Novel Preprocessing Technique for Data Embedding in Engineering Code Generation Using Large Language Model

Yu-Chen Lin*, Akhilesh Kumar†, Norman Chang†, Wenliang Zhang†,
Muhammad Zakir†, Rucha Apte†, Haiyang He†, Chao Wang†, Jyh-Shing Roger Jang*
* National Taiwan University, Taipei, Taiwan
{r11922035, jang}@csie.ntu.edu.tw
† Ansys, Inc., San Jose, Califonia, USA
{Akhilesh.Kumar, Norman.Chang, Wenliang.Zhang, Muhammad.Zakir, rucha.apte, haiyang.he, chao.wang}@ansys.com

2024 IEEE LLM Aided Design Workshop (LAD) | 979-8-3503-7608-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/LAD62341.2024.10691715

***Abstract*—We introduce four principal contributions to augment the capabilities of Large Language Models (LLMs) in generating domain-specific c ode: (i) l everaging L LM-based data splitting and data renovation techniques to refine t he semantic representation within the embedding space; (ii) proposing an effective method for refactoring existing scripts, enabling the generation of new and high-quality scripts with the aid of LLMs; (iii) developing the Implicit Knowledge Expansion and Contemplation (IKEC) Prompt technique; and (iv) showcasing the efficacy o f o ur d ata p re-processing a pproach t hrough a case study using engineering simulation software RedHawk-SC. Our contributions collectively advance the Retrieval-Augmented Generation (RAG) framework, enabling more relevant and precise information retrieval. An arena-style evaluation by 28 domain experts and 182 votes confirms t he s ignificant ef fectiveness of our methods. Notably, our approach achieves up to 1.43 times the improvement in code generation for MapReduce applications compared to the Chain-of-Thought (CoT) technique.**

***Index Terms*—Large language models (LLMs), domain-specific, c ode g eneration, d ata p reprocessing, d ata s plitter, data renovation, domain-specific, R etrieval-Augmented Generation (RAG), prompt engineering, MapReduce, RedHawk-SC (RH-SC)**

I. Introduction

Large Language Models (LLMs) have become instrumental in the progression of Natural Language Processing (NLP), showcasing remarkable versatility in diverse applications, including domain-specific q uestion a nswering a nd c ode generation [1], [2]. Notable advancements have led to state-of-the-art models like GPT-4 [3], Claude, and Mistral, with the latter integrating a "Mixture-of-Experts" (MoE) technique [4], [5]. The Chatbot Arena, featuring an extensive dataset exceeding 100K pairwise votes, serves as a benchmark for evaluating LLMs [6].

The Retrieval-Augmented Generation (RAG) framework is being widely used to enhance LLMs in specialized domains by retrieving factual content from external databases, thereby mitigating hallucinations and improving performance [7], [8]. This method involves parsing data into character-limited, overlapping segments and transforming them into embeddings for vector-based retrieval, endowing LLMs with precise domain-specific knowledge.

Code generation is a key application of LLMs, yet it poses various challenges in domain-specific engineering tasks, such as those on the Ansys RedHawk-SC (RH-SC) platform. Here, users often struggle with creating MapReduce [9] Python scripts due to the complexity of the tasks and a lack of coding expertise. Automating this script generation through natural language instructions could significantly boost productivity and enhance the user experience.

Moreover, applying LLMs to generate MapReduce [9] Python code for the Ansys RedHawk-SeaScape platform requires not only a profound understanding of RH-SC architecture but also expertise in intricate circuit design. The lack of comprehensive technical documentation and scarce Electronic Design Automation (EDA) online resources make it challenging for LLMs to acquire the necessary domain knowledge.

While ChatEDA [10] attempts to address these issues by fine-tuning Llama2 [11] and implementing Self-Planning Code Generation [12] on open-source EDA tools, this approach is costly. Other methodologies like TestPilot [13] and VeriGen [14] have shown limitations in generating code within our specialized context.

To overcome these challenges, this paper presents novel methodologies based on RAG without requiring any kind of pre-training or fine-tuning of the LLM. Our Data Splitter and Data Renovation techniques refine semantic text segmentation and enrich paragraph content, circumventing the disarray typical of direct document extraction. These methods significantly improve RAG's embedding accuracy for more effective information retrieval, thereby enhancing overall performance beyond traditional character count segmentation techniques.

