

A Novel Approach to Generative AI-based Optimized Code Generation for Semiconductor Equipment Interfaces

HyeokSoo Lee^{*,**†}, Minyeol Yang^{**,†}, Jongpil Jeong^{*}

^{*} Department of Smart Factory Convergence, Sungkyunkwan University, 2066 Seobu-ro, Jangan-gu, Suwon, 16419, Korea

^{**} Research & Development Team, THiRA-UTECH Co., Ltd, CK Bldg 4F Hakdong-ro 5-gil 7 Gangnam-gu, Seoul, 06044, Korea

huxlee@g.skku.edu, alsduf63@thirautech.com, jpjeong@skku.edu

Abstract— Generally, standardization defined by the SEMI association is well established and utilized in the semiconductor industry. In particular, most semiconductor equipment supports SECS / GEM communication protocols, and the automation and smart factory construction consist of equipment communication control programs using these standard protocols. We propose to improve development efficiency by automatically generating control program code using generative artificial intelligence technology to develop interface programs that control these semiconductor facilities. In addition, to improve the completeness and utilization of the automatically generated code, this paper presents a method to automatically generate semiconductor equipment control interface codes through generative artificial intelligence based on existing codes and minimize the constraints that may occur due to the hallucination effect, which is a significant weakness of generative artificial intelligence.

Keywords— Generative AI, Code Generation, LLM (Large Language Model), RAG (Retrieval Augmented Generation), Prompt

I. INTRODUCTION

Among many businesses, the semiconductor industry requires enormous investment, so standardization has been promoted since the 70s. It is very well established, and vendors, companies, and institutions in the semiconductor industry follow these standards well. Due to the nature of the process and environmental constraints, semiconductor factories require minimal human intervention, which is why they are fully automated and unmanned. As such, equipment is a key component of automation in semiconductor factories. It would be a great burden on the operators of semiconductor factories if these expensive equipment had to build different control programs for each equipment. Fortunately, most semiconductor equipment vendors are implemented to control equipment according to the SECS (SEMI Equipment Communications Standard) / GEM (The Generic Model for Communications and Control of Manufacturing Equipment) standard defined by the SEMI Association [1,2]. When equipment vendors supply equipment, they guide how to

implement controls based on these SECS/GEM standards and then develop and implement equipment control programs based on these standards [1,2].

South Korea is one of the world's leading semiconductor producers, along with the United States, Taiwan, and Japan, and there are famous global semiconductor manufacturers such as Samsung Electronics and SK Hynix in Korea. However, the country is facing a growing demand for skilled developers due to the expansion of the IT industry and a rapidly declining birth rate, making it more difficult to recruit young developers. In addition, job turnover is more frequent compared to the past, making it difficult for the IT industry to retain skilled developers. To develop a semiconductor equipment control program, it is essential to understand the SECS/GEM standard and programming language and to have experience in developing how to control and interface with equipment based on it. However, it is tough to retain developers who meet these conditions in reality. Therefore, as a way to alleviate this kind of social problem, we thought that if it is possible to generate semiconductor facility control programs using generative artificial intelligence automatically, it would be possible to develop these without understanding SECS/GEM standards and programming languages and lacking development experience in controlling and interfacing equipment.

In the past, tools that automatically generate code based on SECS/GEM standards have been developed and used. These code generation tools predefine which specification is converted to which message based on rules and automatically generate the required message in code accordingly to use the message as a template. In practice, developers modified these code templates to suit the target equipment, customized them for the desired functions, and completed them as end-use code. In other words, these automatic code generation tools created a template according to the SECS/GEM specification that needs to be modified and supplemented with detailed information to modify the finalized code [3]. On the other hand, in the case of code generation using generative AI, automatic code generation according to SECS/GEM

† These authors contributed equally to this work.

specification is expected to generate code similarly based on the basically provided and previously developed code, so the parts that developers need to modify are expected to be much smaller than those of automatic code generation tools.

