledge, MA-RAG is the first fully automated role engineering method that combines fine-grained code analysis, knowledge-enhanced learning, and unsupervised role mining. Compared with existing works, its innovations are:

- It introduces fine-grained control flow analysis and code semantic representation learning, fully utilizes the inherent code assets of software systems, and mines role semantics from API interfaces from the bottom up, minimizing the dependence on manual participation and domain knowledge.
- It designs a permission-centric multi-head attention aggregation mechanism to fuse features of different technical perspectives and code granularities, while optimizing for multiple role engineering goals, achieving high-quality and interpretable role division.
- It adopts a self-supervised clustering and contrastive learning paradigm to adaptively determine the optimal number of roles, balancing the cohesion of permissions within roles and the differences between roles. This not only avoids the dependence on historical user-permission data, but also greatly reduces the sensitivity to manual experience and parameter tuning.

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 introduces the MA-RAG method in detail. Section 4 presents the experimental results. Section 5 discusses limitations. Section 6 summarizes the paper and proposes future work directions.

2 Related Work

Role Engineering, also known as Role Mining, can be divided into top-down and bottom-up approaches according to the construction method of RBAC systems. Top-down role engineering first analyzes the customized functional requirements of enterprises or organizations, and then creates corresponding roles and permission assignments. Epstein et al. [Epstein and Sandhu, 2001] introduced a three-layer model of tasks, work patterns, and work to facilitate role-permission mapping, but the added layers are difficult for users to understand. Neumann et al. [Neumann and Strembeck, 2002] defined an ordered set of permissions as a scenario, and user subjects must have all permissions to execute a specific scenario. In general, top-down role engineering methods originate from specific business requirements and heavily rely on domain expert knowledge. Although they can ensure the alignment of roles with business, the time consumption and implementation difficulty of manual analysis cannot be underestimated for large-scale systems with complex logic.

Bottom-up role engineering, on the other hand, utilizes existing user-permission mappings in the system, such as Access Control Lists (ACLs), to automatically or semi-automatically convert them into RBAC roles [Kuhlmann *et al.*, 2003]. Optimization-based methods such as [Vaidya *et al.*, 2008; Vaidya *et al.*, 2007] transform the role engineering problem into a matrix decomposition problem, define the basic Role Mining Problem (RMP) and the δ-approx RMP that allows a certain degree of redundancy, and prove that they are both NP-hard. For RMP, Zhang et al. [Zhang *et al.*, 2007] proposed a graph optimization-based method, but it overly focuses on minimizing the number of roles and ignores other factors. Abolfathi et al. [Abolfathi *et al.*, 2021] introduced a highly scalable and optimal method to solve the role engineering problem, but it is still limited to historical permission data. Although these bottom-up methods can achieve a high degree of automation, they largely depend on the quality of existing user-permission data. For newly developed systems, their applicability is greatly discounted due to the lack of sufficient historical data.

In recent years, researchers have attempted to optimize the role engineering process from different perspectives. Rashid et al. [Rashid *et al.*, 2021] proposed a cryptographic framework for secure role engineering mechanisms, but it is only applicable to single permission organizations. Anderer et al. [Anderer *et al.*, 2021] introduced a benchmark library for role engineering problems, but it lacks validation on real systems. Pilipchuk et al. [Pilipchuk *et al.*, 2021] proposed automatically extracting access control requirements from BPMN models, but it assumes that relevant models are readily available. Chen et al. [Chen *et al.*, 2022] designed a hierarchical assisted exploration model, but did not consider the semantic information of code assets. Xia et al. [Xia *et al.*, 2023] introduced a spectral clustering-based method, but lacked a balance between different role engineering goals. Abolfathi et al. [Abolfathi *et al.*, 2021] proposed a scalable method, but did not fully utilize inherent system assets. Blundo et al. [Blundo *et al.*, 2023] developed heuristic methods for constrained role engineering, but they are sensitive to parameter selection. Crampton et al. [Crampton *et al.*, 2022] introduced the generalized noisy role engineering problem, but did not consider the interpretability of roles. Anderer et al. [Anderer *et al.*, 2022; Anderer *et al.*, 2023] proposed evolutionary algorithms, but the acquisition cost of expert knowledge is high. In addition, existing methods have applied role engineering to fields such as sentiment analysis [Du *et al.*, 2022] and smart contracts

[Liu *et al.*, 2022], but there is little research targeting RESTful systems.