Furthermore, we also propose Script Augmentation to reconstruct existing scripts into new ones and introduce Implicit Knowledge Expansion and Contemplation (IKEC), a novel prompting technique. Our experiments combine IKEC with the Chain-of-Thought (CoT) approach [15] to explore potential performance improvements.

979-8-3503-7608-1/24/$31.00 ©2024 IEEE

By implementing these data preprocessing and self-planning code generation prompting strategies [12], our approach generates scripts that align with user requirements. Additionally, we established an internal Code Generation Arena at Ansys to solicit expert feedback. With over 180 expert votes, the results affirm that our Splitter and Renovation methods significantly enhance the quality of code generation.

Our contributions provide practical solutions for domain-specific challenges faced by LLMs, particularly in the EDA industry, where traditional approaches fall short. The findings have implications not just for code generation but also for broader LLM applications where data quality and contextual understanding are critical.

II. Methodology

As depicted in Fig. 1, our methodology generates scripts for RH-SC based on user requirements. The process commences with extracting text from documents and utilizing an LLM for semantic segmentation, focusing on restoring improperly formatted content and yielding several distinct paragraphs through post-processing, then using the standard RAG method to renovate each paragraph individually.

Concurrently, a random selection of two scripts from a pool of 14 is reconstructed using the RAG method, a process repeated 14 times to produce a variety of scripts. These datasets are subsequently transformed into Targeting Chunks Vector through the standard RAG preparation process to serve as references for the subsequent RAG operations.

User requirements serve as the query input, which, when paired with an enhanced RAG mechanism, initiates the generation of an initial script structure (Framework of Comments). This framework then guides the generation of complete scripts, acting as a new query within the improved RAG system. Our iterative and progressive approach to script generation is inspired by the ChatEDA [10] paper, although we employ RAG technology instead of fine-tuning.

A. Data Splitter

The RAG method's effectiveness hinges on the relevance of the text as determined by the embedding computations. Conventional RAG techniques, which often segment text by character count, may yield chunks that lack thematic focus. Consequently, these chunks produce embeddings that inadequately represent the target topic, thus reducing the likelihood of retrieving high-quality textual content.

Our Data Splitter addresses this by semantically segmenting text, focusing on meaningful units such as API functions and concepts, while preserving the original text format and rectifying formatting issues. We segment the text into fixed-page units, allowing the LLM to determine paragraph completeness. Incomplete segments are left unclosed, facilitating seamless post-processing to assemble complete paragraphs.

As illustrated in Fig. 2, the Data Splitter precisely partitions the extracted document content into distinct, focused segments, effectively resolving formatting problems that may arise. The task of determining where to split the text is akin to a "binary classification problem," which is relatively straightforward for an LLM. Hence, we utilize an LLM for this task, forgoing the use of the RAG technique in this instance.

B. Data Renovation

Data Renovation addresses the challenge presented by the typically concise nature of technical documents. It enables LLMs to methodically enrich each paragraph with well-understood knowledge, which enhances the embeddings—even if the knowledge is already familiar to the LLM. We leverage the RAG framework provided by LlamaIndex to supply the original documents and scripts with supplementary content that is firmly rooted in the context of the source material, thereby ensuring the reliability of the information.

As depicted in Fig. 2, each paragraph processed by the Data Splitter is subsequently renovated in sequence. It is noteworthy from the figure that the renovation meticulously incorporates additional details about the key terms mentioned in the content.

C. Script Augmentation

Obtaining scripts with detailed comments is often a challenging endeavor. In scenarios where only a limited number of such scripts are available, if we prompt the LLM to generate a "brand new" script based on a reference script, the result is frequently a disarrayed output due to the LLM's limited expertise. As demonstrated in Fig. 2, we selected two scripts at random and employed the RAG framework to encourage "significant structural changes" (reconstruction) in the creation of new scripts with clear, task-specific purposes. This approach ensures that the generated scripts are of a satisfactory quality. By repeatedly generating structurally diverse scripts, we can enrich the pool of reference material for script augmentation.

D. Implicit Knowledge Expansion and Contemplation (IKEC)

The IKEC technique, a novel approach developed in our work, aims to prompt the LLM to leverage its own repository of knowledge to internally expand and enrich the content about which it is most confident. This process involves deliberate and deep contemplation by the LLM before arriving at the final output. Additionally, we experimented with integrating this technique with the CoT process, driven by curiosity about what internal prompts might aid the LLM's performance.