The paper is organized as follows:

Section 2 reviewed the techniques for SECS/GEM standards and specification, automation software architecture, automatic SECS/GEM code generation, and the LLM customization method. Section 3 describes the proposed method, "Generative AI-based Optimized Code Generation for Semiconductor Equipment Interfaces". In Section 4, we conducted four experiments and explained the experimental results. The conclusions and future research directions are included in Section 5.

II. RELATED WORK

A. SECS/GEM Standards and Specification

SEMI (Semiconductor Equipment and Material International) association has more than two thousand companies as members, and the SEMI association defines SECS/GEM standards. SECS-1, the most basic of the SECS/GEM standards, refers to the physical layer based on RS-232 cables and was defined in 1978 [1]. Later, the SECS-2 protocol was established in 1982 and divided into Stream and Function for communication between equipment and host on top of SECS-1 [1]. The SECS-1 and SECS-2 protocols are commonly referred to as the SECS standard. Later, the GEM (Generic Equipment Model), which enhanced the monitor function and data collection function for equipment, was established in 1992, and the HSMS (High-Speed SECS Message Service) based on the TCP/IP protocol was established in 1994 [1]. The SECS/GEM standard facilitates the construction of automation and smart factories by defining equipment control protocols for various functions as standards to enable equipment control and data collection from an automation perspective.

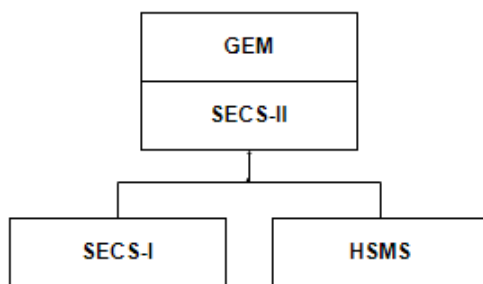


Figure 1. SECS/GEM Architecture [2]

B. Automation Software Major Architecture

The three essential elements for factory automation and smart factory are the manufacturing equipment that runs the process, the EIS (Equipment Interface Server) that controls the equipment, and the manufacturing operation system MES (Manufacturing Execution System) that operates the

manufacturing. Most equipment includes a PLC (Programmable Logic Controller) unit, a simple PLC device. EIS may communicate with this PLC unit, but the interface method is inefficient. An additional controller is required to manage the operation and equipment data to support more efficient system operation. The disadvantage of using such a controller is that it increases the cost of the equipment, but the advantage is that it provides much more efficient control capability. The equipment must include these controllers to support the SECS/GEM standard.

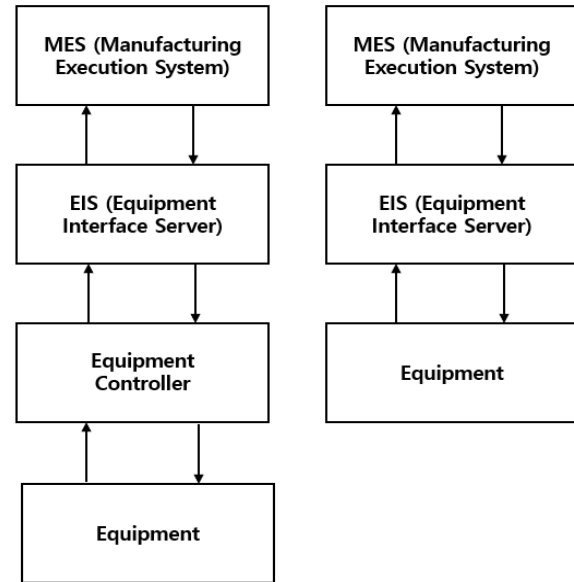


Figure 2. Types of Elements for Factory Automation and Smart Factory

C. Automatic Generation of SECS/GEM Code

Automatic generation of SECS/GEM standard code has been attempted in the past and is still heavily utilized today. If you define the code according to the SECS/GEM standard in advance and select Stream and Function on the screen, it will automatically generate the SECS/GEM code according to the defined standard. The generated code is a kind of code template, and developers can customize it based on the template to meet the detailed specifications of the equipment and the detailed scenario of the targeted task to complete the final code [3]. Smart factory solution provider Thirauteh, Ltd has a product called SECS-GEM Builder that automatically generates SECS/GEM code in this way [4]. SECS-GEM Builder consists of HSMS communication Driver Library and SECS Message Editor/Tester [4]. If you create an SECS/GEM Message structure with the Editor and save it as a file, the user program can read it and generate related code [4].