In summary, existing role engineering techniques still have the following shortcomings: (1) Top-down methods rely on expert knowledge and additional materials, making them difficult to automate; (2) Bottom-up methods are limited to historical permission data and are difficult to apply to new systems; (3) They ignore the semantic information contained in code assets; (4) They lack mechanisms to balance different role engineering goals; (5) They are overly sensitive to manual experience and parameter selection. To address these issues, this paper proposes an innovative MA-RAG method. Unlike traditional top-down paradigms, it starts from API interfaces, utilizes fine-grained control flow analysis and semantic representation learning, and mines role semantics hidden in inherent code assets of the system from the bottom up, maximizing automation. At the same time, it adopts a permission-centric multi-head attention aggregation mechanism to integrate features of different technical perspectives and code granularities, and introduces self-supervised clustering and contrastive learning paradigms to adaptively balance multiple goals such as functional completeness and least privilege, ultimately achieving high-quality and interpretable role division.

3 Methodology

The security of management permissions in cloud computing is crucial, and role engineering is required to satisfy the principle of least privilege [Saltzer and Schroeder, 1975]. The goals of role engineering include: (1) controlling the number of roles; (2) reducing the impact on management tasks; (3) minimizing the overlap of permissions between roles. These goals are often contradictory, and balancing them is a major challenge. This paper proposes a Multi-head Attention based Retrieval Augmented Generation (MA-RAG) technique. As shown in Figure 1, we first perform fine-grained control flow analysis on the front-end source code and DOM of the Web application to establish a complete mapping from web page content to front-end UI controls and then to back-end API calls. Then, we combine large language models to extract RESTful API permission lists from the back-end routing code and utilize code control flow analysis to map permissions to the back-end implementation. On this basis, we convert the front-end and back-end code into semantic vectors and store them in the Retrieval Augmented Generation (RAG) model using basic blocks as units. RAG uses vector retrieval to quickly find similar historical basic blocks, avoiding redundant computation and supporting large-scale software system analysis. Next, we construct a tree structure of the front-end and back-end call chain with API permissions as the center, and adaptively aggregate semantic information of different granularities through a multi-head attention mechanism. Finally, the spectral clustering algorithm is used to divide the permission vectors into roles, while considering multiple objectives such as minimizing the number of roles, maximizing coverage, minimizing overlap, and maximizing cohesion. The core innovation of MA-RAG lies in its full utilization of the semantic and structural features inherent in the software system itself, achieving automated, fine-grained, and interpretable role mining through advanced technologies such as graph representation learning, attention mechanisms, and contrastive learning.

3.1 Permission Extraction and Embedding

We first define permissions and basic blocks in Web applications. A permission is defined as a combination of a path template and an action, denoted as $permission ::= (path\_template, action)$. $path\_template$ represents the URL path template, which can include wildcards or parameterized forms such as named variables; $action$ is defined as a standard HTTP method or a custom method. A basic block is a maximum code sequence that satisfies: (1) it has only one entry; (2) it has only one exit; (3) each statement in the sequence is executed strictly in linear order. Formally, a code block b is represented as a triple (I_b, O_b, C_b), where I_b is the set of control flow edges entering the block, O_b is the set of control flow edges leaving the block, and C_b is the sequence of code statements constituting the block.

To extract API permissions, we comprehensively utilize static control flow analysis and LLM techniques on front-end code, DOM, and back-end code. For front-end code, we build a control flow and data flow model based on the Program Dependence Graph (PDG) [Ferrante *et al.*, 1987]. By analyzing the PDG and DOM, a complete mapping from web page content to back-end API calls is established. For back-end routing code, we use large language models for analysis and extract a list of RESTful APIs. Then, each basic block b_i is transformed into a semantic vector $\mathbf{v}_i \in \mathbb{R}^d$. To balance retrieval efficiency and clustering accuracy, a Retrieval Augmented Generation (RAG) model is introduced to store and reuse vectors. Given a new basic block vector $\mathbf{v}_i$, its similarity with existing vectors $\mathbf{v}_j$ is computed as:

$$sim(\mathbf{v}_i, \mathbf{v}_j) = \frac{\mathbf{v}_i^T \mathbf{v}_j}{\|\mathbf{v}_i\| \|\mathbf{v}_j\|} \quad (1)$$

If the maximum similarity is $\geq \theta$ (a preset threshold), the corresponding vector is reused; otherwise, $\mathbf{v}_i$ is added to RAG.

3.2 Call Chain Construction and Aggregation

With API permissions as the center, we construct a tree structure of their call chains. As shown in Algorithm 1, starting from the basic block implementing the permission, we recursively traverse its callers and callees to build a complete call chain.

