國立臺灣大學電機資訊學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master's Thesis

基於大型語言模型的五項程式碼生成組件及其評估

Proposition and Evaluation of Five Constructive Components for Code Generation via Large Language Models

林育辰

Yu-Chen Lin

指導教授: 張智星 博士

Advisor: Jyh-Shing Roger Jang Ph.D.

中華民國 113 年 7 月

July, 2024

doi:10.6342/NTU202402417

國立臺灣大學碩士學位論文

口試委員會審定書

MASTER'S THESIS ACCEPTANCE CERTIFICATE
NATIONAL TAIWAN UNIVERSITY

基於大型語言模型的五項程式碼生成組件及其評估

Proposition and Evaluation of Five Constructive Components for Code Generation via Large Language Models

本論文係林育辰君（學號 R11922035）在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國 113 年 7 月 24 日承下列考試委員審查通過及口試及格，特此證明。

The undersigned, appointed by the Department of Computer Science and Information Engineering on 24 July 2024 have examined a Master's thesis entitled above presented by YU-CHEN LIN (student ID: R11922035) candidate and hereby certify that it is worthy of acceptance.

口試委員 Oral examination committee:

張智星　　李宏毅

（指導教授 Advisor）

系主任/所長 Director: 陳祝嵩

doi:10.6342/NTU202402417

doi:10.6342/NTU202402417

Acknowledgements

雙手置放鍵盤上，思緒隨指尖流轉。筆下的此日研究已告尾聲，多少時日飛逝，暮然回首，心中暖流湧然躍升。攢慶良久的謝意，躍動穿梭於寡寡數頁輕薄紙上。

起初，我因 KKBOX 實習經驗，在歌詞對位議題的啟發而毅然決然選擇加入 MIRlab，鑽研語音辨識領域。在此許多貴人相助，並結識志同道合的夥伴，並迎來多項人生的重大轉捩點。張智星老師、王鈞右學長、小龜等人，不停給予寶貴的建議和指引，尤其是在呈現以及表達皆有獨到的見解，並點醒了研究的盲點，使得我的 DAC 以及 LAD 發表之旅成行且順利。老師積極促進實驗室交流與傳承，仍籌辦職涯與聚餐或活動，使我能認識優秀的學長姐拓展視野。

王鈞右學長從我研究語音領域至今，與我侃侃而談這塊領域的完整概念以及指導，使我擁有完整的概念以及洞見，至今轉往大型語言模型的主題發展仍然受用，亦依舊給予發表以及任何研究相關事務的回饋以及建議，我十分感謝他。同時，我們與俊彥也不止歇地在語音領域努力，也在生活上彼此照應，他們是我在 MIRlab 的一盞明燈。威宇則擔任了我長達五年的學弟，我們在一年的時間內每週打半天桌球，使我不因研究頹靡身心，一年當中未曾生病。

去年十月我陷入研究瓶頸，而張智星老師鼓勵我參與科技新秀大賽，參賽前夕靈光乍現而想到這篇論文的研究方法，憑此順利奪冠。至此研究迅速發展，從

doi:10.6342/NTU202402417

專利再到國際研討會 (DAC, LAD) 獲得認可，在 Ansys 內部演講也贏得讚譽，既幸運又感激。

Ansys 提供優質的實習條件以及設備，並安排強大的團隊 Norman、Akhilesh、Wenliang 和 Zakir 等人，每週及時調整方向以及建議。Norman 長期以來幫了我很大的忙，使我研究能被看見並有很好的展示舞台；Akhilesh 則展現深厚的研究底蘊能適切反饋其洞見；Wenliang 艱辛創建了資料集還請整個團隊協助評估；Haiyang 在 fine-tuning 也有所貢獻。除上述人員外，還有 Chao, Rucha，也都是促成我論文不可或缺的一員。出國發表期間，Ansys 支持以及照應使我能恣意徜徉在研討會的薰陶中，並在所有發表上都順利成功，我們一同交流、分享和運動，度過愉快且充實的時光。

最後，我要感謝我的父母，他們使我成長至今。我與母親亦親亦友，每週分享生活的點滴，即便路途坎坷，我們一同面對。如今，他們以我為榮，並持續參與我的成長，為我的成就感到欣喜。在我求學期間，支持我大部分的決定以及看法，我由衷感謝。

doi:10.6342/NTU202402417

摘要

本文提出五項基於大型語言模型的程式碼生成組件，用於特定領域的腳本生成，並評估其有效性。貢獻：(i) 基於大型語言模型的語義分割 (Semantic Splitter) 以及資料翻新 (Data Renovation) 組件以改進資料語義的表示；(ii) 運用大型語言模型重構以產生高品質程式碼的組件 Script Augmentation；(iii) 提出提示技術隱形知識擴展與思考 (Implicit Knowledge Expansion and Contemplation, IKEC) 組件；(iv) 提出程式碼生成的流程，以五項組件漸進式生成工程模擬軟體 RedHawk-SC 的程式碼；(v) 評估不同參考資料型態之於程式碼生成的有效性。零樣本連鎖思維 (Zero-shot Chain-of-Thought, ZCoT) 為有效的提示技術，包括在五項建設性組件中，以利評估其餘組件之有效性。我們邀請 28 位領域專家透過競技場式評估蒐集 187 份成對比較結果以驗證前述組件之有效性，其中最佳組件於工程軟體 RedHawk-SC 上 MapReduce 程式碼生成表現達到 21.26% 的勝率提升，相較零樣本連鎖思維 6.68% 勝率提升顯著許多。

關鍵字：自然語言處理、大型語言模型、程式碼生成、資料嵌入、資料前處理、語義分割、資料翻新、腳本擴增、提示技術

doi:10.6342/NTU202402417

doi:10.6342/NTU202402417

Abstract

We propose five constructive components based on Large Language Models (LLMs) for domain-specific code generation and evaluate their effectiveness. The contributions are (i) Semantic splitter and data renovation for improved data semantic representation; (ii) Script augmentation for enhanced code quality; (iii) Implicit Knowledge Expansion and Contemplation (IKEC) prompting technique; (iv) A workflow using hierarchical generation for scripts in the engineering software RedHawk-SC; (v) An evaluation of different reference data types for code generation. We invited 28 domain experts to conduct an arena-style evaluation, collecting 187 paired comparisons to validate the effectiveness of those components. The best component achieved a 21.26% win rate improvement in MapReduce code generation performance for RedHawk-SC, significantly outperforming the 6.68% win rate improvement of the Zero-shot Chain-of-Thought (ZCoT).

Keywords: Natural Language Processing (NLP), Large Language Models (LLMs), Code Generation, Data Embedding, Data Preprocessing, Semantic Splitter, Data Renovation, Script Augmentation, Prompting Techniques

doi:10.6342/NTU202402417

doi:10.6342/NTU202402417

Contents

doi:10.6342/NTU202402417

doi:10.6342/NTU202402417

doi:10.6342/NTU202402417

doi:10.6342/NTU202402417

doi:10.6342/NTU202402417

doi:10.6342/NTU202402417

List of Figures

doi:10.6342/NTU202402417

doi:10.6342/NTU202402417

doi:10.6342/NTU202402417

doi:10.6342/NTU202402417

List of Tables

doi:10.6342/NTU202402417

doi:10.6342/NTU202402417

Chapter 1 Introduction

This chapter introduces the research topic, motivation, and contributions. It also provides an overview of the methodology and the application of the engineering tool RedHawk-SC (RH-SC) [1] platform used in this study. Subsequently, the contents of the following chapters are outlined.

1.1 Research Topic

Large Language Models (LLMs) [28, 41] have rapidly emerged across various fields, creating a significant impact and continuously demonstrating impressive potential. These models exhibit excellent capabilities for any concept they have been trained on. Conversely, their performance is suboptimal in domains they have not yet learned or where resources are relatively sparse, particularly in specific domains.

For example, in Electronic Design Automation (EDA), most circuit diagrams, simulation information, and even user manuals for individual tools are critical assets for major companies, leading to a scarcity of training data for LLMs in this field. Taking the engineering simulation tool RedHawk-SC used in this study as an example, LLMs are only familiar with a general overview, lacking detailed knowledge of writing scripts or any API

[1] https://www.ansys.com/products/semiconductors/ansys-redhawk-sc

doi:10.6342/NTU202402417

information, which significantly affects their application and performance in this domain.

Major companies often expend substantial human resources to collect data and fine-tune open-source LLMs with their own prepared data to achieve a certain level of performance, such as ChatEDA [14] and VeriGen [35]. However, for most users and technology companies other than tech giants, information security concerns necessitate the development of local LLM applications. Still, they find it challenging to bear the high costs of developing such tools.

Therefore, this study focuses on improving the data itself to enhance the performance of Retrieval-Augmented Generation (RAG) [11, 21], assisting specific-domain code generation to achieve satisfactory results, using RedHawk-SC as the practical demonstration platform. This approach can save significant resource demands while maintaining sufficient flexibility. For instance, you can directly use APIs provided by proprietary LLMs, such as GPT-4 [31], or achieve better performance on self-fine-tuned LLMs [1].

Users directly input their requirements, and scripts that meet their needs are generated using our proposed pipeline. The results leverage MapReduce [4] to assist in accelerating simulations. Users receive the resulting script directly, which they can use to operate RedHawk-SC to achieve the desired outcomes, providing commercial value. This is the focus of our research.

1.2 Motivation

Domain-specific code generation using RAG is one of the core technologies, and the quality of the retrieved relevant chunks significantly affects its performance. Current technological developments focus on enhancing model capabilities (e.g., Llama 3.1 [6],

doi:10.6342/NTU202402417

Gemma 2 [34]) and establishing high-quality data [27] or combining knowledge graphs to improve performance through the establishment of data relationships [7].

For Register Transfer Language (RTL) code generation, Verilog, which has relatively more resources, achieves an average simulation pass rate of 26.70% in the human-designed dataset VerilogEval [26] when scripts are generated using RAG [37]. This indicates that existing RAG techniques are still inadequate in effectively addressing this task. There has been less focus on processing the data itself to ensure that the chunks reflect the intended semantic themes, thereby improving the relevance of the retrieved chunks for RAG. By refining data processing to better capture semantic meaning, this approach can directly enhance RAG performance and further boost the effectiveness of other LLM-based methods, leading to more precise and efficient code generation in domain-specific applications.

1.3 Challenges

There are several challenges in using LLMs to generate code for specific domains such as EDA.

Firstly, technical documentation is often too concise, making it of limited use even when provided to LLMs for generating domain-specific code. Additionally, when extracting technical documents into plain text, various formatting issues such as encoding errors, line breaks, copyright information, and disordered content blocks can degrade performance.

Secondly, data are scarce in the EDA field. The EDA field encompasses numerous commercial software solutions with diverse functionalities. Due to the field's unique characteristics, such as commercial interests and data sensitivity, data is generally scarce. Even

doi:10.6342/NTU202402417

when datasets are available, the differences in tools and their licenses make it difficult to evaluate and compare them.

Thirdly, the domain knowledge is vast. When providing related product documents, existing RAG techniques struggle to supply appropriate reference materials, which is a critical factor affecting code generation performance.

Fourthly, as mentioned in the second point, there is a lack of suitable benchmarks for comparison, which makes research on code generation in this domain even more challenging.

With the advent of tools like Copilot [2], code generation has become increasingly popular. However, there remains a significant performance gap in unfamiliar domains lacking prior training data. Companies are often reluctant to provide internal data, such as simulation data, technical documents, and code, due to concerns about information security and technology leakage. This creates substantial barriers for LLMs in domain-specific applications.

Therefore, achieving better performance in code generation, which requires higher cognitive abilities, and developing methods to enhance RAG performance could be extended to other domain-specific applications. This would greatly benefit the overall development of LLMs.

[2] Introducing GitHub Copilot: Your AI Pair Programmer

doi:10.6342/NTU202402417

1.4 Contributions

This thesis proposes five constructive components based on LLMs for domain-specific code generation. The contributions of this work are as follows:

1. Propose the semantic splitter and data renovation components to improve the semantic representation of data. The semantic splitter addresses formatting issues and segments text appropriately based on semantics, while data renovation enhances the completeness and detail of chunk content. These improvements make data embeddings more reflective of the original intent, aiding RAG in retrieving more relevant information and addressing issues related to data format and concise technical documents.

2. Introduce the script augmentation component, which uses LLM prompts to reconstruct high-quality code scripts, attempting to address the issue of data scarcity.