III. Experiment

We focus on the generation of code for the RH-SC engineering simulation platform. To create a standard test dataset, we enlisted domain experts to design 20 User Requirements and their corresponding gold scripts. For the evaluation, we designed an in-company Code Generation Arena to conduct an ablation study and assess the effectiveness of different methods. We invited 28 RH-SC experts from Ansys to participate in the arena evaluations.

In each session, an expert was presented with a task randomly selected from the 20 User Requirements and two corresponding scripts generated by two different combinations chosen at random from a predefined set of eight combinations.

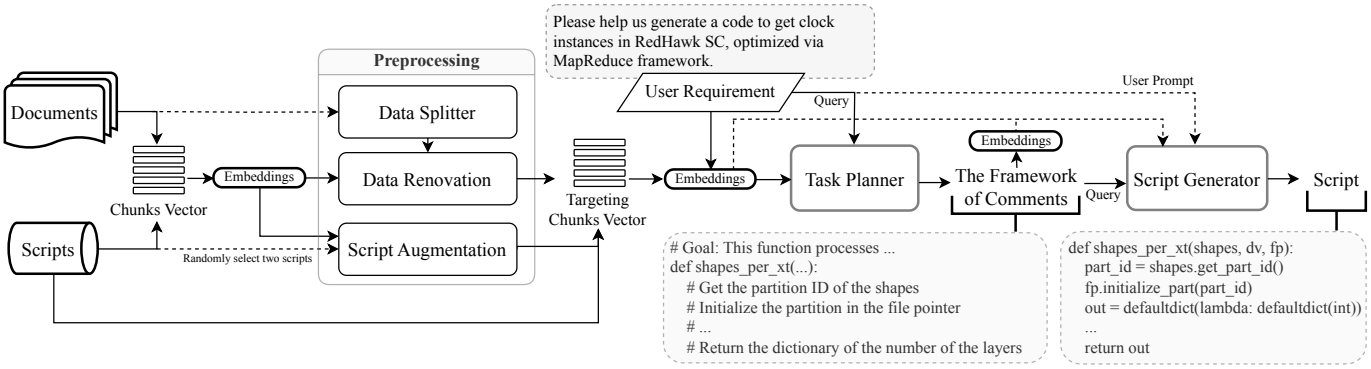

Fig. 1. **Flowchart of the proposed methodological framework**: This figure presents the main techniques based on LLMs, each to be elaborated upon within the paper. The conversion of text from all document types and scripts into Chunks Vector is standardized, involving segmentation into 1024-character chunks and subsequent embedding using the OpenAI text-embedding-ada-002 API. For documents with excessive length, a script is employed to divide the text into segments of three pages each, which are then processed individually and subsequently reassembled using a predefined format to yield the complete and accurate content. Documents are categorized into three types: (1) Manual, outlining RedHawk-SC usage and concepts; (2) API Documentation, with comprehensive function definitions, descriptions, and usage; and (3) MapReduce Documentation, detailing the application of MapReduce for simulation acceleration in RedHawk-SC. The Scripts comprise 14 expert-composed scripts complete with task narratives, objectives, and comments tailored for RedHawk-SC simulations. The Framework of Comments depicts code outlines annotated by LLMs. All methods, except for the Data Splitter, are based on the RAG approach and incorporate the use of embeddings.