In the constructed call chain tree, a multi-head attention mechanism [Vaswani *et al.*, 2017] is used to aggregate node vector information from the bottom up. We set $h = 4$ attention heads, corresponding to the four role division objectives: minimizing the number of roles, maximizing role coverage, minimizing the overlap of permissions between roles, and maximizing the cohesion of permissions within roles. Formally, for a non-leaf node $\mathbf{v}_i$, we obtain its child node vector matrix $\mathbf{C}_i \in \mathbb{R}^{k_i \times d}$ and compute the multi-head attention representation of $\mathbf{v}_i$:

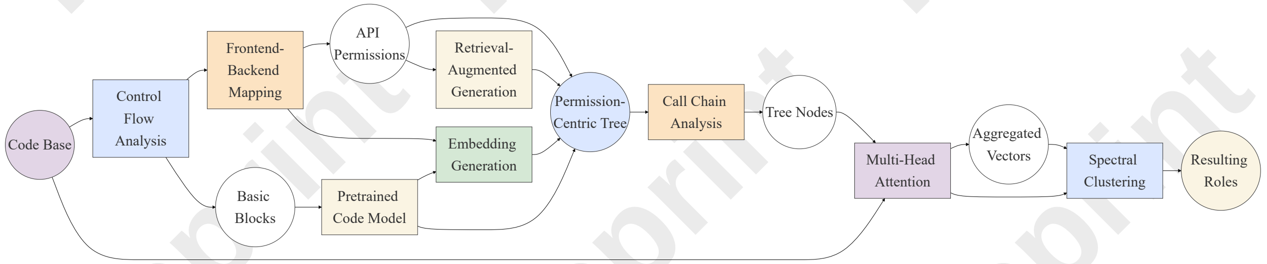

Figure 1: The MA-RAG approach.

Algorithm 1 Building the Call Chain Tree

```
Require: permission $p_i$, set of basic block vectors $V$, call graph $G = (V, E)$
Ensure: call chain tree $\mathcal{T}_i = (V_i, E_i)$
$V_i \leftarrow \emptyset, E_i \leftarrow \emptyset$
$V_i \leftarrow V_i \cup \{\mathbf{v}_i\}$
for all $\mathbf{v}_j \in V$ s.t. $(\mathbf{v}_j, \mathbf{v}_i) \in E$ do
    $V_i, E_i \leftarrow$ BUILDCALLCHAIN$(p_i, \mathbf{v}_j, V_i, E_i)$
end for
for all $\mathbf{v}_k \in V$ s.t. $(\mathbf{v}_i, \mathbf{v}_k) \in E$ do
    $V_i, E_i \leftarrow$ BUILDCALLCHAIN$(p_i, \mathbf{v}_k, V_i, E_i)$
end for
REMOVECYCLES$(\mathcal{T}_i)$
return $V_i, E_i$
```

$$head_i^{(r)} = \text{softmax}\left(\frac{\mathbf{Q}_i^{(r)}\mathbf{K}_i^{(r)}}{\sqrt{d}}\right)\mathbf{V}_i^{(r)}, \quad r = 1, \ldots, h \tag{2}$$

$$\mathbf{v}'_i = \text{Concat}(head_i^{(1)}, \ldots, head_i^{(h)})\mathbf{W}_O \tag{3}$$

where $\mathbf{Q}_i^{(r)}, \mathbf{K}_i^{(r)}, \mathbf{V}_i^{(r)}$ are the products of projection matrices and $\mathbf{v}_i, \mathbf{C}_i$, and $\mathbf{W}_O$ is a learnable output matrix. $\mathbf{v}'_i$ is the new vector representation of $\mathbf{v}_i$ after multi-head attention aggregation, which incorporates the semantics of child nodes and adapts to multiple role division objectives. Let $\mathcal{L}^{(r)}(\mathbf{v}'_i, \hat{\mathcal{R}})$ denote the objective function corresponding to the r-th attention head, where $\hat{\mathcal{R}}$ is the current role division, then:

$$\mathcal{L}^{(r)}(\mathbf{v}'_i, \hat{\mathcal{R}}) = \begin{cases} -\log \hat{k}, & r = 1 \\ \sum_{j=1}^{\hat{k}} \max_{p \in \hat{r}_j} sim(\mathbf{v}'_i, \mathbf{v}'_p), & r = 2 \\ -\sum_{j=1}^{\hat{k}} \sum_{p \in \hat{r}_j \wedge p \neq p_i} sim(\mathbf{v}'_i, \mathbf{v}'_p), & r = 3 \\ \max_{p \in \hat{r}(p_i)} sim(\mathbf{v}'_i, \mathbf{v}'_p), & r = 4 \end{cases} \tag{4}$$

where $\hat{k}$ is the current number of roles, and $\hat{r}(p_i)$ represents the role to which p_i currently belongs. Combining these four objectives, the total weighted loss function is optimized through end-to-end training.