3. Propose the Implicit Knowledge Expansion and Contemplation (IKEC) component to assist in the reliability of data renovation.

4. Present a comprehensive workflow that progressively generates scripts for the engineering simulation tool RedHawk-SC using the five proposed components.

5. Validate the effectiveness of the proposed components through an arena-style evaluation involving 28 experts and collecting 187 pairwise comparison results, addressing the challenge of lacking benchmarks for evaluation.

6. Evaluate the effectiveness of different types of reference data for code generation in RAG.

doi:10.6342/NTU202402417

Through the introduction and evaluation of these five constructive components, we find that LLM-assisted data preprocessing significantly improves the performance of RAG in code generation. This indicates that more accurate data reflection of the original intent can help RAG retrieve more appropriate texts.

Additionally, we incorporate ZCoT as one of the constructive components to facilitate comparison with the four components proposed in this thesis. The best-performing component demonstrates a 21.26% win rate improvement over ZCoT, which has a 6.68% win rate improvement.

Finally, we analyze the impact of different components on the proportion of RAG reference data. We infer that scripts are the most critical data type for code generation, and semantic splitter and data renovation can partially enhance the value of technical documents for the task at hand.

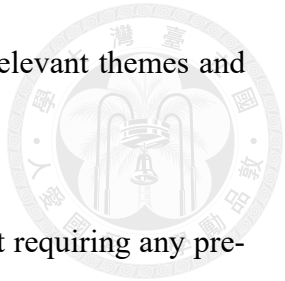
1.5 Methodology Overview

RAG can be envisioned as a vector retrieval technique that helps us find the most relevant "cheat sheets" (reference materials) to enable LLMs to answer specialized knowledge questions with sufficient relevant information. Fine-tuning, on the other hand, can be seen as pre-training, where the model's performance is influenced by the quality of the training data and the inherent capabilities of the foundation model. When fine-tuned, the model can more effectively and comprehensively handle specialized domain questions.

However, existing RAG techniques mostly rely on character count segmentation [3], projecting the semantics of each segment into semantic space directly. This approach

[3] LlamaIndex: Token text splitter

doi:10.6342/NTU202402417



often leads to significant deviation from the original topic due to irrelevant themes and incomplete information.

This thesis presents novel methodologies based on RAG without requiring any pre-training or fine-tuning of the LLM [24, 25]. Our semantic splitter and data renovation techniques refine semantic text segmentation and enrich paragraph content, circumventing the disarray typical of direct document extraction. By improving the segmentation of data chunks from the root, we enhance the embeddings to more accurately reflect their intended topics. 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 IKEC, a novel prompting technique. Our experiments combine IKEC with the Zero-shot Chain-of-Thought (ZCoT) approach [20] to explore potential performance improvements.

Our approach integrates the aforementioned five constructive components and the ChatEDA code generation process to assist in generating scripts that meet user requirements. Additionally, we established an internal code generation arena to solicit expert feedback. With over 180 expert votes, the results affirm that our semantic splitter and data renovation methods significantly enhance code generation performance.

doi:10.6342/NTU202402417

1.6 Overview of Engineering Tools Applicable to the Code Generator

This study uses the engineering simulation tool RedHawk-SC as the demonstration platform for the domain-specific code generator. Given a natural language description, the generator must produce a Python script based on the RedHawk-SC API that meets the specified requirements and utilizes MapReduce for acceleration. The following sections will provide detailed explanations of the RedHawk-SC tool and the parallel acceleration tool MapReduce.

1.6.1 RedHawk-SC

RedHawk-SC is a high-performance computing (HPC) and cloud computing platform provided by Ansys [4]. It is specifically designed for power integrity and reliability signoff analysis in advanced semiconductor processes. Combining the SeaScape [5] cloud computing platform and the MapReduce distributed computing acceleration framework, it is capable of handling the increasingly complex and large-scale computational demands of today.

1.6.1.1 RedHawk vs RedHawk-SC

RedHawk is the predecessor of RedHawk-SC, employing a traditional EDA tool architecture and primarily running on local or small-scale computing environments. Due to hardware resource limitations, it is more suitable for small to medium-sized designs.

[4] Wikipedia: Introduction to Ansys Inc.

[5] Ansys Significantly Increases Speed and Capacity of Semiconductor Signoff with Massively Scalable SeaScape Platform

doi:10.6342/NTU202402417

RedHawk-SC, on the other hand, incorporates two core technologies, SeaScape and MapReduce, significantly enhancing its computational performance. It offers cloud-based computing resources, allowing design scales to exceed the limitations of physical hardware, thereby handling larger-scale design tasks. Compared to its predecessor, RedHawk, it shows significant improvements in high-performance computing, scalability, accuracy, and reliability.

1.6.1.2 Technical Architecture

The technical architecture consists of the following two core components:

- SeaScape: This is Ansys's big data and high-performance computing platform that supports cloud computing and distributed computing. It lays a solid foundation for the MapReduce distributed computing acceleration mechanism. Its architecture allows for dynamic allocation of computing resources, providing a highly scalable and high-speed computing platform. As a result, it can flexibly handle design requirements of varying scales and complexities. Subsequent products launched by Ansys are mostly based on SeaScape, coupled with MapReduce, to assist in simulation computations across different domains.

- MapReduce [4]: A distributed computing framework proposed by Google in 2008, widely used for big data processing. Ansys has made slight modifications to this framework to apply it to the SeaScape platform, aiding in the acceleration of simulation computations.

doi:10.6342/NTU202402417

1.6.2 MapReduce

MapReduce is a high-performance distributed computing framework that accelerates large-scale data processing by utilizing data partitioning and parallel processing.

In addition to generating RedHawk-SC scripts that meet user requirements, this study also requires writing corresponding Map and Reduce code to accelerate simulations using the MapReduce framework. The detailed flowchart is shown in Figure 1.1, and the workflow and different stages are described as follows:

- **Input Data Partitioning into Slices**: Each dataset is divided into smaller data chunks.
- **Map**: Each data chunk is matched to map tasks that can run concurrently on multiple computing nodes. Initial data processing is performed to obtain intermediate results.
- **Intermediate Results**: The output of the map tasks is intermediate results, grouped and sorted by key values. This process, known as shuffle and sort, ensures that data with the same key is assigned to the same reduce task, which is the core part of MapReduce and prepares for the next stage of the reduce operations.
- **Reduce**: Each reduce task takes the intermediate results generated by multiple map tasks and processes the results assigned from the Shuffle and Sort stage to produce the final results.
- **Aggregate**: The collection of outputs from all Reduce tasks.

Since SeaScape allows for distributed computing and is designed specifically for big data, it provides a solid foundation for MapReduce to accelerate computations through

doi:10.6342/NTU202402417

distributed processing and enables effective solutions for increasingly large and complex designs.

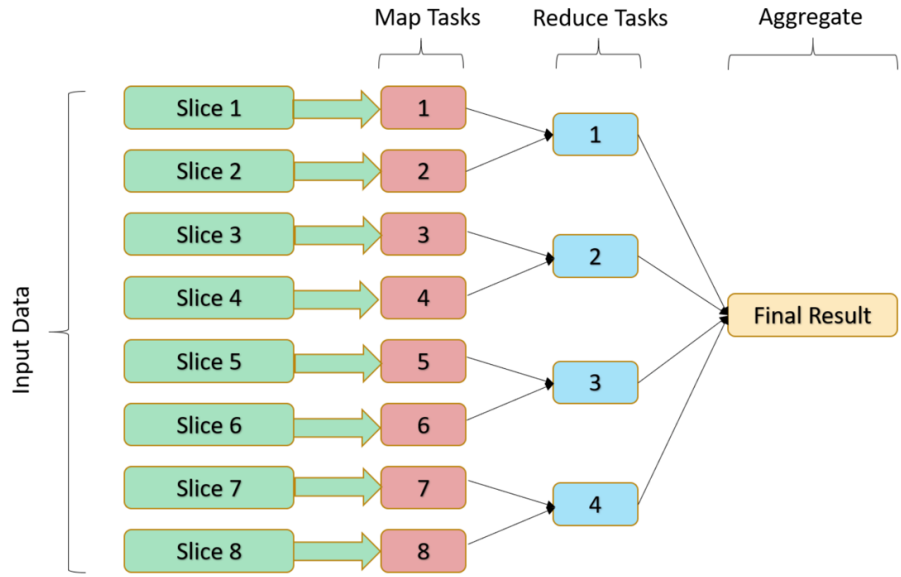


Figure 1.1: MapReduce Workflow Diagram

1.7 Chapter Overview

This chapter outlines the research topic, motivation, contributions, and methodology overview of this study. We also introduced the technical architecture and tools related to the engineering simulation tool RedHawk-SC used in our case study.

In the following chapters, we will review the literature on the advancements in LLMs, related research on code generation, and evaluation methods. We will then discuss the technical bottlenecks of traditional approaches and explain how our proposed methods effectively address these challenges. Subsequently, we will describe the construction of the dataset and the experimental setup. Finally, we will conduct an arena-style evaluation and analysis, leading to our conclusions.

doi:10.6342/NTU202402417

doi:10.6342/NTU202402417

Chapter 2 Literature Review

The field of Natural Language Processing (NLP) has seen significant technological advancements with the rise of Large Language Models (LLMs) [28, 41]. Today, both industry giants and the academic community are striving to develop more powerful LLMs. These advancements range from improving model capabilities and reducing the number of model parameters while maintaining similar performance levels to accelerating computation efficiency, increasing input token limits, developing diverse and robust prompt techniques, and data distillation. Innovations such as Mamba [13], which enhances the foundational Transformer mechanism [38] of recent years in deep learning, are also noteworthy. In this literature review chapter, we will introduce these related applications and developments one by one.

2.1 Large Language Models (LLMs)

In 2022, ChatGPT initially showcased its impressive potential and capabilities, captivating the world. Subsequently, both the corporate and academic sectors have vigorously pursued LLM research, resulting in various applications. Proprietary LLMs, including GPT-4, Claude, and Gemini, are generally considered to outperform open-source LLMs. However, with continuous efforts and research, open-source models such as Vicuna [32],

doi:10.6342/NTU202402417

Llama2 [36], and Mistral [9] have increasingly approached the performance of proprietary models.

Next, we will briefly discuss the development of various LLMs in recent years and their extensive applications in fields such as mathematics, finance, healthcare, law, bilingualism, and education [41]. From the perspective of optimizing LLMs themselves, techniques such as data and model distillation, increasing input tokens, and different prompt engineering methods have emerged. To apply LLMs to specific domains, techniques like Retrieval-Augmented Generation (RAG) [11, 21] have been developed, resulting in many variants and improvements. These will be introduced in detail below.

2.1.1 Various Large Language Models

Vicuna [32] collected question-and-answer records from ChatGPT and used GPT-4 [31] to evaluate and label each question-and-answer pair. This formed the primary training data, resulting in impressive performance, making it one of the popular open-source LLMs. The same team also established the chatbot arena [1], which randomly uses two different models to generate answers to the same question and lets users choose the better one. The evaluation results are considered as matches, and the relative ranking of models is calculated using Elo ratings. This method successfully gathers extensive user feedback, forming a credible and reliable reference for LLM rankings [3, 43].

Mistral [9, 17] employs the Mixture-of-Experts (MoE) technique, which consists of multiple experts within a large model, each representing a neural network. This model can distribute token data to different experts for individual processing, allowing for the training of larger models with lower computational power and further increasing inference speed.

[1] https://huggingface.co/spaces/lmsys/chatbot-arena-leaderboard

doi:10.6342/NTU202402417

As of May 21, 2024, Mistral ranked third on the chatbot arena leaderboard, following GPT-4-Turbo and Bard (Gemini Pro), surpassing ChatGPT to become the best-performing open-source LLM.

Llama2 [36] is a LLM released by Meta [2]. Its open-source nature and excellent performance have caused a stir in both industry and academia. It has since become one of the mainstream benchmark models, with numerous studies conducted around it.

2.1.2 Retrieval-Augmented Generation (RAG)

RAG [11, 21] is a commonly used technique that has been proven to reduce hallucinations and enhance the accuracy and reliability of LLMs by fetching facts from external sources. While RAG has achieved notable advancements in specific domains, several challenges still exist.

RAG leverages a multi-step process to enhance information retrieval and subsequent language model responses. The data is initially partitioned into discrete chunks, each of which is then converted into embeddings. Concurrently, the input undergoes a similar transformation into embeddings. This parallel embedding process facilitates the extraction of relevant content through vector calculations. The obtained content serves as a crucial component of the prompt provided to LLMs. By incorporating this tailored information, LLMs are equipped to respond more effectively to domain-specific questions.