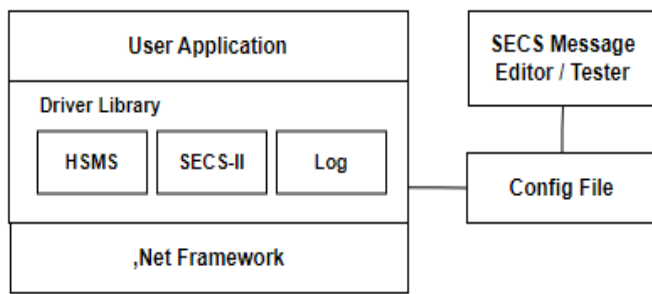


Figure 3. SECS-GEM Builder Architecture [4]

D. LLM (Large Language Model) Customization Method

LLM (Large Language Model) is a deep learning model that finds a sentence's most natural word arrangement. It works by predicting the next word given the previous words in a sentence, and it learns a large amount of language data to do so. LLM can be divided into upstream and downstream tasks, where the upstream task is to learn broadly by training a large corpus, and the downstream task is to learn deeply to solve a specific problem [4]. Therefore, LLM is a deep learning method that pre-trains a large amount of language data and applies it by transfer learning. Hallucination and controllability are major risks when utilizing LLM. There are methods used to improve and complement these risks: Fine Tuning, In-context Learning / Prompt Engineering, and RAG (Retrieval Augmented Generation) [5].

1) Fine-tuning: Downstream tasks are characterized by using all the data to update the entire model. On the downside, the larger the language model, the more expensive it is to update the entire model. Due to these drawbacks, prompted tuning and in-context text learning are gaining traction [6].

2) In-context Learning: This method uses only a portion of the downstream task data to perform the downstream task without updating the model. There are three main methods of in-context learning: Zero-Shot Learning, One-Shot Learning, and Few-Shot Learning [7].

3) RAG (Retrieval Augmented Generation): RAG optimizes the output of large language models by consulting a trusted knowledge base outside of the training data source before generating responses [8].

III. PROPOSED METHOD

This paper proposes a method to automatically generate SECS/GEM code controlling equipment for equipment interface using LLM. The existing SECS/GEM codes that control the facility are stored in the vector database, and when the user requests code generation through Prompt, the desired SECS/GEM code is generated through LLM and RAG stored in the database. Due to much customization work, fine-tuning requires many computing resources, so we decided to use relatively light and accessible low-cost RAGs.

We converted the code and related documentation into PDF format, loaded the PDF file into the LangChain-based service,

and embedded it in the Vector Store. When a user makes a query, it first searches for information in the Vector Store based on the query, combines the search results and the query, queries the LLM service, and generates a final answer in the LLM service to respond. Figure 4 shows the architecture of the SECS/GEM code generator.

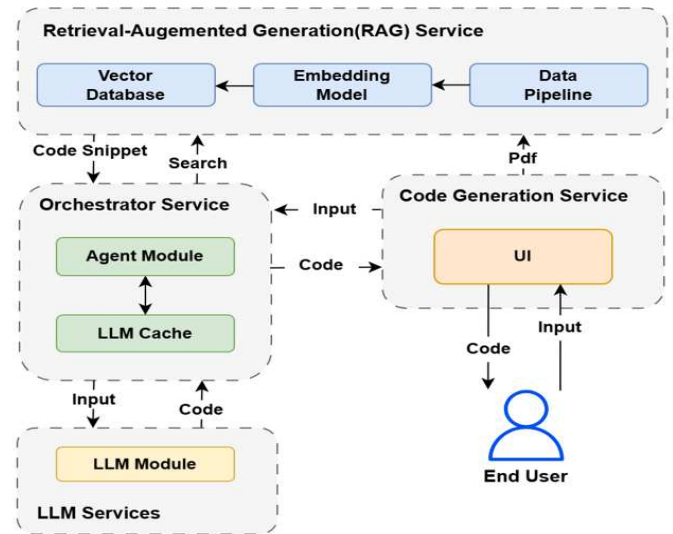


Figure 4. SECS/GEM Code Generator Architecture

In this proposal, there are two main pressures. The first is the data acquisition, categorization, and storage part, and the second is the user request for code generation, and the code is automatically generated. Figure 5 describes securing, categorizing, and storing data, and Figure 6 shows the process of requesting code generation and generating code automatically.