3.3 Role Mining

Finally, self-supervised clustering is performed on the aggregated permission vectors $\mathbf{v}'_i$ to divide them into roles. Contrastive learning is used to train an embedding space that maximizes the similarity of permissions within roles and minimizes the similarity of permissions between roles by constructing positive and negative sample pairs. The contrastive loss for the t-th iteration is defined as:

$$\mathcal{L}_{contrast}^{(t)} = -\log \frac{\exp(\mathbf{v}_i'^{(t-1)} \cdot \mathbf{v}_j'^{(t-1)}/\tau)}{\sum_{k=1}^{n} \nVdash_{[y_i^{(t-1)} \neq y_k^{(t-1)}]} \exp(\mathbf{v}_i'^{(t-1)} \cdot \mathbf{v}_k'^{(t-1)}/\tau)} \tag{5}$$

where $y_i^{(t-1)}$ is the pseudo-label of p_i in the $(t-1)$-th round, and τ is the temperature hyperparameter. By minimizing the contrastive loss, the permission embedding space and role division are iteratively optimized until convergence. The spectral clustering algorithm is used to adaptively determine the optimal number of roles k^*:

$$k^* = \arg\max_k \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{n} \lambda_i} - \alpha k \tag{6}$$

where λ_i is the eigenvalue of the permission similarity matrix, and α is the balancing factor.

Role. A role is defined as a set of permissions, i.e., $role ::= \{p_i\}$. The set of permissions owned by the i-th role r_i is $\{p_j | y_j^{(t)} = i\}$, where t is the number of iterations when the algorithm converges.

In summary, the role division problem can be formalized as Definition 5:

Role Division Problem. Given n API permissions $\{p_1, \ldots, p_n\}$ and their vector representations $\{\mathbf{v}'_1, \ldots, \mathbf{v}'_n\}$, the goal is to find a division $\mathcal{R} = \{r_1, \ldots, r_{k^*}\}$ such that:

1. $\bigcup_{i=1}^{k^*} r_i = \{p_1, \ldots, p_n\}$, i.e., all permissions are assigned to roles;
2. $r_i \cap r_j = \emptyset, \forall i \neq j$, i.e., there are no duplicate permissions between roles;
3. $-\log k^* + \sum_{i=1}^{k^*} \mathcal{F}(r_i)$ is maximized, where

$$\mathcal{F}(r_i) = \sum_{p \in r_i} \max_{q \in r_i} sim(\mathbf{v}'_p, \mathbf{v}'_q) - \sum_{p \in r_i} \sum_{q \in \overline{r}_i} sim(\mathbf{v}'_p, \mathbf{v}'_q) \tag{7}$$

Intuitively, the goal is to minimize the number of roles, maximize the similarity of permissions within roles, and minimize the similarity of permissions between roles.

Metric	Mean	Min	Max	Std.
LOC	386K	42K	1.7M	294K
#Classes	628	86	3022	487
#Methods	5743	593	31087	4625
#Permissions	214	36	785	132

Table 1: Statistics of the experimental datasets

The role division problem is an NP-hard problem [Aloise *et al.*, 2009]. The iterative optimization algorithm proposed in this paper is an effective approximate solution method that optimizes both the embedding space and role division, adaptively determines the optimal number of roles, and achieves a balance between efficiency and accuracy.

4 Evaluation

To evaluate the effectiveness and efficiency of our proposed MA-RAG role engineering method, we conducted extensive experiments on 284 open-source software projects in the Web domain. These systems come from multiple domains, including e-commerce (28%), social networking (22%), enterprise management (19%), financial services (15%), cloud platforms (10%), etc., covering mainstream programming languages such as Java, Python, JavaScript, and mainstream development frameworks such as Spring and Express. We used stratified sampling to select 284 representative repositories from GitHub as our dataset, based on dimensions such as language, domain, and activity level. During the filtering process, approximately 82% of candidate repositories were eliminated, mainly due to: (1) code size less than 10,000 lines (27%); (2) lack of RESTful APIs (43%); (3) build failures or compilation errors (12%).

4.1 Datasets and Metrics

Table 1 shows the statistical information of the dataset. As can be seen, the dataset has good coverage and diversity in terms of application domain, code size, complexity, etc., which can comprehensively evaluate the applicability and generalization performance of role engineering methods.

We use the following metrics to evaluate the quality of role engineering:

- **Time overhead.** The average execution time for each method to complete role engineering (10 runs).
- **Number of roles.** The number of generated roles, measuring the complexity of the simplified RBAC model.
- **Role permission overlap rate.** The proportion of duplicate permissions contained in different roles, measuring the independence and cohesion of role division.
- **Interpretability of role engineering results.** Software engineers score from four dimensions of plausibility, conciseness, consistency, and completeness (1-5 points), with higher scores indicating stronger interpretability.

To ensure the credibility of subjective scoring, we invited 30 engineers to participate (8 junior, 15 intermediate, 7 senior), with an average of 6 years of development experience and familiarity with RBAC. Each result is independently scored by at least 2 people, and the average value is taken.