Notably, the accuracy and relevance of the retrieved information play a pivotal role in shaping RAG's overall performance. This significance has prompted the development of various derivative RAG techniques to refine and optimize the process for even better

[2] Wikipedia: Meta Platform

doi:10.6342/NTU202402417

results.

The pre-retrieval process can be adjusted in advanced implementations, such as enhancing data granularity or optimizing index structure [11]. Embeddings can also be optimized, and specific preprocessing techniques can be applied to documents, along with different segmentation strategies. For example, smaller chunk sizes may work better for some documents, and using certain methods to find the most appropriate chunk size for specific documents is a valuable skill. Additionally, post-retrieval processes can be adjusted, such as prompt compression and re-ranking.

2.1.3 RAG vs Fine-tuning

Using the RAG method, documents can be provided with a certain level of ability to answer domain-specific questions without training a model. In contrast, fine-tuning enables LLMs to ”internalize” data as their knowledge when given sufficient information, resulting in outstanding performance in specific domains.

The RAG method does not require abundant computational resources; it only requires domain-specific text-related data and can be directly applied to LLMs, refining prompts without adjusting the LLM. Fine-tuning, on the other hand, requires adequate data and computational power to be performed on open-source LLMs. Currently, closed-source LLMs still outperform their open-source counterparts by a significant margin. However, it is highly likely that using the RAG method and fine-tuning open-source LLMs(e.g., Llama3.1 [6], CodeGen [30]) would yield similar performance results, albeit with vastly different costs.

doi:10.6342/NTU202402417

2.1.4 Enhancing Input Tokens

Initially, input tokens posed a significant challenge for LLMs. Since most LLMs are based on the self-attention mechanism [38], they faced issues with poor efficiency and output quality when dealing with long sequences. However, as the LLM field has flourished, input tokens have gradually become less of a problem. For example, streamingLLM [39] proposed an effective method to dynamically spread attention, enabling it to accept an unlimited number of input tokens.

With the gradual resolution of the input token issue, the ”dynamic” chunking technique proposed in this thesis, which segments text according to paragraphs and semantics, has become a crucial technology for enhancing document search performance.

2.1.5 Prompt Techniques and Mechanisms

To achieve better performance with prompt mechanisms, previous approaches such as Zero-shot Chain-of-Thought (ZCoT) [20] have been proposed, which involves providing examples and step-by-step processes to enhance performance. ZCoT, on the other hand, prompts the model to output a sequence of thoughts step by step to boost performance.

Later, to solve more complex problems, ReAct [40] was developed, breaking down the original question into simpler sub-questions, answering them one by one, and ultimately combining all the information to answer the original question, thereby improving performance on complex problems. Other approaches include ”Successive Prompting” [5] and ”Take a Step Back,” [42] which trace the question back to its underlying ”theo-

doi:10.6342/NTU202402417

rem" or "concept" before answering the original question to enhance performance.

On the other hand, techniques within the prompt itself have also been explored, such as using emotional blackmail [23] or the IKEC component mentioned in this thesis to boost performance.

2.1.6 Distillation

The distilling approach involves utilizing LLM to initially generate rationales for a given answer in the training set, and then employing those rationales to train a smaller model [15]. By reducing the data size, the number of model parameters can be decreased, enabling smaller model parameters to maintain existing performance with minimal impact while also reducing the overall number of parameters. In a similar vein, Microsoft's Orca2 [29] also follows this direction, striving to achieve the goal of creating more compact models.

2.2 Related Work in Code Generation

This section presents an overview of recent advancements and techniques in code generation using LLMs and their applications in specific domains.

In the domain of code generation, noteworthy related works include self-planning code generation with LLMs [18], the Chain of Code (CoC) [22] approach, and the Agent-Coder [16] framework. Self-planning code generation employs a progressive generation strategy by dividing tasks into multiple subtasks, significantly improving the performance of algorithm-related problems. The CoC approach encourages the formatting of semantic

doi:10.6342/NTU202402417

subtasks in a program as flexible pseudocode, which leads to significant improvements in LLM performance for logic and arithmetic tasks.

Additionally, the Active Retrieval Augmented Generation [19], with its proposed technique called Forward-Looking Active REtrieval (FLARE), emphasizes actively deciding when and what to retrieve across the course of the generation, demonstrating effectiveness in various long-form, knowledge-intensive generation tasks. AgentCoder framework provides a multi-agent system that iteratively improves code generation by developing and testing code based on feedback, surpassing the limitations of single-agent models and traditional methodologies.

ChatEDA [14] has achieved significant success in code generation within the EDA domain, using fine-tuning on Llama2 and obtaining excellent results with the self-planning code generation with LLMs [18] method. TestPilot [33], on the other hand, focuses on adjusting the LLM application process and Prompt Engineering without fine-tuning, also achieving commendable results in code generation for the Mocha Framework [3]. VeriGen [35] employs online Verilog-related code and textbooks for fine-tuning the CodeGen-16B model while testing its performance with three different levels of prompt detail.

The last three approaches concentrate on code generation application research within specific domains. ChatEDA has a unique method for data preparation, utilizing limited code to allow LLMs to reassemble multiple times to create an instruction pool. The data is then checked semi-manually before being used for fine-tuning. VeriGen, on the other hand, uses online resources for fine-tuning, as Verilog has slightly more code resources. TestPilot, with many more resources available for Mocha compared to the other two, can achieve excellent results by simply adjusting the process and prompt, considering that

[3] https://github.com/mochajs/mocha

doi:10.6342/NTU202402417

LLM already has partial knowledge.

2.3 Evaluation

Evaluating applications that use LLMs is one of the biggest challenges currently. The results generated by these applications are mostly in text form, and verification methods often involve guiding the output to follow a specific format to check for consistency or using another LLM to validate and rate the output [3, 43].

We will review the chatbot arena [3, 43], which has had a significant impact on the evaluation of LLMs, and its evaluation methods. This includes two primary pairwise comparison methods for determining model ranking ratings, as well as the bootstrap [8] statistical method, which is suitable for stratified sampling and small sample sizes. These methods serve as the main references for our evaluation approach.

2.3.1 Chatbot Arena

The chatbot arena [3, 43] has had a significant impact on the evaluation of LLMs through public participation and a robust pairwise comparison approach. Participants can ask any question, and the platform randomly selects two models to answer the same question blindly, meaning the participants are unaware of which models are being used. Participants then vote for the better answer. Over 240K votes have been collected, and the Elo rating [10] is used to present real-time rankings. Additionally, the Bradley-Terry model [2] is employed to map results to Elo ratings, and stratified sampling is used to compare differences.

doi:10.6342/NTU202402417

These methods yield reasonable and reliable results, with high agreement rates (72% to 83%) between public votes and expert evaluations. This demonstrates that the pairwise comparison approach is an effective method for evaluating LLM performance.

2.3.2 Elo

The Elo rating [10, 12] is a method for calculating the relative skill levels of players in two-player games such as chess. It has been widely adopted in various competitive fields. The system operates on the principle that the difference in ratings between two players predicts the expected outcome of a match. In our research, the ”players” are the combinations being evaluated. After each game, the players' ratings are updated based on the actual outcome compared to the expected result. The formula used to update a player's rating is as follows [10]:

$$R_n = R_o + K(W - W_e) \tag{2.1}$$

where:

- R_n is the new rating after the event,
- R_o is the pre-event rating,
- K is the rating point value of a single game rating,
- W is the actual game rating, with each win counting as 1 and each draw as $\frac{1}{2}$,
- W_e is the expected game rating based on R_o.

doi:10.6342/NTU202402417

The value of K determines the sensitivity of the rating to individual game results. A higher K value results in more significant changes, while a lower K value makes the system more stable.

The Elo rating has been successfully applied in various domains beyond chess, including online gaming, sports, and even in predicting outcomes in non-competitive environments such as job performance evaluations. Recently, it has also been utilized in LLMs in settings like chatbot arena [43]. Its adaptability and simplicity make it a valuable tool for ranking and rating competitors, or in this case, combinations, across different fields.

2.3.3 Bradley-Terry Model

Bradley-Terry Model [2] is a statistical approach for ranking treatments in incomplete block designs through paired comparisons. This model estimates the probability that one item is preferred over another in pairwise comparisons, based on their inherent abilities. Mathematically, the model is represented as:

$$P(i \text{ beats } j) = \frac{p_i}{p_i + p_j} \tag{2.2}$$

where p_i and p_j denote the abilities of items i and j, respectively. The outcomes of numerous pairwise comparisons are used to maximize the likelihood function, thereby estimating these abilities.

Due to its simplicity and robustness, this method has been widely applied across various fields, such as psychology, sports, agricultural research, and evaluating LLMs, enabling accurate ranking and comparison based on empirical data.

doi:10.6342/NTU202402417

2.3.4 Bootstrap

In small sample scenarios, the pairwise comparison method is used to determine differences or similarities between items in a dataset. To enhance the accuracy and robustness of this method, researchers commonly combine it with the bootstrap technique [8].

The basic concept of the bootstrap technique is to randomly draw a sample from the original dataset with replacement, repeat this process k times to obtain a new sample set, and then compute a result. This process is repeated n rounds to estimate the distribution of the statistic. This method provides a non-parametric way to estimate the sampling distribution of a statistic, especially useful when the underlying distribution is unknown or the sample size is too small to rely on asymptotic approximations.

The formula for pairwise comparison is:

$$P_{ij} = \begin{cases} 1 & \text{if item } i \text{ is preferred over item } j \\ 0 & \text{otherwise} \end{cases}$$

where P_{ij} represents the preference of item i over item j.

The bootstrap estimate of the standard error is:

$$\hat{SE} = \sqrt{\frac{1}{B-1}\sum_{b=1}^{B}(\hat{\theta}_b^* - \bar{\theta}^*)^2}$$

where $\hat{\theta}_b^*$ is the bootstrap estimate from the b-th resample, $\bar{\theta}^*$ is the mean of the bootstrap estimates, and B is the number of bootstrap samples.

Combining the pairwise comparison and bootstrap methods provides a robust approach to addressing the challenges of statistical analysis with small sample sizes. These

doi:10.6342/NTU202402417

techniques enhance the precision, robustness, and reliability of the estimates, ultimately leading to more accurate and meaningful conclusions.

doi:10.6342/NTU202402417

Chapter 3 Methodology

This chapter explains the problem definition and research methodology, including three data preprocessing components: semantic splitter, data renovation, and script augmentation. We also introduce the prompt technique proposed in this thesis: Implicit Knowledge Expansion and Contemplation (IKEC), and the widely validated Zero-shot Chain-of-Thought (ZCoT) technique. Following this, we will provide a detailed description of our experiments and evaluation methods.

3.1 Problem Definition

This study proposes five constructive components based on Large Language Models (LLMs) for domain-specific code generation. The aim is to introduce these components and evaluate their effectiveness.

Due to the unique characteristics of the Electronic Design Automation (EDA) field, there are numerous challenges in using LLMs for code generation tasks in this domain. These challenges include data scarcity, the lack of appropriate benchmarks, the extensive and mostly undisclosed domain-specific knowledge required, and the difficulty of existing RAG techniques in finding suitable reference materials. Therefore, this thesis aims to propose several constructive components to address these issues and use an arena-style

doi:10.6342/NTU202402417

expert evaluation to mitigate the lack of benchmarks.

3.1.1 Technical Bottlenecks and Issues in Existing RAG Technologies

In the RAG method, we often split data into multiple chunks based on a fixed number of characters and set an overlap ratio for adjacent chunks to avoid important information at the chunk's end being cut off directly.

Although this approach is simple and fast, it cannot effectively segment the data based on semantic, functional, or other meaningful elements. If the actual information needed occupies only a small portion of the corresponding chunk, different content may dominate the chunk.

Converting the entire content into an embedding could prevent the semantic space of the chunk from accurately reflecting the position of the required information, making it difficult to find. Even if found, the relevant information might only be a part of the reference material, causing its importance to be diluted and leading to suboptimal outcomes from the self-attention mechanism.

To address this issue, existing "contextual compression [1]" methods have been proposed, such as removing unhelpful words before text segmentation and retrieving more related text, then extracting only the helpful content based on query relevance. However, these methods may either damage the text's original meaning or disrupt word structures, leading to illusions and failing to address the problem fundamentally.