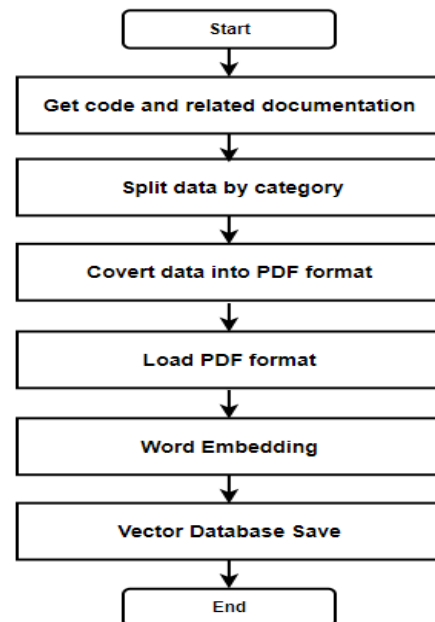


Figure 5. Processes to build RAG

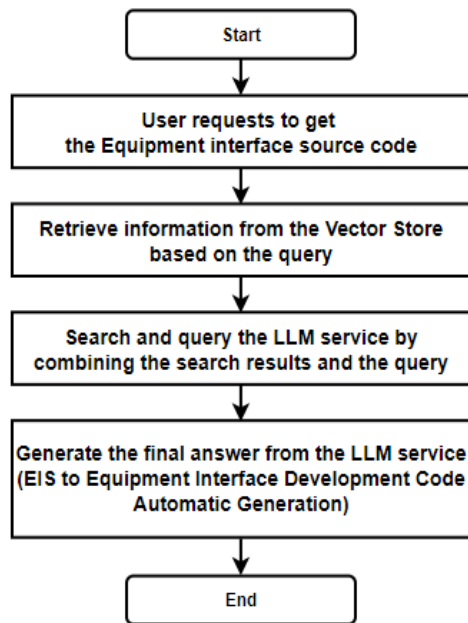


Figure 6. Processes to generate code

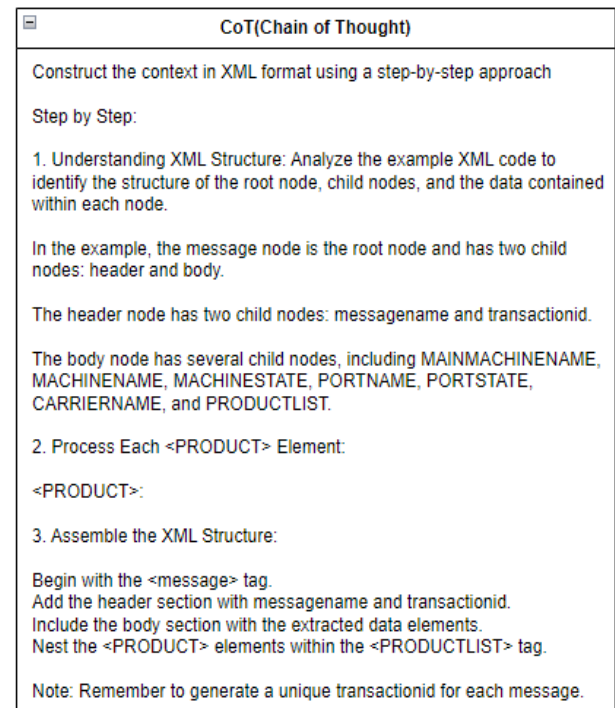


Figure 7. Prompt Template used in the experiment

IV. EXPERIMENT AND RESULTS

A. Experimental Environment and Methods

For this experiment, the SW configuration is minimized to the essential elements of the architecture proposed in section 3. The HW environment is configured as a server-based environment, and the server's specifications are in Table 1.