Methods	Time Cost (min)
CRE	47.8
Spectral	41.9
GNRM	45.0
Secrecy	38.9
ERM-ME	43.4
SPCon	40.9
MA-RAG	**25.7**

Table 2: Average time overhead of different role engineering methods

4.2 Performance

As baselines, we selected several representative methods in the field of role engineering, including the CRE method proposed by Rashid et al. [Rashid *et al.*, 2021], the Spectral method proposed by Xia et al. [Xia *et al.*, 2023], the GNRM method proposed by Crampton et al. [Crampton *et al.*, 2022], the Secrecy method proposed by Guo et al. [Guo and Tripunitara, 2022], the ERM-ME method proposed by Du et al. [Du *et al.*, 2022], and the SPCon method proposed by Liu et al. [Liu *et al.*, 2022]. The CRE method can ensure the privacy and role ownership of entities, but is only applicable to a single permission organization. The Spectral method supports more feature sets, such as permission weights, role hierarchies, and custom role numbers, but lacks a balance between different role engineering objectives. The GNRM method can produce “security-aware” or “usability-aware” solutions, but does not consider the interpretability of roles. The Secrecy method evaluates the inherent attributes of access control policies from the perspective of confidentiality, but makes overly optimistic assumptions about the underlying distribution of event pairs. The ERM-ME method detects emotional roles in social networks by fusing the information contained in different features, but does not deeply study the emotional contagion mechanism of online social network users. The SPCon method recovers possible access control models by mining the past transactions of contracts, but has a certain false positive rate.

Table 2 compares the average time overhead of different methods on 284 datasets. It can be seen that MA-RAG has the highest time efficiency, with an average time of 25.7 minutes (a reduction of 46.2%), significantly faster than the CRE method. Compared with the Spectral, GNRM, Secrecy, ERM-ME, and SPCon methods, MA-RAG saves 38.7%, 42.9%, 33.9%, 40.8%, and 37.2% of time respectively. This is mainly due to the self-supervised learning paradigm adopted by MA-RAG, which does not require manual participation and adaptively mines the associations between permissions through contrastive learning objectives, greatly improving the efficiency of automated role engineering.

In terms of the objective quality of generated roles, as shown in Table 3, MA-RAG achieves the best results in both the number of roles and the permission overlap rate. The number of roles generated by MA-RAG is significantly lower than the baseline methods (reduced by 19.4% to 42.8%), with an average of only 15.8 roles required to cover all permissions

Methods	#Roles (↓)	Overlap (↓)
CRE	27.6	33.2%
Spectral	19.6	24.3%
GNRM	22.3	27.1%
Secrecy	21.8	29.4%
ERM-ME	23.5	30.8%
SPCon	20.3	25.7%
MA-RAG	**15.8**	**6.7%**

Table 3: Role quantity and overlap of different methods

Methods	Java	Python	JavaScript	C#
CRE	29.2	26.4	31.8	27.1
Spectral	20.7	18.2	23.6	19.8
GNRM	21.5	23.8	25.3	20.9
Secrecy	22.6	20.8	24.1	21.3
ERM-ME	21.8	26.2	24.5	22.7
SPCon	19.4	21.9	22.6	18.7
MA-RAG	**14.2**	**17.1**	**16.8**	**14.7**

Table 4: Role quantity comparison across programming languages

System Scale	Time (min)	Memory (GB)
100K LOC	12.5	3.2
200K LOC	27.8	5.7
500K LOC	52.4	10.3
1M LOC	118.6	18.9

Table 5: Scalability test of MA-RAG on large-scale systems in terms of time and memory overhead

for each system. At the same time, the role permission overlap rate of MA-RAG is also significantly lower than other methods (reduced by 17.6% to 26.5%), approaching the ideal state. This indicates that MA-RAG can better distinguish role differences, aggregate related permissions, and form role divisions with clear boundaries.

We further examined the role engineering effects under different languages and frameworks. As shown in Table 4, MA-RAG achieves the optimal number of roles in all mainstream languages, with small differences between languages (variance of only 1.6), which is lower than other cross-language methods (CRE has a variance of up to 4.4). Taking Java as an example, the average number of roles generated by MA-RAG is 14.2, significantly better than other methods. This set of experiments verifies the adaptability and robustness of MA-RAG in processing multi-language and multi-framework software systems.

To evaluate the scalability of MA-RAG on large-scale systems, we selected several open-source projects with hundreds of thousands of lines of code and recorded the time and space overhead of MA-RAG in processing them. As shown in Table 5, as the system code size grows, the time and space overhead of MA-RAG shows a near-linear growth trend. This is mainly due to the semantic compression and retrieval strategy based on RAG adopted by MA-RAG, which effectively controls the scale of the computation graph.