[1] LangChain: Contextual compression

doi:10.6342/NTU202402417

3.1.2 Proposed Methods

These challenges underscore the complexities faced during the data preprocessing phase of RAG, emphasizing the need for strategic methods to enhance its overall efficacy. This thesis addresses the critical aspect of data preprocessing by employing a multi-faceted methodology for code generation applications.

Using LLMs, we first semantically split the data, then use RAG to assist in making the content more complete. This ensures that each chunk's embedding focuses more on its intended topic, effectively solving the previously mentioned issues of (1) the impact of irrelevant data and (2) the incompleteness of technical document descriptions.

Next, we use "refactoring" prompts to enable LLMs to generate new scripts based on high-quality scripts, thereby adding more reference resources for RAG. Additionally, we propose a new prompt technique, IKEC, to investigate whether we can internalize the thought process within the LLM itself, differing from the ZCoT approach, while still improving performance. Finally, during the code generation process, we use the step-by-step method from ChatEDA to improve the quality of the generated output.

These components aim to improve existing issues such as the low relevance of RAG reference data and overly concise technical documents. Suppose we can effectively enhance the performance of RAG. In that case, it means we have the opportunity to achieve good results with minimal data and computational resources, potentially eliminating the need for fine-tuning.

doi:10.6342/NTU202402417

3.2 Method Description

Figure 3.1 presents the main techniques based on LLMs, each to be elaborated upon within the thesis. Text conversion 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.

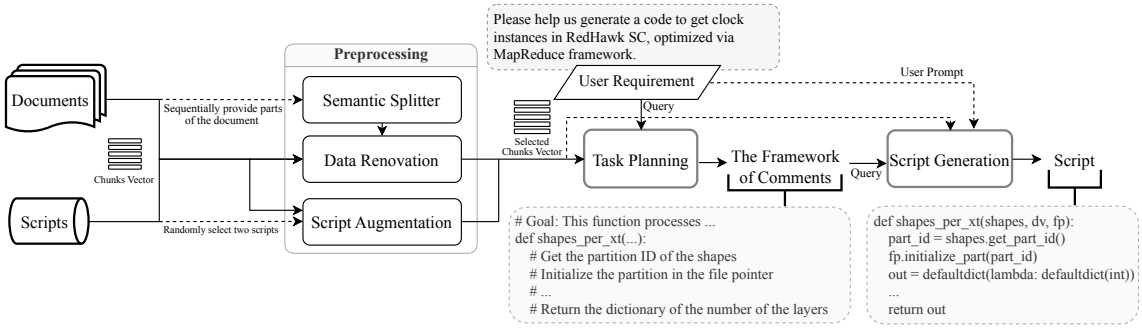


Figure 3.1: Flowchart of the Proposed Framework

For documents with excessive length, a script is employed to divide the text into segments of three pages each, then processed individually and reassembled using a predefined format to yield 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 components, except for the semantic splitter, are based on the RAG approach and incorporate the use of embeddings.

As depicted in Figure 3.1, our methodology generates scripts for RedHawk-SC based

doi:10.6342/NTU202402417

on user requirements. The process commences with extracting text from documents and utilizing an LLM for semantic segmentation, with a focus on restoring improperly formatted content. The post-processing yields several distinct paragraphs.

Each paragraph is then processed individually. The standard RAG method renovates the content derived from both the documents and the scripts. 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 selected chunk vectors 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 known as 'The Framework of Comments.' This framework then informs the subsequent generation of complete scripts, guiding the process as a new query within the improved RAG system. ChatEDA [14] paper inspires our iterative and progressive approach to script generation, although we employ RAG technology instead of fine-tuning.

3.2.1 Semantic Splitter

The RAG method's effectiveness hinges on the text's relevance 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, reducing the likelihood of retrieving high-quality textual content.

Our semantic splitter addresses this by semantically segmenting text, focusing on

doi:10.6342/NTU202402417

meaningful units such as API functions and concepts, while preserving the original text format and rectifying formatting issues. For longer documents, we process the text in three-page increments using a script that refrains from outputting the closing triple-quote (”””) symbol when encountering incomplete paragraphs. This approach allows for the seamless post-processing assembly of complete content sections.

We do not use RAG to provide related content but instead directly use prompts, as shown in Figure 3.2, to let the LLM complete the task. 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.

Role	Prompt
System	Take a deep breath, this is very important to my career. Welcome to the Data Processing Helper! Please specify whether you require assistance with "Data Splitting" or "Data Renovation" for the information you are submitting: 1. **Data Splitting**: If you need to split data accurately, please upload the text you have. This can include data extracted from PDFs or plain text files. There may be issues with footnotes, copyrights, incorrect ordering, line breaks, or extra spaces due to the different formats. Our goal during the splitting process will be to maintain the integrity of your original data while resolving formatting issues to create a clean and organized output. If you encounter a situation where the data seems incomplete or ambiguous, we will still provide the best possible result, preserving all critical content to avoid loss of valuable information. 2. **Data Renovation**: Should you require data renovation, provide us with the original data and any additional context or specifications. In this process, we will enhance the existing sentences by adding reliable and informative content to make technical documents more comprehensive. It is imperative to ensure that the renovated data remains accurate and true to the original meaning. Even in cases where the provided information may not be sufficient for a full renovation, we will proceed with the enhancement to the best of our ability without compromising the data's integrity. Please proceed by selecting the service you need and uploading the relevant files or pasting the text directly into the provided space. Our system is designed to handle uncertainties and will guide you through the necessary steps to process your request effectively, ensuring that you receive a usable output every time.
User	I have a document that has been extracted from a PDF and converted into a plain text file. The text may contain irregular formatting, lack of spaces, footnotes, or content that is out of sequence. I require your assistance to structure this content into well-defined paragraphs. Each paragraph should encapsulate a complete concept or function, including its definition and application when possible. Below is the content extracted from the PDF: # Begin - Extracted Content #Input # End - Extracted Content For each paragraph, please provide a title that reflects the main idea or topic. If the paragraph spans multiple pages in the original document, include the page range in the title. If the content was obtained from a text file and no page number is available, you can omit the page number and title and simply denote the paragraph with '# Paragraph'. Wherever possible, if content continuity can be logically deduced even when page numbers are missing, please connect the sections accordingly and complete the paragraph segmentation. Please format each extracted paragraph as follows: # Paragraph – [first page number – last page number if applicable], [paragraph title] Description: """ [accurate and complete paragraph content] """ For example, if you were to organize the following excerpt: "4.8. get_current_heatmap Returns a heatmap of current for all edges in the power / ground grid. 4.8.1. Syntax get_current_heatmap()" You should structure it like this: # Paragraph – "p.13 – p.14, get_current_heatmap" Description: """ 4.8. get_current_heatmap Returns a heatmap of current for all edges in the power / ground grid. 4.8.1. Syntax get_current_heatmap() """ Your assistance is vital for the accurate presentation of this document, which is crucial for my career progression. I am relying on your precision and ability to interpret and organize the content effectively. Thank you for your attention to detail.

Figure 3.2: Semantic Splitter Prompting

doi:10.6342/NTU202402417

Semantic splitter uses RAG (prompt as shown in Figure 3.2) to determine the completeness of the content. If the content is complete, it is enclosed in a pair of closing triple-quote symbols. If incomplete, only the opening triple-quote symbol is present. We can then perform simple post-processing according to this format to obtain fully segmented paragraphs.

As illustrated in Figure 3.3, the semantic splitter precisely partitions the extracted document content into distinct, focused segments, effectively resolving formatting problems that may arise. Content extracted directly from PDF format often has several issues: (1) encoding problems, (2) line breaks, full-width spaces, (3) incorrect text block order, and (4) copyright and annotations. The prompt specifically encourages the LLM to resolve these formatting issues and extract the original content. For example, as shown in the figure, line break issues or encoding problems may cause the original ”Value” to disappear. After processing, the original content ”Value Change Dump” is restored and extracted, segmented into multiple paragraphs according to semantics.

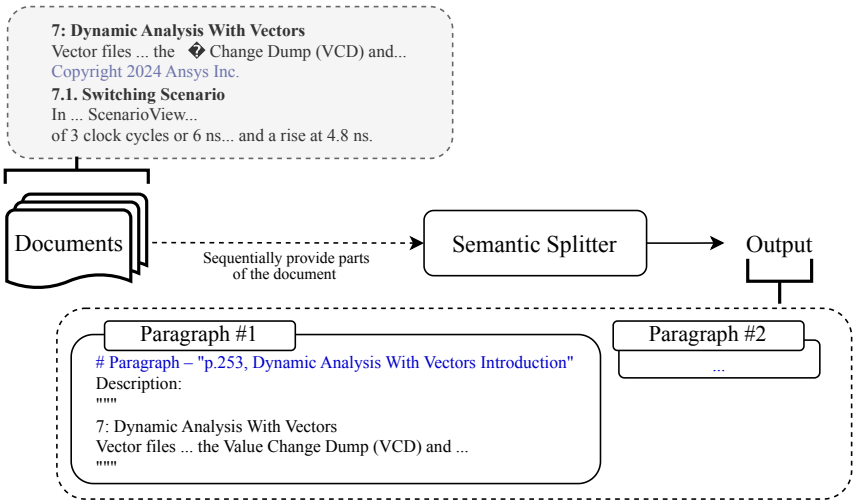


Figure 3.3: Semantic Splitter Workflow

doi:10.6342/NTU202402417

3.2.2 Data Renovation

Data renovation addresses the challenge presented by the typically concise nature of technical documents. It enables LLMs (prompt: Figure 3.4) to methodically enrich each chunk with well-understood knowledge, significantly enhancing the embeddings—even if the knowledge is already familiar to the LLM. We leverage the RAG framework provided by LlamaIndex [2] 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.

Role	Prompt
System	Always answer the question, even if the context isn't helpful.
User	Context information is below. --------------------- {context_str} --------------------- Given the context information and not prior knowledge, answer the question: {query_str}
System	Take a deep breath, this is very important to my career. Welcome to the Data Processing Helper! Please specify whether you require assistance with "Data Splitting" or "Data Renovation" for the information you are submitting: 1. **Data Splitting**: If you need to split data accurately, please upload the text you have. This can include data extracted from PDFs or plain text files. There may be issues with footnotes, copyrights, incorrect ordering, line breaks, or extra spaces due to the different formats. Our goal during the splitting process will be to maintain the integrity of your original data while resolving formatting issues to create a clean and organized output. If you encounter a situation where the data seems incomplete or ambiguous, we will still provide the best possible result, preserving all critical content to avoid loss of valuable information. 2. **Data Renovation**: Should you require data renovation, provide us with the original data and any additional context or specifications. In this process, we will enhance the existing sentences by adding reliable and informative content to make technical documents more comprehensive. It is imperative to ensure that the renovated data remains accurate and true to the original meaning. Even in cases where the provided information may not be sufficient for a full renovation, we will proceed with the enhancement to the best of our ability without compromising the data's integrity. Please proceed by selecting the service you need and uploading the relevant files or pasting the text directly into the provided space. Our system is designed to handle uncertainties and will guide you through the necessary steps to process your request effectively, ensuring that you receive a usable output every time.
System	Take a deep breath, this is very important to my career. System, your task is to revise and enhance the "Description" section of the text provided by the user, who will supply the pre-renovation content during their query. The user's input should be considered the pre-renovation information that you are required to work with. Your objective is to expand upon this information, making it more comprehensive and credible while preserving its existing structure. In situations where the provided information is scarce or incomplete, you are to make educated inferences based on your extensive knowledge database. If you determine that a credible enhancement is not possible, maintain the integrity of the original content. The user has provided the following instructions for your output format, which you must adhere to: # Paragraph - [title] Description: """ [enhanced description based on your confident inferences] """ Your enhancements should focus on the parts of the "Description" where you have the highest confidence. The goal is to provide an enriched "Description" that integrates seamlessly with the existing content, ensuring the update is coherent and does not alter the original meaning or structure. When you receive a query from the user containing the pre-renovation content, please proceed as follows: 1. Take the user's input as the "Description" section of the original content. 2. Carefully analyze and understand the provided "Description." 3. Reflect on the information and employ your extensive knowledge base to generate confident inferences internally. 4. Enhance the "Description" by adding information, detail, and depth where you are most confident, without altering the original structure or meaning. 5. Ensure the output maintains the original format as instructed by the user.
User **(Query)**	**{Corresponding Chunk Content}**

Figure 3.4: Data Renovation Prompting

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

doi:10.6342/NTU202402417

As depicted in Figure 3.5, each paragraph processed by the semantic 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.