TABLE 1. HARDWARE SPECIFICATION

Item	Specification
CPU	Intel(R) Core(TM) I7-9700
RAM Size	43 GB
Graphic Card	Nvidia Geforce RTX 2080-Ti

The experiment was conducted to automatically convert SECS messages into XML-formatted code for actual system development. The following steps were taken for this experiment.

- We obtained the code to convert SECS messages to XML and loaded it into PDF. After embedding it in Word, we saved the loaded information in Vector DB.
- The user wrote the necessary prompts in detail to automatically generate the code so that the desired code could be generated.
- When a user asks for a request, it collects the relevant information from the Vector DB. It queries LLM with the prepared prompts to automatically generate XML codes for SECS messages.

The prompts we used in our experiment are shown in Figure 7 below.

Figure 8 shows an example of SECS message, and Figure 9 shows the XML translated code.

```

<L [3]
  <U4 [1] DATAID '0' >
  <U4 [1] CEID '1' > ((Event Name: JobProcessStarted)
  <L [3] L 2
    <L [2]
      <U4 [1] RPTID '1'
      <L [5]
        <A [7] MACHINENAME 'EQP001' >
        <A [6] MACHINESTATE 'NORMAL' >
        <A [4] PORTNAME 'PORT01' >
        <A [9] PORTSTATE 'EXECUTING' >
        <A [8] CARRIERNAME 'CST01' >
      <U4 [1] RPTID '100'
      <L [3]
        <L [2]
          <A [PRODUCTNAME] 'PD001' >
          <L [11]
            <A [10] LOTNAME LOT001' >
            <A [1] POSITION '1' >
            <A [16] PRODUCTREQUESTNAME 'PR001' >
            <A [3] LAYERNAME '1st' >
            <A [7] PROCESSOPERATIONNAME 'OP001' >
            <A [19] PRODUCTSPECCNAME 'SPEC01' >
            <A [11] MACHINERECIPENAME 'RCP001' >
            <U2 [1] PRODUCTSTATE '0' >
            <A [2] JUDGEMENT 'OK' >
            <A [1] INSPECTIONBITFLAG '0' >
            <A [1] PRERUNFLAG 'N' >
          
```

Figure 8. Example of SECS Message

```

<message>
<header>
<messagename>JobProcessStarted</messagename>
<transactionid>1001</transactionid>
</header>
<body>
<MACHINENAME>EQP001</MACHINENAME>
<MACHINESTATE>NORMAL</MACHINESTATE>
<PORTNAME>PORT01</PORTNAME>
<PORTSTATE>EXECUTING</PORTSTATE>
<CARRIERNAME>CST01</CARRIERNAME>
<PRODUCTLIST>
<PRODUCT>
<PRODUCTNAME>PD001</PRODUCTNAME>
<LOTNAME>LOT001</LOTNAME>
<POSITION>1</POSITION>
<REQUESTNAME>PR001</REQUESTNAME>
<LAYERNAME>1st</LAYERNAME>
<OPERATIONNAME>OP001</OPERATIONNAME>
<SPECNAME>SPEC01</SPECNAME>
<RECIPENAME>RCP001</RECIPENAME>
<STATE>0</STATE>
<JUDGEMENT>OK</JUDGEMENT>
<INSPECTIONFLAG>0</INSPECTIONFLAG>
<PRERUNFLAG>N</PRERUNFLAG>
</PRODUCT>

```

Figure 9. Example of Automatic Generated XML Code

B. Metrics for Experiments

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a performance evaluation metric for text summarization models. ROUGE is good for evaluating the performance of natural language generation models such as automatic text summarization and machine translation. It calculates the performance score by comparing the summaries or translations generated by the model with a human reference. Various ROUGE metrics include ROUGE-N (ROUGE-1, ROUGE-2), ROUGE-L, ROUGE-W, ROUGE-S, and ROUGE-SU [9, 10]. ROUGE-N and ROUGE-L are used as metrics in this paper.