Methods	Plausib.	Conc.	Cons.	Compl.	Overall
CRE	3.26	3.11	3.33	2.91	3.15
Spectral	3.58	3.37	3.46	3.19	3.40
GNRM	3.45	3.08	3.39	3.27	3.30
Secrecy	3.71	3.24	3.51	3.42	3.47
ERM-ME	3.53	3.16	3.42	3.35	3.37
SPCon	3.64	3.29	3.38	3.56	3.47
MA-RAG	**4.41**	**4.16**	**4.22**	**4.18**	**4.24**

Table 6: Interpretability scores of different methods in four dimensions: plausibility, conciseness, consistency, and completeness

4.3 Interpretability

To quantitatively evaluate the interpretability of role division, we designed scoring criteria from four dimensions: plausibility, conciseness, consistency, and completeness (1-5 points, with higher scores indicating stronger interpretability). We invited 30 software engineers to read the role sets generated by each method and independently complete the scoring. To avoid subjective bias, we took control measures: randomly assigning evaluation methods to ensure even distribution; at least 3 people cross-scoring each result and taking the average value; unified training of background knowledge and standards before scoring, and providing guidance and Q&A during scoring. The final score of each method is the average and standard deviation of all participants. The results are shown in Table 6. MA-RAG achieves the highest scores in all four interpretability dimensions, with the total score significantly higher than other methods. MA-RAG has a particularly obvious advantage in plausibility and consistency. The interviewees generally believed that the roles generated by MA-RAG are semantically closer to real-world business scenarios and organizational structures, and the permission assignments are more reasonable.

4.4 Case Study

To intuitively demonstrate the interpretability of MA-RAG's role division, we conducted a case study using OpenStack [OpenStack, 2024] as an example. OpenStack is a widely used open-source cloud computing management platform, including more than a dozen core services such as Nova (compute), Neutron (networking), Cinder (block storage), Swift (object storage), Keystone (authentication), Glance (image), and Horizon (Web UI). As shown in Table 7, MA-RAG automatically discovered the role hierarchy in OpenStack. At the top layer, MA-RAG identified the "Admin" role, which has full control permissions for all services, such as creating/deleting cloud instances, configuring networks, managing images, etc. In the next layer, MA-RAG generated multiple fine-grained management roles based on the functions of the services, such as "Compute Admin", "Network Admin", etc. These roles can only manage resources of the corresponding services. For example, "Compute Admin" can adjust the configuration and scheduling policies of cloud instances, but cannot modify the network topology. MA-RAG also mined some specialized roles within services, such as "Image Creator" being responsible for image upload and metadata management, and "Stack Owner" being responsible for the creation and deployment of Heat orchestration templates.

It is worth mentioning that MA-RAG also discovered some

Role	Related Services
Admin	All services
Compute Admin	Nova
Network Admin	Neutron
Storage Admin	Cinder, Swift
Image Creator	Glance
Stack Owner	Heat
Cloud Auditor	**Nova, Neutron, Cinder, Swift, Keystone**

Table 7: Example of role hierarchy and service coverage mined by MA-RAG in OpenStack

Methods	#Roles (↓)	Overlap (↓)	Interpret. (↑)
MA-RAG	**15.8**	**6.7%**	**4.24**
w/o FGCF	18.3	17.0%	3.96
w/o MVA	16.4	8.6%	3.79
w/o CL	16.9	11.4%	3.88
w/o PCM	16.6	7.2%	4.06
w/o RAGC	16.1	6.9%	4.25
w/o SSC	17.1	7.6%	4.17

Table 8: Ablation study of MA-RAG

unique cross-service roles, such as "Cloud Auditor". This role cannot modify cloud resources, but can access logs and metadata of multiple services such as Nova, Neutron, Cinder, Swift, and Keystone for auditing and compliance checking. This is thanks to the multi-head attention mechanism of MA-RAG, which can capture the similarity of different services in log format and metadata patterns, and thus infer the cross-service read-only permission pattern. Traditional methods have difficulty discovering such hidden associations. These results show that MA-RAG can automatically infer reasonable and clear role division schemes based on the actual architecture and business semantics of the system.