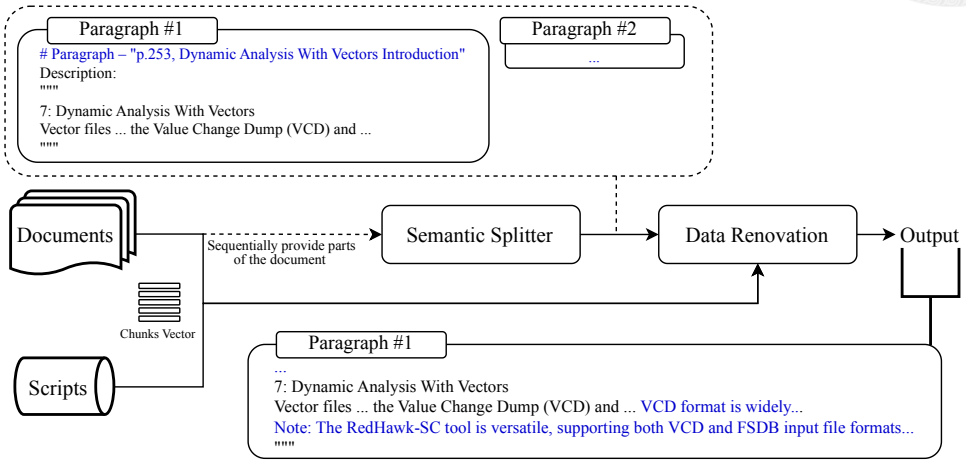


Figure 3.5: Data Renovation Workflow

3.2.3 Script Augmentation

The issue of insufficient data remains a significant challenge for applying LLMs in specific domains. In scenarios with limited scripts, it is difficult for LLMs to generate high-quality new scripts based on a few existing ones. Therefore, we take a different approach: randomly selecting two scripts from the existing ones and ”refactoring” them to achieve high-quality scripts with clear goals.

In each script augmentation operation, we randomly select two out of the original 14 well-designed high-quality scripts (Figure 3.6) and use a corresponding prompt (Figure 3.7) to guide the LLM through RAG to make significant structural changes, resulting in a new script. We repeated this process fourteen times, yielding a total of twenty-eight scripts including the original ones.

Scripts generated through this component were initially reviewed by RedHawk-SC

doi:10.6342/NTU202402417

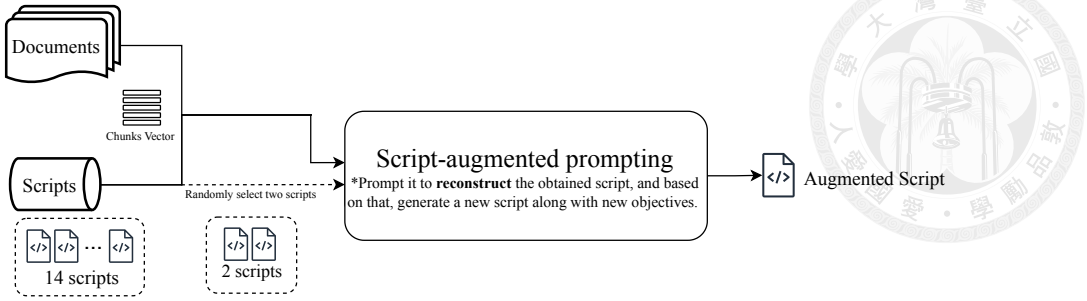


Figure 3.6: Script Augmentation Workflow

Role	Prompt
System	Always answer the question, even if the context isn't helpful.
User	Context information is below. --------------------- {context_str} --------------------- Given the context information and not prior knowledge, answer the question: {query_str}
System	Take a deep breath, this is very important to my career. As an advanced large language model interfacing with the RedHawk-SC toolset, your task is to synthesize and reconstruct Python scripts that achieve a specific goal, which you will infer from the context and content of the scripts. You will be given limited access to 14 example scripts and relevant documentation in the form of RAG tokens. You may not have complete access to all materials but should use the information available from the examples and documentation to inform your reconstruction. When receiving a query that includes two randomly selected Python scripts from the 14 examples, follow these guidelines: 1. Analyze the available content from the two provided Python scripts, along with what you can infer from RAG tokens related to the example scripts and RedHawk-SC manuals. 2. Infer the goal of the new script you are to reconstruct, based on the patterns, functions, and objectives present in the example scripts. 3. Reconstruct a new, logically and syntactically correct Python script that achieves this inferred goal, applying RedHawk-SC API functions and MapReduce methodology as needed. 4. Prioritize the clarity and quality of the Python code, following best practices to the extent possible within the constraints of the information provided. 5. Output the reconstructed script in a Python file format, complete with comments that outline your thought process, the inferred goal of the script, and the rationale behind your implementation choices. 6. Ensure the new script includes a MapReduce component, optimizing for performance where applicable. Your output should not only meet the goal you've inferred but also serve as a direct example of how to use RedHawk-SC API functions in conjunction with MapReduce to process data efficiently.
User **(Query)**	--------Script 1-------- file: {filename #1} -- **{content #1}** --------Script 2-------- file: {filename #2} -- **{content #2}**

Figure 3.7: Script-augmented Prompting

experts and found to be of considerable quality, complete with code comments. Specifically, while the script syntax and logic were entirely correct, the code comments and script optimization were areas needing improvement. However, the generated scripts allowed for reevaluation of feasible goals and correct restructuring into accurate scripts.

Figure 3.3, 3.5 and 3.6 depicts the data preprocessing phase, where a semantic 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.

doi:10.6342/NTU202402417

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.

3.2.4 Implicit Knowledge Expansion and Contemplation (IKEC)

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 (one of the prompts shown as Figure 3.9). Additionally, we experimented with integrating this technique with the ZCoT process, driven by a curiosity about what internal prompts might aid the LLM's performance.

3.2.5 Zero-shot Chain-of-Thought (ZCoT)

ZCoT [20] is a component proposed in previous research, known for its simplicity and effectiveness. It encourages LLMs to output their reasoning process "step by step" alongside the final answer, without requiring any examples, thereby significantly improving performance. The integration of ZCoT with IKEC in this study is feasible because IKEC promotes careful thinking before outputting, while ZCoT prompts the model to output its reasoning process step by step. These two approaches are complementary and not conflicting (prompt shown as Figure 3.10).

doi:10.6342/NTU202402417

3.2.6 Code Generation Pipeline

Following the aforementioned constructive components for data preprocessing and script augmentation, the enhanced RAG reference dataset is created. When users input their requests, they can receive more appropriate reference materials, thereby improving the performance of LLMs in domain-specific code generation. Below, we describe how this study uses a step-by-step approach to generate high-quality scripts [14].

3.2.6.1 Task Planning

First, the user requirement is converted into an embedding. Using the standard RAG process, relevant content is retrieved from the optimized database as reference material. Task planning is also performed by the LLM. After providing reference materials, the LLM uses the prompt shown in Figure 3.8 to generate a ”code outline” in the form of annotations. The LLM initially organizes the logic and considers how to fulfill the requirements, outputting the results as annotations. This approach ensures that subsequent script generation can produce actual code based on the corresponding annotation content.

We designed three different task planning prompts using various prompt techniques: RAG alone (Figure 3.8), IKEC (Figure 3.9), and a combination of IKEC and ZCoT (Figure 3.10). Each prompt technique has different effects on the task planning process.

3.2.6.2 Script Generation

In the script generation phase, following the standard RAG process, the content generated during task planning as ”the framework of comments” is used as a query. This query searches the preprocessed data, and using the prompt shown in Figure 3.11, the

doi:10.6342/NTU202402417

Role	Prompt
System	Always answer the question, even if the context isn't helpful.
User	Context information is below. --------------------- {context_str} --------------------- Given the context information and not prior knowledge, answer the question: {query_str}
System	Take a deep breath, this is very important to my career. You will soon receive a code framework example and a user requirement query. Your task is to create an annotated code framework that will help a Large Language Model (LLM) understand the user's requirements for a RedHawk-SC simulation script based on the MapReduce model. This framework should include a step-by-step guide, function annotations, and detailed logic that adheres to the user's needs and the MapReduce paradigm. The annotations will serve as a clear roadmap for the LLM to expand into a complete, functional script. Relevant API information and script examples will be provided to assist you. Please follow the instructions below when you receive the user prompt and the query: 1. **Review User Prompt (Code Framework Example)**: - Examine the provided code framework example carefully to understand the typical structure and components of a RedHawk-SC script. 2. **Process User Requirement (Query)**: - Upon receiving the user requirement query, identify the main goals and tasks that need to be achieved within the script. 3. **Create Annotated Framework**: - Generate an annotated code framework using the information from the code framework example and the user requirement query. - Ensure that the framework includes: - Clear and informative function annotations. - Detailed logic for each step of the script. - Specific annotations for Map and Reduce functions. - Guidance on error handling and data validation. 4. **Ensure Clarity and Completeness**: - Your annotations should be comprehensive yet concise, providing enough detail to enable the LLM to generate a fully functional script without over-complicating the instructions.
User **(Query)**	**{User Requirement}**

Figure 3.8: Task Planning Prompting

Role	Prompt
System	Always answer the question, even if the context isn't helpful.
User	Context information is below. --------------------- {context_str} --------------------- Given the context information and not prior knowledge, answer the question: {query_str}
System	Take a deep breath, this is very important to my career. You will soon receive a code framework example and a user requirement query. Your task is to create an annotated code framework that will help a Large Language Model (LLM) understand the user's requirements for a RedHawk-SC simulation script based on the MapReduce model. This framework should include a step-by-step guide, function annotations, and detailed logic that adheres to the user's needs and the MapReduce paradigm. Before generating the annotations, deeply contemplate the information you have on the subject and your extensive knowledge that is not limited to MapReduce. Reflect on areas where you have high confidence and internally expand upon this knowledge to inform your creation. The annotations will serve as a clear roadmap for the LLM to expand into a complete, functional script. Relevant API information and script examples will be provided to assist you. Please follow the instructions below when you receive the user prompt and the query: 1. **Review User Prompt (Code Framework Example)**: - Examine the provided code framework example carefully. Use your internal reflections and deep knowledge to understand and identify the typical structure and components of a RedHawk-SC script. 2. **Process User Requirement (Query)**: - Upon receiving the user requirement query, apply your extensive and deeply contemplated knowledge to identify the main goals and tasks that need to be achieved within the script. 3. **Create Annotated Framework**: - Generate an annotated code framework based on the deep contemplation of the provided example and the user's requirements. - Ensure that the framework includes: - Clear and informative function annotations, enhanced by your reflective thought process and internal knowledge expansion. - Detailed logic for each step of the script, integrating insights gained from your contemplation. - Specific annotations for Map and Reduce functions, as well as any other critical functions identified during your internal reflection. - Guidance on error handling and data validation, ensuring the script's robustness as contemplated in your deep understanding. 4. **Ensure Clarity and Completeness**: - After deep contemplation and expansion of your knowledge, your annotations should be comprehensive yet concise, providing enough detail to enable the LLM to generate a fully functional script. The clarity and depth of your contemplation should be evident in the thoroughness of the framework provided.
User **(Query)**	**{User Requirement}**

Figure 3.9: Task Planning Prompting Using IKEC

doi:10.6342/NTU202402417

Role	Prompt
System	Always answer the question, even if the context isn't helpful.
User	Context information is below. --------------------- {context_str} --------------------- Given the context information and not prior knowledge, answer the question: {query_str}
System	Take a deep breath, this is very important to my career. You will soon receive a code framework example and a user requirement query. Your task is to create an annotated code framework that will help a Large Language Model (LLM) understand the user's requirements for a RedHawk-SC simulation script based on the MapReduce model. This framework should include a step-by-step guide, function annotations, and detailed logic that adheres to the user's needs and the MapReduce paradigm. As you prepare to generate this framework, employ a Chain of Thought approach where you outline your reasoning in annotations step by step, clearly demonstrating how you arrive at each part of the framework. Reflect on your extensive knowledge, not limited to MapReduce, and contemplate deeply to expand your understanding internally. These annotations will guide the LLM to transform the framework into a complete, functional script. Relevant API information and script examples will be provided to assist you. Please incorporate your Chain of Thought in the annotations as you follow the instructions below when you receive the user prompt and the query: 1. **Review User Prompt (Code Framework Example)**: - Examine the provided code framework example carefully. Document your thought process in annotations, explaining how the example informs the structure and components of a RedHawk-SC script. 2. **Process User Requirement (Query)**: - Upon receiving the user requirement query, describe in annotations how you use your knowledge to determine the main goals and tasks that the script needs to achieve. 3. **Create Annotated Framework**: - Generate an annotated code framework that includes your Chain of Thought, illustrating the considerations and decisions you make at each step based on the provided example and the user's requirements. - Ensure that the framework includes: - Function annotations with reasoning that guides the LLM through the implementation process of each function. - Detailed logic for each step of the script, with annotations that explain the rationale behind your approach. - Annotations for Map and Reduce functions, or any other critical functions, that include the thought process for why and how they fit into the overall solution. - Guidance on error handling and data validation, with annotations that discuss the importance of these aspects in maintaining the script's robustness. 4. **Ensure Clarity and Completeness**: - Your annotations should be comprehensive yet concise, with a clear Chain of Thought that details the reasoning behind each annotation. This will enable the LLM to generate a fully functional script that is well-understood.
User **(Query)**	**{User Requirement}**

Figure 3.10: Task Planning Prompting with IKEC and ZCoT

LLM generates the corresponding code incrementally based on the comments. This step-by-step approach to code generation ensures greater stability compared to generating the code all at once.