- ROUGE-N: It measures the n-gram overlap between the system and the reference summary. For instance, ROUGE-1 refers to the overlap of unigrams (a single word), and ROUGE-2 refers to the overlap of bigrams (two continuous words) [11].
- ROUGE-L: It is based on the length of the LCS (longest common sequence). It calculates a weighted harmonic mean (F-measure) that combines precision and recall scores [11].

(1), (2), and (3) show the equations of ROUGE-N and ROUGE-L [11].

$$\text{ROUGE-1} = \frac{\sum_{n\text{-gram} \in \text{reference}} \min((A), (B))}{\sum_{n\text{-gram} \in \text{reference}} \text{count}(n\text{-gram in reference})} \quad (1)$$

- (A) : count(n-gram in reference)
 (B) : count(n-gram in generated)

$$\text{ROUGE-2} = \frac{\sum_{\text{bigram} \in \text{reference}} \min(C), (D))}{\sum_{\text{bigram} \in \text{reference}} \text{count}(\text{bigram in reference})} \quad (2)$$

(C) : count(bigram in reference)

(D) : count(bigram in generated)

$$\text{ROUGE-L} = \frac{\text{LCS}(\text{reference, generated})}{\text{length}(\text{reference})} \quad (3)$$

Where:

- n-gram, bigram: Represents a unigram or bigram in the reference or generated text.
- count(x): Refers to the number of times x appears in the text.
- min: Takes the minimum value between the reference and generated n-gram/bigram counts.
- LCS(reference, generated): Refers to the length of the longest common subsequence between the reference and generated text.
- length(reference): Denotes the total length of the reference text.

Table 2 below shows the evaluation criteria for the ROUGE-N and ROUGE-L metrics for the experimental results [11].

TABLE 2. ROUGH SCORE SCALE

ROUGE Metric	Excellent	Good	Moderate
ROUGE-1	0.5+	> 0.5	0.4 ~ 0.5
ROUGE-2	-	> 0.4	0.2 ~ 0.4
ROUGE-L	-	≤ 0.4	0.3 ~ 0.4

C. Experimental Results

The experiment was conducted four times, and the similarity values ROUGE-1, ROUGE-2, and ROUGE-L were checked by comparing the generated code with the prepared finished code to verify the generated code. Table 2 shows the ROUGE-1, ROUGE-2, and ROUGE-L values for the four experiments and the average value of each.

TABLE 3. SUMMARY OF EXPERIMENTAL RESULTS

Data Set #	ROUGE-1	ROUGE-2	ROUGE-L
1	0.66	0.42	0.60
2	0.79	0.68	0.79
3	0.69	0.56	0.69
4	0.62	0.47	0.61
Mean	0.69	0.53	0.67

The experimental results show that the ROUGE-1 values are greater than 0.5, which satisfies the highest “Excellent” criteria, and the ROUGE-2 and ROUGE-L metrics are greater than 0.4, which satisfies the highest “Good” criteria. This

implies that the generated code is quite similar to the finished code.

D. Experimental Results Analysis

We experimented to verify the validity of the proposed method. To experiment with generating SECS messages into XML code, which is developer code, we stored the completed XML code in the database in advance and configured prompt template to handle the considerations for automatically generating SECS messages into XML code. The system is configured so that when a user enters a sentence for a code request, the system first retrieves and checks the relevant information from the stored database and requests the connected generative AI to generate the code using the configured prompt template. The experiment was conducted four times, and the evaluation metrics ROUGE-1, ROUGE-2, and ROUGE-L were checked. According to the predefined evaluation criteria, the similarity was confirmed to be good.

V. CONCLUSIONS

The topic of this paper is to propose a method to automatically generate developer code for semiconductor equipment interfaces using generative AI. For this purpose, the data and information required for automatic code generation should be stored in the database RAG, and a prompt template should be configured to establish detailed procedures and inquiries so that users can query the generative AI and automatically generate the developer's code. When a user requests developer code, the proposed environment first retrieves the information related to the request from the stored data and uses the prepared prompt to query the generative AI with the retrieved information to generate the desired developer code automatically.