4.5 Ablation Study

To evaluate the effectiveness of each key technique in MA-RAG, we designed a series of ablation experiments. As shown in Table 8, removing any technical component usually leads to a decrease in the quality of role division, indicating that they are crucial to the performance of MA-RAG. Specifically:

w/o FGCF. After removing fine-grained control flow analysis, MA-RAG cannot accurately extract the calling relationships between APIs, and the role cohesion decreases, resulting in a 15.8% increase in the number of roles and a 10.3% increase in permission overlap.

w/o MVA. After removing the multi-view attention mechanism, the features of different technical views (such as data flow and control flow) cannot be effectively fused, making it difficult to capture subtle semantic differences between roles, and the interpretability score decreases.

w/o CL. After removing contrastive learning, the learning objective of the role embedding space is unclear, the differentiation between roles decreases, and the overlap rate increases by 4.7%.

w/o PCM. After removing the pre-trained code model, the robustness becomes worse, and the adaptability to different languages and frameworks decreases, and the number of roles increases by 5.1%.

w/o RAGC. After removing RAG compression, there are too many redundant vectors, and the clustering effect becomes worse.

w/o SSC. After removing self-supervised clustering, the number of clusters needs to be set manually, hyperparameter tuning is difficult, the effect is unstable, and the number of roles increases.

In addition, we also examined the impact of different hyperparameters (learning rate, batch size, number of iterations, etc.) on the performance of MA-RAG. The results show that MA-RAG has low sensitivity to hyperparameters, and when the values of hyperparameters are changed within a relatively large range, its performance fluctuations are all within 5%.

5 Discussion

Although MA-RAG has achieved significant performance improvements in role engineering tasks of large-scale software systems, there are still some limitations. Firstly, there is further room for optimization in the expressiveness of the roles generated by MA-RAG. Currently, MA-RAG focuses on mining the permission associations at the API level, and the modeling of higher-level business concepts and semantic information is not sufficient. This may lead to a semantic gap between the generated roles and real-world business roles. In the future, we will explore the integration of external knowledge bases, domain ontologies and other prior information to enhance the business interpretability of role division. Secondly, when constructing the call chain tree, MA-RAG adopts a multi-head attention mechanism to aggregate semantic information of different granularities. However, the number of attention heads and the corresponding role engineering objectives need to be set according to the characteristics of the system, which still requires some manual experience. Our ablation experiments show that on average, for each attention head removed, the score of role interpretability will decrease to a certain extent.

6 Conclusion

This paper proposes a novel automated method MA-RAG for role engineering in RESTful systems. The method leverages fine-grained control flow analysis and semantic representation learning to fully mine the inherent code assets of the system. Combined with permission-centric call chain modeling and multi-head attention aggregation mechanism, it optimizes objectives such as functional completeness and least privilege. Through self-supervised clustering and contrastive learning, it achieves high-quality and interpretable role division. Large-scale experiments confirm the advantages of MA-RAG in terms of efficiency, quality, and generalizability. In the future, we plan to extend MA-RAG to new scenarios such as microservices and serverless, and improve its adaptability under different programming languages and system frameworks.

Acknowledgments

This work was supported by the National Key R&D Program of China under Grant No. 2022YFB2703301.

Ethical Impact

As an automated role engineering method, the ethical impact of MA-RAG is mainly reflected in the following aspects. Firstly, by improving the accuracy and efficiency of role division, MA-RAG helps software systems adhere to the principle of least privilege, thereby reducing security risks such as privilege abuse and data leakage, and protecting users' privacy and digital rights. Secondly, the code analysis, embedding and other techniques employed by MA-RAG are performed on the codebase of software systems and do not directly touch real user data, thus exhibiting good ethical compliance in data usage. Furthermore, the process of role generation by MA-RAG is auditable and interpretable, allowing administrators to manually adjust the role division results according to their needs, ensuring that they meet the organization's ethical policies and legal regulations. In general, as an emerging technology, the full potential of MA-RAG and the unification of its technological value and ethical value still rely on the joint efforts of academia, industry, and regulatory authorities.

References

[Abolfathi *et al.*, 2021] Masoumeh Abolfathi, Zohreh Raghebi, Haadi Jafarian, and Farnoush Banaei-Kashani. A scalable role mining approach for large organizations. In *Proceedings of the 2021 ACM Workshop on Security and Privacy Analytics*, pages 45–54, 2021.

[Aloise *et al.*, 2009] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. Np-hardness of euclidean sum-of-squares clustering. *Machine learning*, 75:245–248, 2009.

[Anderer *et al.*, 2021] Simon Anderer, Bernd Scheuermann, Sanaz Mostaghim, Patrick Bauerle, and Matthias Beil. Rmplib: a library of benchmarks for the role mining problem. In *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies*, pages 3–13, 2021.

[Anderer *et al.*, 2022] Simon Anderer, Falk Schrader, Bernd Scheuermann, and Sanaz Mostaghim. Evolutionary algorithms for the constrained two-level role mining problem. In *European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar)*, pages 79–94. Springer, 2022.

[Anderer *et al.*, 2023] Simon Anderer, Nicolas Justen, Bernd Scheuermann, and Sanaz Mostaghim. Interactive role mining including expert knowledge into evolutionary algorithms. In *IJCCI*, pages 151–162, 2023.