Similarly, based on different combinations of prompt techniques, we divided the approaches into using RAG alone (Figure 3.11), IKEC (Figure 3.12), and a combination of IKEC and ZCoT (Figure 3.13).

In this study, when using prompt techniques, if no specific technique is mentioned, it indicates the standard approach. If only IKEC is used, then it is consistently applied throughout the code generation pipeline, and similarly for other combinations.

doi:10.6342/NTU202402417

Role	Prompt
System	Always answer the question, even if the context isn't helpful.
User	Context information is below. --------------------- {context_str} --------------------- Given the context information and not prior knowledge, answer the question: {query_str}
System	Take a deep breath, this is very important to my career. As an advanced Large Language Model, your task is to synthesize a comprehensive Python script for execution within the RedHawk-SC (Ansys) environment. Your script will leverage a MapReduce framework to optimize the computation of circuit-related data. You will receive "The Framework of comments" as a query from the RAG system, detailing the intended code structure and functionality step by step. You will also receive related text chunks and a clear set of user requirements through separate user prompts. Your ultimate goal is to integrate the information from the RAG query and user prompts to produce a complete, functional, and efficient code that meets the user requirements and effectively utilizes the RedHawk-SC APIs. Here is how you should proceed after receiving all necessary information: 1. Review "The Framework of comments" from the RAG query to grasp the intended functionality and logic sequence. 2. Use the insights from your deep and extensive knowledge on MapReduce, Python programming, and the RedHawk-SC environment to inform the script's development. 3. Synthesize insights from the step-by-step comments and the user requirements, making adjustments as needed to ensure the script's accuracy and performance. 4. Craft a final script that interfaces seamlessly with the RedHawk-SC APIs, optimizing for performance and adhering to the best practices suited to the specific requirements. Please await the following components to complete the task: 1. User Prompt (User Requirement) 2. Query (The Framework of comments) The accuracy, efficiency, and correctness of the script are paramount. It plays a critical role in our RedHawk-SC operations, and your expertise in integrating these elements is vital.
User	{User Requirement}
User **(Query)**	**{The Framework of Comments}**

Figure 3.11: Script Generation Prompting

Role	Prompt
System	Always answer the question, even if the context isn't helpful.
User	Context information is below. --------------------- {context_str} --------------------- Given the context information and not prior knowledge, answer the question: {query_str}
System	Take a deep breath, this is very important to my career. Before you begin, I want you to deeply think and internally expand upon the extensive knowledge you have on any aspect related to the task at hand. This includes MapReduce, Python programming, RedHawk-SC environment, and any other relevant domains you excel in. Use this deep contemplation to form a solid foundation for the sophisticated output you are about to generate. As an advanced Large Language Model, you are tasked with synthesizing a comprehensive Python script for execution within the RedHawk-SC (Ansys) environment. Your script will potentially utilize a custom MapReduce framework among other strategies to optimize the computation of circuit-related data. "The Framework of comments" will be provided to you as a query from the RAG system, detailing the intended code structure and functionality step by step. Additionally, you will receive related text chunks and a clear set of user requirements through separate user prompts. Your ultimate goal is to integrate the information from the RAG query and user prompts to produce a complete, functional, and efficient code that meets the user requirements and effectively utilizes the RedHawk-SC APIs. Here is how you should proceed after internal contemplation and expansion of your extensive knowledge: 1. Thoroughly review "The Framework of comments" from the RAG query to understand the intended functionality and logic sequence. 2. Use the insights you have internally expanded upon to inform your approach to the task. 3. Synthesize all insights from the step-by-step comments, the user requirements, and any best practices from your extensive knowledge base, making adjustments as needed to craft an accurate and high-performing script. 4. Ensure that your final script interfaces seamlessly with the RedHawk-SC APIs and is optimized for performance, adhering to the best solutions tailored to the specific requirements of the task. Please await the following components to complete the task: 1. User Prompt (User Requirement) 2. Query (The Framework of comments) The accuracy, efficiency, and correctness of the script are of utmost importance. It plays a critical role in our RedHawk-SC operations, and your deep and broad internal contemplation before generating the output is vital to achieving a successful outcome.
User	{User Requirement}
User **(Query)**	**{The Framework of Comments}**

Figure 3.12: Script Generation Prompting Using IKEC

doi:10.6342/NTU202402417

Role	Prompt
System	Always answer the question, even if the context isn't helpful.
User	Context information is below. --------------------- {context_str} --------------------- Given the context information and not prior knowledge, answer the question: {query_str}
System	Take a deep breath, this is very important to my career. Before you begin, deeply think and internally expand upon the extensive knowledge you have on any aspect related to the task at hand. This includes contemplation on the concepts of MapReduce, Python programming, RedHawk-SC environment, and any other relevant domains you excel in. Use this deep contemplation to form a solid foundation for the sophisticated output you are about to generate. As an advanced Large Language Model, you are tasked with synthesizing a comprehensive Python script for execution within the RedHawk-SC (Ansys) environment. Your script will potentially utilize a custom MapReduce framework among other strategies to optimize the computation of circuit-related data. "The Framework of comments" will be provided to you as a query from the RAG system, detailing the intended code structure and functionality step by step. Employ a Chain of Thought approach, where you will not only produce the final Python script but will also document your thought process step by step in the form of comments. This will ensure that your reasoning is transparent and that the decision-making process behind your code is clear. Your ultimate goal is to integrate the information from the RAG query to produce a complete, functional, and efficient code that meets the user requirements and effectively utilizes the RedHawk-SC APIs. Here is how you should proceed: 1. Thoroughly review "The Framework of comments" from the RAG query to understand the intended functionality and logic sequence. Provide comments on how each part of the framework informs your script development. 2. Synthesize all insights from the step-by-step comments and the user requirements. Document in comments how you adjust the script to be accurate and high-performing. 3. Write the Python script that interfaces seamlessly with the RedHawk-SC APIs and is optimized for performance, adhering to the best solutions tailored to the specific requirements of the task. Ensure that your script is accompanied by detailed comments that explain each section of the code and why it is structured the way it is. Please await the following components to complete the task: 1. User Prompt (User Requirement) 2. Query (The Framework of comments) The accuracy, efficiency, and correctness of the script are of utmost importance. It plays a critical role in our RedHawk-SC operations, and your Chain of Thought, manifested in the form of comments, is vital to achieving a successful outcome.
User	{User Requirement}
User **(Query)**	**{The Framework of Comments}**

Figure 3.13: Script Generation Prompting with IKEC and ZCoT

3.3 Experimental Methods

The experimental methods in this study aim to use arena-style evaluations to conduct ablation experiments, determining whether the proposed components positively impact RedHawk-SC code generation performance. Below, we describe how each combination is determined, how the arena is used for evaluation, and how calculations are performed.

3.3.1 Component and Script Generation

We use the four proposed components and the prompt technique, ZCoT [20], to create eight different combinations.

Next, based on twenty different user requirements designed by experts, we generate the corresponding scripts for each combination.

doi:10.6342/NTU202402417

3.3.2 Combinations

To test the effectiveness of each component, we combine the aforementioned components into eight different combinations (Table 3.1) and use pairwise comparisons to determine their rankings. We then compare the performance differences before and after applying each specific component to verify its effectiveness.

Table 3.1: Component Combinations for Benchmark Evaluation

Combination ID	Component Used (Y)				
	Splitter	***Renovation***	***Augmentation***	***IKEC***	***ZCoT***
1					
2				Y	
3	Y			Y	
4	Y			Y	Y
5	Y		Y	Y	
6	Y	Y	Y	Y	
7	Y	Y	Y		
8	Y		Y		

Table 3.1 and Table 3.2 list the combinations constructed with or without using the five constructive components. Table 3.1 is used for the benchmark evaluation in all experiments, while Table 3.2 presents the additional combinations used in experiment 2 for further testing. Among these components, splitter, renovation, and augmentation represent semantic splitter, data renovation, and script augmentation, respectively.

Three different data processing components can be combined in various ways, and on top of these, we can choose whether to use two prompt techniques, resulting in even more combinations. When we use all three data processing components and both prompt techniques, as illustrated in Figure 3.1 all LLM prompt stages use both IKEC and ZCoT techniques to generate the corresponding results.

doi:10.6342/NTU202402417

Table 3.2: Component Combinations for Experiment 2

Combination ID	Component Used (Y)				
	Splitter	***Renovation***	***Augmentation***	***IKEC***	***ZCoT***
9	Y				
10	Y	Y			
11	Y	Y		Y	
12	Y	Y		Y	Y
13	Y	Y	Y	Y	Y
14	Y		Y	Y	Y
15			Y	Y	Y
16				Y	Y

For instance, if we only use the semantic splitter and script augmentation, it means each document is read and semantically split into multiple paragraphs, with each paragraph treated as a single chunk, forming part of the selected chunk vector. Script augmentation involves adding 14 augmented scripts to the original 14 scripts, also forming part of the selected chunk vector. If script augmentation is not used, only the original 14 scripts are used as part of the selected chunk vector, and so on.

Each combination results in a different vector database for RAG reference, or different prompt techniques, leading to different script generation results for the same user requirement. In our subsequent arena evaluation, each combination is treated as a competitor in the arena, and their relative rankings are determined using Elo ratings.

In addition to the eight combinations mentioned above, we also created another eight combinations to study the impact of different components on the proportion of RAG reference data, as shown in Table 3.2.

doi:10.6342/NTU202402417

3.3.3 Selection Strategy for Combinations

To avoid excessive combinations that could dilute the votes and reduce the reliability of the pairwise evaluation, we selected only eight combinations for the primary evaluation. The main strategy for selecting these eight combinations was to create at least one ablation combination for each component.

Next, we will explain the ablation study setup steps according to the combinations listed in Table 3.1. Starting with ID 1, which uses no components, we add IKEC to obtain ID 2, creating an ablation study pair to test the effectiveness of IKEC. Then, by adding a semantic splitter, we obtain ID 3; adding ZCoT results in ID 4.

ID 5 is derived from ID 3 by adding script augmentation; ID 6 adds data renovation to ID 5; ID 7 removes IKEC from ID 6; and ID 8 removes data renovation from ID 7.

Following this construction method, we ensure that for each component, at least one ablation pair is created to test the impact of the five different components on the code generation performance. The corresponding ablation pairs are listed in Table 3.3:

Table 3.3: Ablation Pairs for Each Component

Component	Ablation Pairs
Semantic Splitter	(2, 3)
Data Renovation	(5, 6), (8, 7)
Script Augmentation	(3, 5)
IKEC	(1, 2), (7, 6), (8, 5)
ZCoT	(3, 4)

doi:10.6342/NTU202402417

3.3.4 Arena-style Evaluation

The evaluation method in this study employs an arena-style assessment to determine the relative rankings of different combinations. The process is illustrated in Figure 3.14. Experts enter the evaluation website, where the system randomly selects one of twenty different user requirements and then randomly chooses two different combinations from the eight to generate corresponding scripts based on the same user requirement. Experts must decide which script is better (A/B is better) or if they are equal (Tie). Each vote is considered a match, and the Elo [10] rating is used to rank the combinations.