We have shown that generative AI can be used to automatically generate code for specific standards and that it is possible to utilize generative AI in no-code or low-code areas. It is expected that such automatic code generation can be leveraged with technical documentation and deliverables to revolutionize the efficiency of development work. Future research is interested in applied technologies for autonomization, and we would like to study information generation based on documents and related information to replace tasks that require specialized knowledge or experience by humans.

REFERENCES

- [1] Shams A. Laghari, Selvakumar Manickam, and Shankar Karuppayah, "A Review on SECS/GEM: A Machine-to-Machine (M2M) Communication Protocol for Industry 4.0", *International Journal of Electrical and Electronic Engineering & Telecommunications*, vol. 10, Mar, 2021.
- [2] R Dongbao Jiao and Xiudong Peng, "Development of semiconductor communication software device based on SECS/GEM protocol", *7th International Conference on Information Science, Computer Technology and Transportation*, 2022.
- [3] Zhiqian Liang, SECS/GEM message discovery for rapid equipment integration. Master's Thesis, Nanyang Technological University, Nanyang, China, 2007.
- [4] THiRA-UTECH Homepage Solution Introduction [Online]. Available: <https://thirautech.com/gallery/>
- [5] Cheonsu Jeong, "Generative AI service implementation using LLM application architecture: based on RAG model and LangChain framework", *Journal of Intelligence and Information Systems*, vol. 29, Dec, 2023.
- [6] Biao Zhang, Zhongtao Liu, Colin Cherry, and Orhan Firat, "When Scaling Meets LLM Finetuning: The Effect of Data, Model and Finetuning Method", *12th International Conference on Learning Representations*, 2024.
- [7] Ohad Rubin, Jonathan Herzig, and Jonathan Berant, "Learning To Retrieve Prompts for In-Context Learning", *2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2022.
- [8] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang, "Retrieval-Augmented Generation for Large Language Models: A Survey", Dec, 2023.
- [9] Chin-Yew Lin, "ROUGE: A Package for Automatic Evaluation of summaries", *The Workshop on Text Summarization Branches Out*, 2004.
- [10] WIKIPEDIA ROUGE (metric). [Online]. Available: [https://en.wikipedia.org/wiki/ROUGE_\(metric\)](https://en.wikipedia.org/wiki/ROUGE_(metric))
- [11] What is the ROUGE Score (Recall-Oriented Understudy for Gisting Evaluation)? [Online]. Available: <https://klu.ai/glossary/rouge-score>

HyeokSoo Lee received a M.S. in Computer Science from Fairleigh Dickinson University, New Jersey, US in 1995. He is working toward a Ph.D. in smart factory convergence at Sungkyunkwan University, Suwon, Korea. He is currently an employee as a head of the Research & Development team of THiRA-UTECH Co. Ltd. His research interests are in Smart Factory, Reinforcement Learning, Deep Learning, Robotics, and Data Analysis.

Minyeol Yang received a M.S. degree in Smart Factory Convergence from Sungkyunkwan University, Suwon, South Korea, in 2024. He is currently an employee as a research engineer of the Research & Development team of THiRA-UTECH Co. Ltd. His research interests are in Smart Factory, Industrial AI, Deep Learning, and Generative AI.

Jongpil Jeong received a bachelor's degree in engineering from Sungkyunkwan University, Suwon, South Korea, and a master's and Ph.D. degree in computer engineering from Sungkyunkwan University in 2003 and 2008, respectively. He has been with Sungkyunkwan University since 2008 and has been an Associate Professor with the Department of Smart Factory Convergence at Sungkyunkwan University since 2016. He is the principal investigator of MAKE UNIC, a key support area for smart manufacturing at Sungkyunkwan University. His research interests include smart factory, industrial AI, anomaly detection, manufacturing data analysis, AI-based fault diagnosis and prediction, 5G-based smart manufacturing, industrial IoT applications, AI platforms, cloud platforms, and industrial security.