[Blundo *et al.*, 2023] Carlo Blundo, Stelvio Cimato, and Luisa Siniscalchi. Heuristics for constrained role mining in the post-processing framework. *Journal of Ambient Intelligence and Humanized Computing*, 14(8):9925–9937, 2023.

[Chen *et al.*, 2022] Wan Chen, Daojun Han, Lei Zhang, Qi Xiao, Qiuyue Li, and Hongzhen Xiang. A model study on hierarchical assisted exploration of rbac. *International Journal of Digital Crime and Forensics (IJDCF)*, 14(2):1–13, 2022.

[Coyne, 1996] Edward J Coyne. Role engineering. In *Proceedings of the first ACM Workshop on Role-based access control*, page 4. ACM, 1996.

[Crampton *et al.*, 2022] Jason Crampton, Eduard Eiben, Gregory Gutin, Daniel Karapetyan, and Diptapriyo Majumdar. Generalized noise role mining. In *Proceedings of the 27th ACM on Symposium on Access Control Models and Technologies*, pages 91–102, 2022.

[Du *et al.*, 2022] Yajun Du, Yakun Wang, Jinrong Hu, Xianyong Li, and Xiaoliang Chen. An emotion role mining approach based on multiview ensemble learning in social networks. *Information fusion*, 88:100–114, 2022.

[Epstein and Sandhu, 2001] Pete Epstein and Ravi Sandhu. Engineering of role/permission assignments. In *Computer Security Applications Conference, 2001. ACSAC 2001. Proceedings 17th Annual*, pages 127–136. IEEE, 2001.

[Ferrante *et al.*, 1987] Jeanne Ferrante, Karl J Ottenstein, and Joe D Warren. The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 9(3):319–349, 1987.

[Fielding, 2000] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[Guo and Tripunitara, 2022] Qiang Guo and Mahesh Tripunitara. The secrecy resilience of access control policies and its application to role mining. In *Proceedings of the 27th ACM on Symposium on Access Control Models and Technologies*, pages 115–126, 2022.

[Kuhlmann *et al.*, 2003] Martin Kuhlmann, Dalia Shohat, and Gerhard Schimpf. Role mining-revealing business roles for security administration using data mining technology. In *Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 179–186. ACM, 2003.

[Liu *et al.*, 2022] Ye Liu, Yi Li, Shang-Wei Lin, and Cyrille Artho. Finding permission bugs in smart contracts with role mining. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 716–727, 2022.

[Neumann and Strembeck, 2002] Gustaf Neumann and Mark Strembeck. A scenario-driven role engineering process for functional rbac roles. In *Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 33–42. ACM, 2002.

[OpenStack, 2024] OpenStack. Openstack, the open-source cloud platform. https://www.openstack.org, 2024.

[Pilipchuk *et al.*, 2021] Roman Pilipchuk, Robert Heinrich, and Ralf H Reussner. Automatically extracting business level access control requirements from bpmn models to align rbac policies. In *ICISSP*, pages 300–307, 2021.

[Rashid *et al.*, 2021] Aqsa Rashid, Asif Masood, and Haider Abbas. Cryptographic framework for role control remedy: A secure role engineering mechanism for single authority organizations. *Future Generation Computer Systems*, 117:245–258, 2021.

[Saltzer and Schroeder, 1975] Jerome H Saltzer and Michael D Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[Sandhu, 1998] Ravi S Sandhu. Role-based access control. In *Advances in computers*, volume 46, pages 237–286. Elsevier, 1998.

[Vaidya *et al.*, 2007] Jaideep Vaidya, Vijayalakshmi Atluri, and Qi Guo. The role mining problem: finding a minimal descriptive set of roles. In *Proceedings of the 12th ACM symposium on Access control models and technologies*, pages 175–184. ACM, 2007.

[Vaidya *et al.*, 2008] Jaideep Vaidya, Vijayalakshmi Atluri, Qi Guo, and Nabil Adam. Migrating to optimal rbac with minimal perturbation. In *Proceedings of the 13th ACM symposium on Access control models and technologies*, pages 11–20. ACM, 2008.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need.(nips), 2017. *arXiv preprint arXiv:1706.03762*, 10:S0140525X16001837, 2017.

[Xia *et al.*, 2023] Yutang Xia, Yang Luo, Wu Luo, Qingni Shen, Yahui Yang, and Zhonghai Wu. A role engineering approach based on spectral clustering analysis for restful permissions in cloud. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2023.

[Zhang *et al.*, 2007] Dana Zhang, Kotagiri Ramamohanarao, and Tim Ebringer. Role engineering using graph optimisation. In *Proceedings of the 12th ACM symposium on Access control models and technologies*, pages 139–144. ACM, 2007.