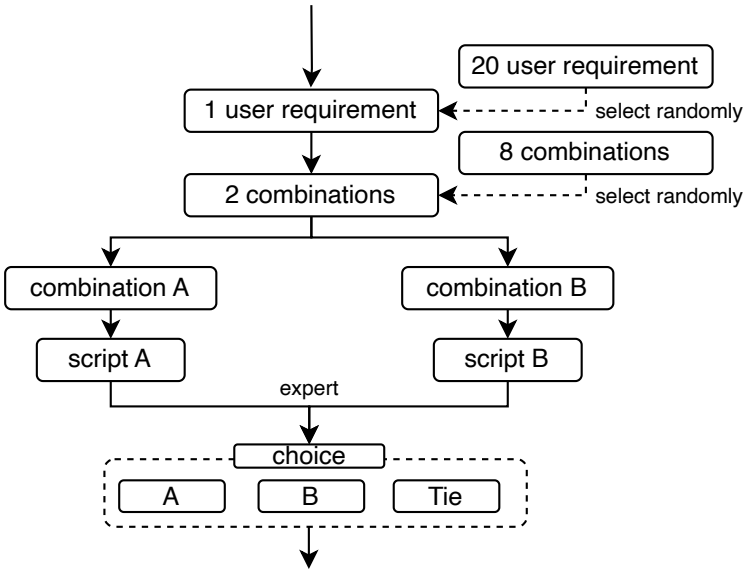


Figure 3.14: Arena-style Evaluation Pipeline

This means that a total of 160 scripts are generated in this study, and they are randomly selected from twenty test cases for pairwise evaluation by users. Each combination is treated as a competitor in the arena, and each vote is a match. Elo rating is used to update ratings and dynamically adjust the relative rankings on the leaderboard.

In the end, we invited twenty-eight RedHawk-SC experts and collected 187 votes. By examining the rating differences before and after implementing each component, we

doi:10.6342/NTU202402417

can determine the effectiveness of that component on code generation. The detailed experimental methods are described below.

3.3.4.1 Explanation and Examples

After evaluating the collected logs through the assessment above process (Figure 3.15), we gathered 187 voting results. These votes were collected by randomly selecting scripts generated by different combinations for the same user requirement and then calculating their relative rankings based on user choices.

However, the order of log entries in Figure 3.15 can affect the Elo rating calculations due to the time-sensitive nature of Elo ratings. The rating of a single combination at different points in time can vary, influencing the expected outcome of matches between different combinations and ultimately affecting the rating changes. Therefore, Elo rating calculations are inherently sequential.

Figure 3.15: Example of Arena Evaluation Log

For evaluators, their choices remain consistent regardless of when the evaluation occurs because they are unaware of the combinations used to generate the scripts during the evaluation process. To address this, we followed the bootstrap method used by chatbot arena [3, 43]. We randomly sampled one entry from the records with replacement, repeated this process one hundred times, and then calculated the Elo ratings for each combination. This process was repeated one thousand times, resulting in an Elo rating sequence composed of one thousand points for each combination. The median of this sequence is

doi:10.6342/NTU202402417

used as the basis for subsequent evaluations.

3.4 Methodology Review

This section reviews the overall process of the research methodology and its corresponding evaluation methods. Initially, we used the original documents and scripts as reference sources for early-stage data preprocessing. Each document segment was semantically split into paragraphs, attempting to correct PDF formatting errors. Using LLMs, we determined the completeness of the semantics and output the content in different formats. Post-processing ensured that each paragraph was complete. Then, each paragraph was sent for renovation to obtain the refined content.

Additionally, new scripts were generated through the refactoring process. The data obtained from these two different preprocessing components served as the reference source for subsequent RAG tasks. Users only needed to input their requirements to first generate a code outline and then produce the final script, completing the entire workflow.

For evaluation, we defined eight different combinations and invited participants for arena-style assessments. These assessments used various calculation methods to determine the relative rankings. We then analyzed the effectiveness of each component based on different ablation pairs.

doi:10.6342/NTU202402417

Chapter 4 Datasets and Experimental Setup

This chapter introduces the datasets used in this study, followed by a detailed explanation of the experimental setup employed in this research.

4.1 Application Scenarios

This study aims to develop applications of Large Language Models (LLMs) that can generate RedHawk-SC scripts based on user requirements. In practice, users need to have several key skills to use RedHawk-SC effectively: first, familiarity with Python programming to write scripts with clear logic; second, knowledge of circuit simulation; and third, the ability to understand and flexibly use thousands of APIs available on RedHawk-SC. To address these specialized needs, the workflow proposed in this study can generate scripts, making it easier and more flexible for users to utilize RedHawk-SC for various analytical applications.

doi:10.6342/NTU202402417

4.2 Dataset

The dataset used in this study is divided into two types. The first type serves as the foundation for building the vector database, enabling the Retrieval-Augmented Generation (RAG) technique to retrieve the necessary information. The second type is a validation dataset Ansys-RHSC-20, used to verify the effectiveness of each component.

4.2.1 RAG Reference

When applying the RAG technique, the content is divided into chunks, and each chunk is converted into embeddings to build the vector database. This facilitates the retrieval of appropriate reference chunks. Table 4.1 lists the primary data used in this study to build the vector database.

Table 4.1: RAG Reference Materials

Type	Format	Total	Release Date
Manual	PDF	517 pages	February 2023
API	PDF	1102 pages	February 2023
MapReduce	PDF	14 pages	April 2019
Script	Python	14 scripts	March 2024

The manual primarily explains the technical concepts of the RedHawk-SC product, including common APIs and their usage, spanning 517 pages. API documentation details all the APIs used by RedHawk-SC, including definitions and usage instructions, totaling 1102 pages. MapReduce document explains how to use the MapReduce framework within the RedHawk-SC product, covering 14 pages. All these technical documents are provided by Ansys Inc.

The scripts, designed by Ansys RedHawk-SC experts, play a crucial role in this study.

doi:10.6342/NTU202402417

As shown in Figure 4.1, each Python file consists of two parts: user requirements written in comments and the corresponding golden script that meets these requirements. Each line of code is accompanied by a comment explaining its purpose. Only a portion of the content is excerpted here. It is important to note that the same user requirement can be achieved with different scripts. There are fourteen scripts in the RAG reference, all provided by Ansys for this study.

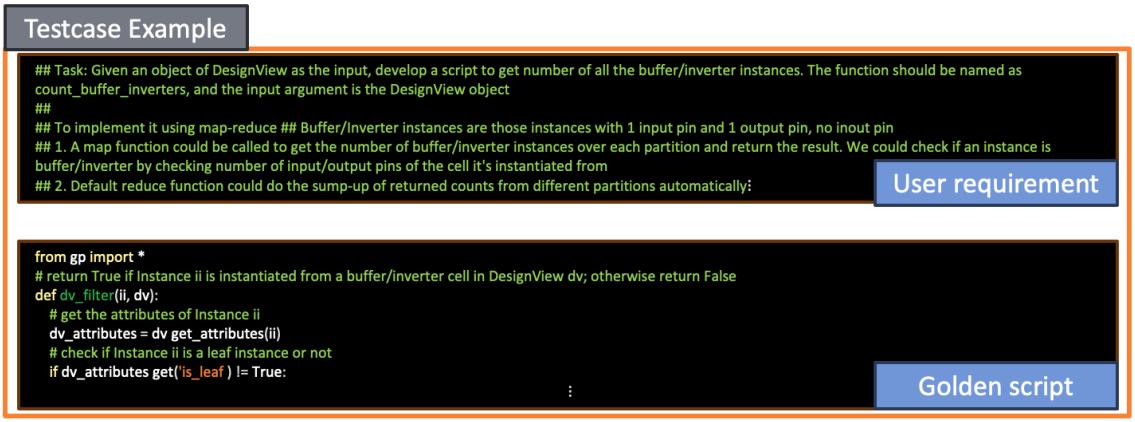


Figure 4.1: Example of Test Case with User Requirement and Golden Script

4.2.2 Ansys-RHSC-20

Ansys-RHSC-20 is a dataset designed by Ansys RedHawk-SC experts specifically for validating the tool. This dataset consists of 20 pairs of data samples, as illustrated in Figure 4.1. Each pair includes a user requirement and a corresponding gold script. Unlike the scripts in the RAG reference, where the user requirement and the corresponding script are in the same file, here they are separated into two different files: one serving as the query and the other as the reference standard answer.

doi:10.6342/NTU202402417

4.3 Evaluation Metrics

Elo rating is used as the primary evaluation metric. The Elo formula essentially estimates win probability based on rating differences, thus win rate is also employed as a supplementary metric. By setting the initial Elo rating to 0, the analysis results become more intuitive and easier to understand. All evaluation metrics and methods follow the approach used in the chatbot arena, with minor adjustments to certain parameters (Table 4.6) and refer to Section 2.3).

The reference data ratio in RAG is calculated as a percentage based on the number of reference sources. During the script generation process, 12 chunks are referenced per script, with a total of 20 test cases. Therefore, we identify the corresponding reference sources for these 240 chunks (manual, API, MapReduce, script) and calculate their percentages. This allows us to estimate the impact of different types of data by analyzing the changes in the proportions of data source combinations used.

4.4 Machine Environment

The machine environment used in this study is summarized in Table 4.3. The experiments primarily utilize APIs to interact with LLMs on this machine, as shown in Table 4.3.

doi:10.6342/NTU202402417

Table 4.2: Machine Specifications

Specification	Details
CPU	Intel Xeon E5-2698 v4 @ 2.2GHz
GPU	NVIDIA Tesla V100-SXM2-32GB
RAM	512 GB
Operating System (OS)	Ubuntu 16.04.7 LTS
Python	3.5.2

4.5 Experimental Environment

The experimental environment used in this study is detailed in Table 4.3. We utilized the LLM API and Embedding API provided by the Azure OpenAI platform. The temperature parameter ranges from a minimum of 0.0 to a maximum of 2.0. Lower values result in less variation and more accurate, consistent answers, while higher values lead to more variability and creativity. To ensure the reproducibility of our experiments, we set the temperature to 0.0. The embedding API is used to convert individual chunks into corresponding embedding vectors in the semantic space.

Table 4.3: API Versions

Specification	LLM API	Embedding API
Model	gpt-4-turbo	text-embedding-ada-002
Version	2023-07-01-preview	2023-05-15
Platform	Azure OpenAI	Azure OpenAI
Temperature	0.0	-
Embedding Dim	-	1536

We used JupyterLab [1] to set up the Python environment on our machine. The main packages used are listed in Table 4.4.

[1] JupyterLab Documentation

doi:10.6342/NTU202402417

Table 4.4: Versions of Key Python Packages

Package	Version
langchain	0.0.229
llama-index	0.10.19
openai	1.14.0
pdfminer	20191125
jupyterlab	4.0.3

4.6 Experimental Parameters

The main parameters related to the RAG technique used in this study are listed in Table 4.5, with all other values set to the default settings of LlamaIndex [2] and LangChain [3]. In the RAG technique, data retrieval is used to find the most similar chunks for reference. The ‘similarity_top_k‘ value in the table indicates the number of closest chunks to be considered by RAG. The basic approach of RAG involves splitting the content into multiple chunks based on the number of tokens, with some overlap between chunks. As shown in the table, the text is consistently split into chunks of 1024 tokens, with an overlap of 20 tokens between adjacent chunks.

Table 4.5: Key RAG Parameters

Parameter	Value	Unit
similarity_top_k	6	chunks
chunk_size	1024	tokens
chunk_overlap	20	tokens

The meaning of the Elo parameters is consistent with the discussion in the literature review (session 2.3.2). The corresponding parameter values are listed in Table 4.6. For ease of observation, the initial rating has been changed from the commonly used value of

[2] https://docs.llamaindex.ai/en/stable/
[3] https://www.langchain.com/

doi:10.6342/NTU202402417

1000 to 0. This adjustment is made because the Elo formula relies on the relative difference in ratings, so changing the initial rating merely shifts the values without affecting their relative relationships or differences.

Table 4.6: Elo Parameters

Parameter	Value
INIT_RATING	0
K	16
BASE	10
SCALE	400

4.7 Roadmap of Experiments

In Experiment 1, we employ an expert-voted arena-style evaluation to analyze the effectiveness of each constructive component using various pairwise comparison methods. Experiment 2 analyzes the importance of different types of data for code generation by examining the changes in RAG reference source ratios. This analysis will include the presence or absence of components and their corresponding rating variations.

doi:10.6342/NTU202402417

doi:10.6342/NTU202402417

Chapter 5 Experiments and Analysis

In the experiments and analysis chapter, we will detail the significance of the different combinations used in the study. We will then explain the evaluation process and methods, using Elo ratings and the Bradley-Terry [2] model to calculate each combination's performance and win rates. Each component's effectiveness in code generation will be analyzed individually. Additionally, we will analyze the importance of different data types on code generation by examining the impact of changes in the proportion of RAG reference data.