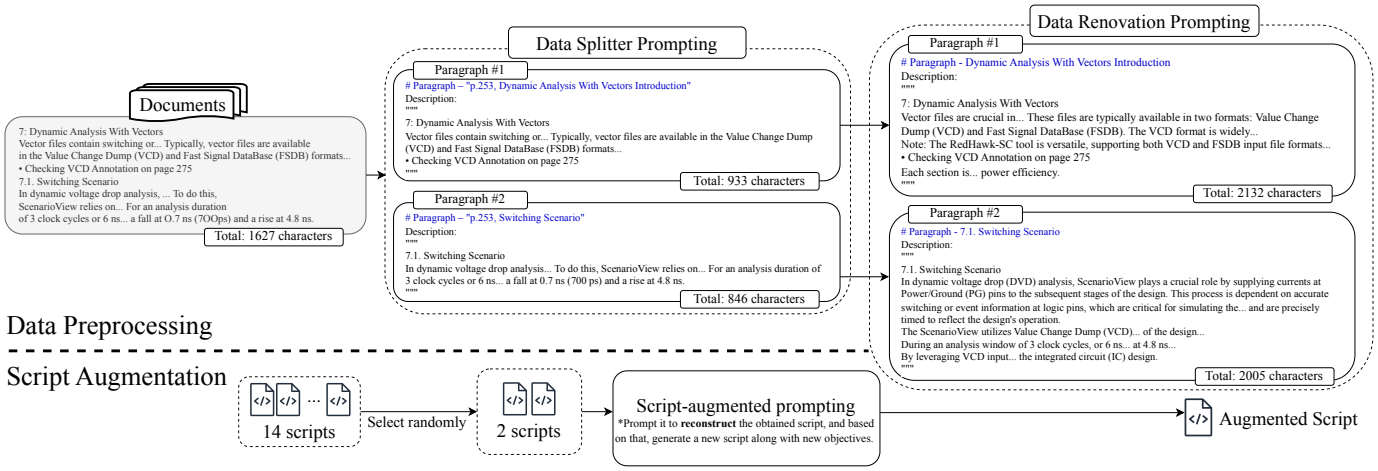

Fig. 2. **Prompt workflow example illustration**: This figure depicts the Data Preprocessing phase, where a "Data Splitter" segments text from documents, preserving the original content while reformatting it suitably. Subsequent "Data Renovation" processes are applied to reconstruct the segmented content accurately. Ellipses ("...") indicate the omission of large text portions for brevity. The Script Augmentation section illustrates the selection of two scripts from the original set of 14 to form part of the Prompt, stimulating the generation of new Scripts. Iterating this procedure yields a variety of Scripts.

The experts were blinded to the actual combinations used and had to decide which script was better, declare a tie, or choose 'undecided' if unable to decide.

The Elo Rating System, an effective approach for determining the relative ranking among multiple contenders, was employed to evaluate the results, as shown in TABLE I. This system has been widely applied in recent years for LLM evaluations [6], [16].

The top eight combinations listed in the TABLE I were selected for the Ablation Study, while the remaining eight were not included in the arena assessment. These relative rankings enabled us to explore the impact of including or omitting specific components in our approach.

In the original RAG framework, each chunk was set to contain 1024 characters by default. In our small experiment on chunk character counts, we found that after applying our Data Splitter, the average chunk size was reduced to 317 characters, suggesting that the original chunks contained superfluous information that could adversely affect the embedding process. Subsequent application of Data Renovation increased the average chunk size to 1165 characters, which is 3.67 times larger than that of the splitter-processed chunks. Following this, we will analyze the results of each component.

A. Data Splitter

As shown in TABLE I, the introduction of the Data Splitter (ID 20) resulted in a significant Elo score increase of 63 points over ID 11 ($p \approx 10^{-317} < 0.01$). This suggests that ID 20's script avoided an over-reliance on MapReduce Information and instead referenced the script material more substantially.

TABLE I

REDHAWK-SC CODE GENERATION ANSYS ARENA (DOUBLE-BLIND, IN-COMPANY): THE EVALUATION INVOLVES 28 DOMAIN EXPERTS AND COLLECTS 182 VOTES. AN ELO RATING SYSTEM [6], [16] WITH PARAMETERS (K=16, SCALE=400, BASE=10, INIT_RATING=1000) SHUFFLES SUBMISSIONS 1000 TIMES, RECALCULATING SCORES EACH ITERATION TO DERIVE THE MEDIAN ELO SCORE, WHICH IS REPRESENTED IN THE "ELO" COLUMN. THE 2.5% AND 97.5% PERCENTILES CORRESPOND TO THE PERCENTILES OF THE ELO SCORES OBTAINED FROM 1000 SHUFFLES. ALL MODELS UTILIZE GPT-4-TURBO (2023-07-01-PREVIEW) WITH A CONTEXT SIZE OF 128K AND FOLLOW THE LLAMAINDEX RAG METHOD WITH A SIMILARITY_TOP_K PARAMETER OF 6. THE APPROACH SECTION LISTS FIVE METHODS, WITH "Y" DENOTING THEIR INCLUSION. THE REFERENCE AVERAGE INDICATES THE MEAN SOURCE PERCENTAGE ACROSS FIVE DATA TYPES IN THE RAG PROCESS. "UNIQUE ID," "SCRIPT AUGMENTATION," "RENOVATION," AND "(AUGMENTED) SCRIPT" ARE ABBREVIATED AS "UNI. ID," "AUG.," "RENO.," AND "(AUG.) SCRIPT," RESPECTIVELY.

	Arena Information			**Approach (Y/)**					**Reference Average (Source Count Percentage) (%)**				
Uni. ID	***Rank***	***Elo***	***(2.5%, 97.5%)***	***IKEC***	***CoT***	***Aug.***	***Splitter***	***Reno.***	***Manual***	***API***	***MapReduce***	***Script***	***(Aug.) Script***
19	1	1054	(-59, +57)	Y	Y		Y		7.09	6.67	0.84	85.42	-
12	2	1045	(-62, +59)			Y	Y	Y	1.67	14.59	0.00	37.92	45.83
9	3	1013	(-61, +59)			Y	Y		3.33	4.59	0.00	38.34	53.75
20	4	1010	(-60, +60)	Y			Y		6.67	5.42	0.42	87.50	-
5	5	994	(-57, +64)						19.58	27.08	42.08	11.25	-
14	6	980	(-57, +59)	Y		Y	Y	Y	1.25	12.09	0.00	35.42	51.25
8	7	955	(-57, +61)	Y		Y	Y		2.08	3.75	0.00	37.50	56.67
11	8	947	(-56, +59)	Y					14.58	22.50	52.50	10.42	-
6	-	-	-	Y	Y				16.67	17.92	57.08	8.34	-
15	-	-	-				Y		8.75	5.42	0.42	85.42	-
18	-	-	-				Y	Y	3.34	29.59	0.42	66.67	-
16	-	-	-	Y			Y	Y	4.59	27.50	0.84	67.09	-
17	-	-	-	Y	Y		Y	Y	5.00	30.00	0.84	64.17	-
7	-	-	-	Y	Y	Y			10.84	3.33	14.59	28.75	42.50
10	-	-	-	Y	Y	Y	Y		1.67	2.92	0.00	32.92	62.50
13	-	-	-	Y	Y	Y	Y	Y	0.42	11.25	0.00	35.84	52.50

Furthermore, the reduced percentages of Manual and API after splitting might imply that the RAG became more efficient at selecting truly relevant content.

B. Data Renovation

After Data Renovation, IDs 8 and 14 saw a significant Elo score increase of 25 points ($p \approx 10^{-62} < 0.01$), and IDs 9 and 12 exhibited a significant increase of 32 points ($p \approx 10^{-96} < 0.01$), as noted in TABLE I. Analyzing the reference material percentages, we noted a decrease in reliance on Scripts and Manuals and an increase in API references post-renovation. This shift suggests that Data Renovation positively impacts the RAG's reference selection process.

C. Script Augmentation

As shown in TABLE I, it led to a significant decrease in Elo score for ID 20 as compared to ID 8, with a loss of 55 points ($p \approx 10^{-286} < 0.01$). The reference material indicates a shift towards using Scripts, with 56.67% referencing Augmented Scripts. While the Augmented Scripts are of good quality, they are derivatives of the original Scripts and may contain redundant content. Additionally, the original expert-designed Scripts likely represent the highest quality. Despite the observed performance drop, the generated Scripts are still of high quality, warranting further investigation into their potential.

D. Chain-of-Thought (CoT)

From TABLE I, IDs 20 and 19 show that implementing the CoT technique [15] resulted in a significant Elo score increase of 44 points ($p \approx 10^{-162} < 0.01$). The reference material percentages remained relatively unchanged, suggesting the effectiveness of the CoT prompting technique itself. This finding aligns with the results reported in the CoT literature.

E. Implicit Knowledge Expansion and Contemplation (IKEC)

In TABLE I, a decrease in Elo scores was observed in several comparisons post-IKEC implementation: ID 5 and ID 11 by 47 points ($p \approx 10^{-213} < 0.01$), ID 12 and ID 14 by 65 points ($p = 0 < 0.01$), and ID 9 and ID 8 by 58 points ($p \approx 10^{-286} < 0.01$). After IKEC, there was a general decrease in reference percentages for Manuals, APIs, and Scripts, with an increased reliance on MapReduce and Augmented Scripts. This shift in reference material usage is identified as a key factor affecting performance.

IV. CONCLUSION

We introduce four notable contributions to improving RAG performance for LLMs in specific-domain problems: Data Splitter, Data Renovation, Script Augmentation, and IKEC. By semantically segmenting text and renovating content with high LLM confidence levels, these techniques facilitate the improvement of topic-focused embedding during the data retrieval process for RAG. The novel application of Data Splitter and Data Renovation techniques to enhance embeddings at the data source level is particularly innovative.

The effectiveness of these contributions has been validated through a comprehensive evaluation involving a panel of 28 domain experts and the analysis of 182 votes. The results demonstrate that the Data Splitter and Data Renovation methods notably enhance the quality of code generation for RHSC in MapReduce applications within specialized domains. Specifically, the improvement attributed to these methods is quantifiably greater than the CoT, with the Data Splitter method achieving an improvement that is 1.43 times that of the CoT prompting, and the Data Renovation method achieving an improvement that is 0.45 times that of CoT.

REFERENCES

[1] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, et al., "A survey of large language models," 2023.

[2] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, et al., "Large language models: A survey," 2024.

[3] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, et al., "Gpt-4 technical report," 2023.

[4] A. Eliseev and D. Mazur, "Fast inference of mixture-of-experts language models with offloading," 2023.

[5] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, et al., "Mixtral of experts," 2024.

[6] W.-L. Chiang, L. Zheng, Y. Sheng, A. N. Angelopoulos, T. Li, D. Li, et al., "Chatbot arena: An open platform for evaluating llms by human preference," 2024.

[7] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, et al., "Retrieval-augmented generation for knowledge-intensive nlp tasks," 2021.

[8] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, et al., "Retrieval-augmented generation for large language models: A survey," 2024.

[9] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[10] Z. He, H. Wu, X. Zhang, X. Yao, S. Zheng, H. Zheng, et al., "Chateda: A large language model powered autonomous agent for eda," in *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*. IEEE, 2023, pp. 1–6.

[11] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, et al., "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[12] X. Jiang, Y. Dong, L. Wang, Z. Fang, Q. Shang, G. Li, et al., "Self-planning code generation with large language models," 2023.

[13] M. Schäfer, S. Nadi, A. Eghbali, and F. Tip, "An empirical evaluation of using large language models for automated unit test generation," 2023.

[14] S. Thakur, B. Ahmad, H. Pearce, B. Tan, B. Dolan-Gavitt, R. Karri, et al., "Verigen: A large language model for verilog code generation," *arXiv preprint arXiv:2308.00708*, 2023.

[15] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, et al., "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24824–24837, 2022.

[16] Y. Bai, A. Jones, K. Ndousse, A. Askell, A. Chen, N. DasSarma, et al., "Training a helpful and harmless assistant with reinforcement learning from human feedback," 2022.

APPENDIX

A. Acknowledgements

I would like to extend my sincere gratitude to Ansys Inc. for their support and resources. Special thanks to Sean Harvey for assisting with model deployment, the Ansys RH-SC team for their evaluation efforts, and Jibin John for providing twenty scripts. I am also deeply grateful to Jun-You Wang for his long-term professional advice and to the Natural Language Processing (NLP) group of the Multimedia Information Retrieval Laboratory (MIRlab) at National Taiwan University for their regular discussions and support. Additionally, I appreciate the Tech New Stars Competition [1] organizing team for their efforts in filming the champion interview [2][3] and the assistance from Ansys's marketing team.

B. Additional Information

The links and information related to existing methods mentioned in this paper are provided here with corresponding footnotes.

- In Section I - Introduction, we mentioned a typical RAG technique that involves parsing data into character-limited, overlapping segments and transforming them into embeddings for vector-based retrieval, endowing LLMs with precise domain-specific knowledge [4]. The related link can be found here.
- In Section II - Methodology, we referred to our use of LlamaIndex [5]. The related link can be found here.

[1] https://www.tns-doit.tw/?menuid=14524&lgid=1&siteid=100624

[2] https://www.linkedin.com/feed/update/urn:li:activity:7196109502096547840/

[3] https://www.linkedin.com/feed/update/urn:li:activity:7135173172676489216/

[4] https://docs.llamaindex.ai/en/stable/api/llama_index.node_parser.SentenceSplitter.html

[5] https://docs.llamaindex.ai/en/stable/