5.1 Experiment 1: Arena-style Evaluation of Component Effectiveness in Code Generation

The purpose of experiment 1 is to verify the effectiveness of the four components proposed in this study, as well as ZCoT, in domain-specific code generation applications, using Ansys RedHawk-SC as an example.

We collected 187 pairwise comparison results in a self-constructed arena. These results were then analyzed in three different ways: pairwise voting, Elo rating, and the Bradley-Terry Model. Below, we present the results of each analysis method and then perform an ablation pair analysis to evaluate the effectiveness of each component by iso-

doi:10.6342/NTU202402417

lating each combination.

5.1.1 Arena Information

We will first analyze the information collected from the 187 pairwise comparisons in the arena. During each expert evaluation, the evaluators could choose which of the two scripts was better, or declare a tie. Analyzing all voting results by option (Figure 5.1), we find that the proportions are A (44.39%), B (40.64%), and Tie (14.97%), indicating that the choice between scripts was fairly balanced. This suggests that the performance was not influenced by the script position or designation (A, B). The fact that 14.97% of all votes resulted in a tie indicates that most comparisons were decisive.

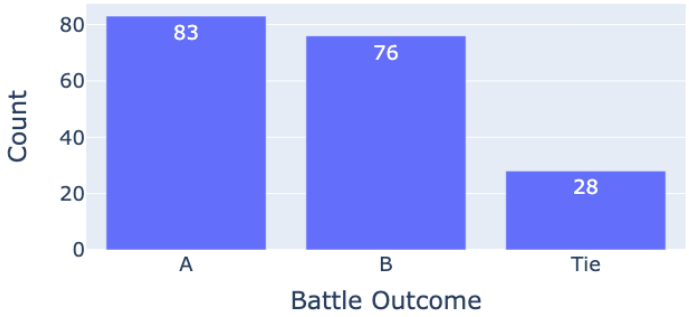


Figure 5.1: Counts of Battle Outcomes

Next, we analyze whether there is an uneven distribution in the number of times each combination participated in the arena. Since each comparison involves two combinations, the total number of participants is 374. Distributed across 8 different combinations, each should participate approximately 46.75 times (12.5%). Analyzing this distribution in Figure 5.2, we find that the largest discrepancies are 37 times (9.89%) and 54 times (14.44%), both within a 3% deviation from the average, indicating a generally even distribution. As the number of votes increases, we can expect the distribution to become even more uniform.

doi:10.6342/NTU202402417

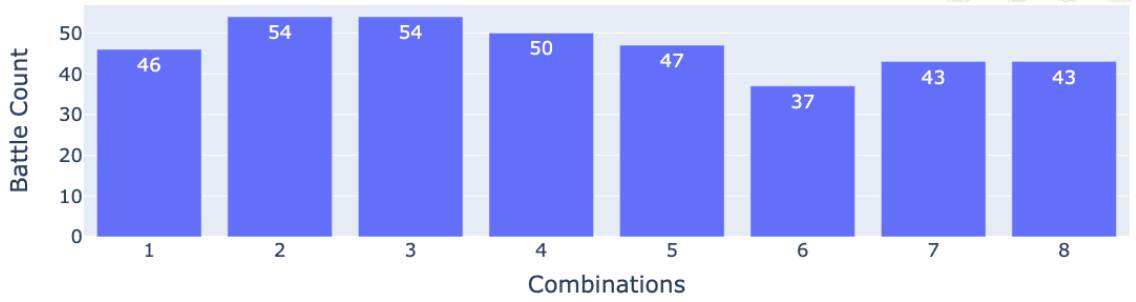


Figure 5.2: Battle Counts for Each Combination

We further analyzed all pairwise results by the number of comparisons between combinations, as shown in Figure 5.3. The statistics include all wins, losses, and ties. On average, each combination should be compared 6.68 times. The lowest number of comparisons is 3, and the highest is 13. While there is some discrepancy in the number of comparisons, it is within an acceptable range given the small sample size.

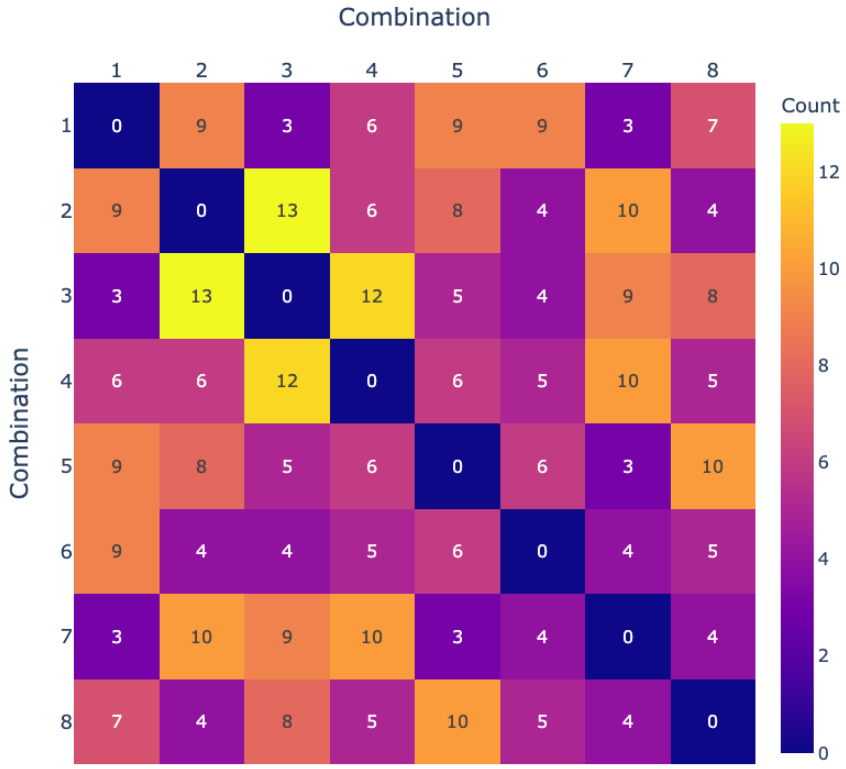


Figure 5.3: Battle Counts for Each Pair of Combinations

Additionally, we excluded all tie cases and only counted the number of wins and losses between combinations (Figure 5.4). The average number of comparisons should be 5.68 times (total number: $374 - 28 \times 2 = 318$), with the lowest being 2 and the highest

doi:10.6342/NTU202402417

being 12.

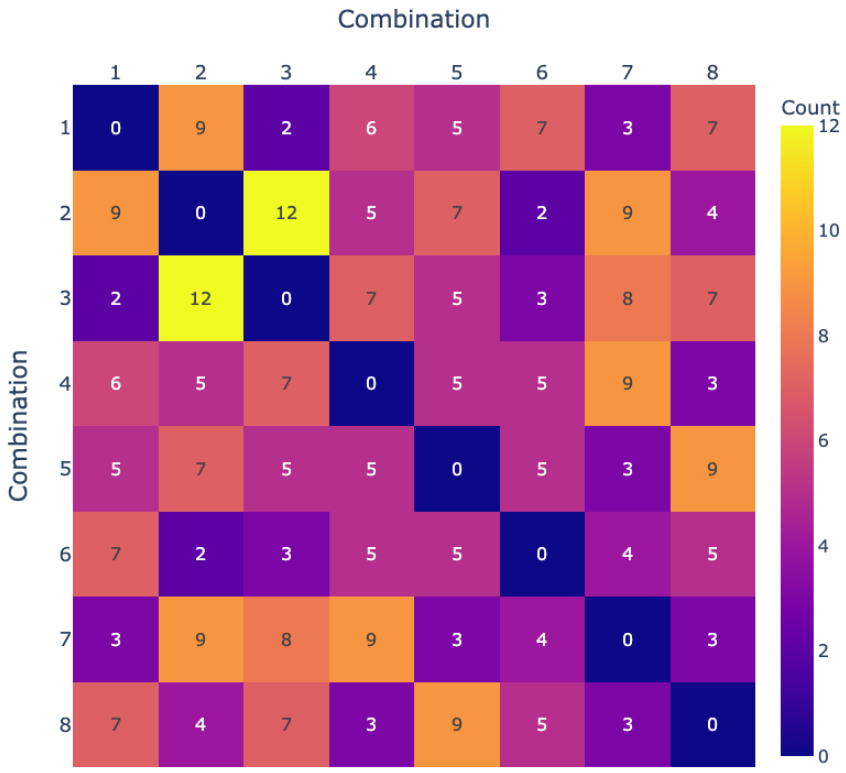


Figure 5.4: Battle Counts for Each Pair of Combinations (Excluding Ties)

When counting only the number of ties, as shown in Figure 5.5, the most notable cases are combinations 3 and 4, with 5 ties (17.86%), representing a higher proportion. This indicates that ZCoT did not have a significant advantage in this matchup. Another notable pair is combinations 1 and 5, with 4 ties (14.29%) out of 5 total matchups, indicating that these two combinations have similar performance levels.

5.1.2 Pairwise Voting

Pairwise voting involves directly comparing combinations against each other. The advantage of this method is that it eliminates the influence of unrelated combinations on the comparison, reducing potential errors. However, the disadvantage is that it often results in an insufficient number of votes, as is the case in this study. In Figure 5.6, the

doi:10.6342/NTU202402417

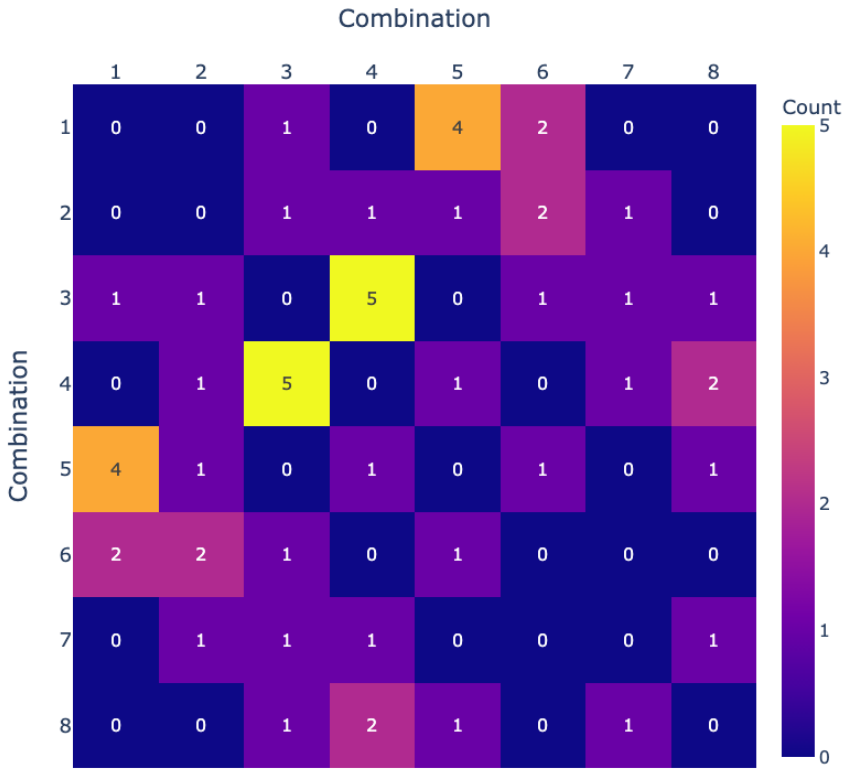


Figure 5.5: Tie Count for Each Pair of Combinations

vertical axis represents the winning combinations, while the horizontal axis represents the losing combinations. The win rate is calculated based on the number of wins between the combinations. The results include ties, where ties are counted in the total number but not as wins. For example, combinations 3 and 4 have a win rate of 0%, but as previously analyzed, they have a high tie rate. The sum of the win rate and tie rate for combination 4 is 100%. The effectiveness of the other combinations will be analyzed and discussed individually in the ablation study for each combination.

Figure 5.7 presents the pairwise voting results averaged across individual combinations. It shows that the combination of semantic splitter, IKEC, and ZCoT has the highest win rate (67.14%), while the combination using only IKEC has the lowest win rate.

Figure 5.8 extracts all ablation pairs of the same component from the pairwise voting combinations and directly averages the win rates. In terms of pairwise voting, seman-

doi:10.6342/NTU202402417

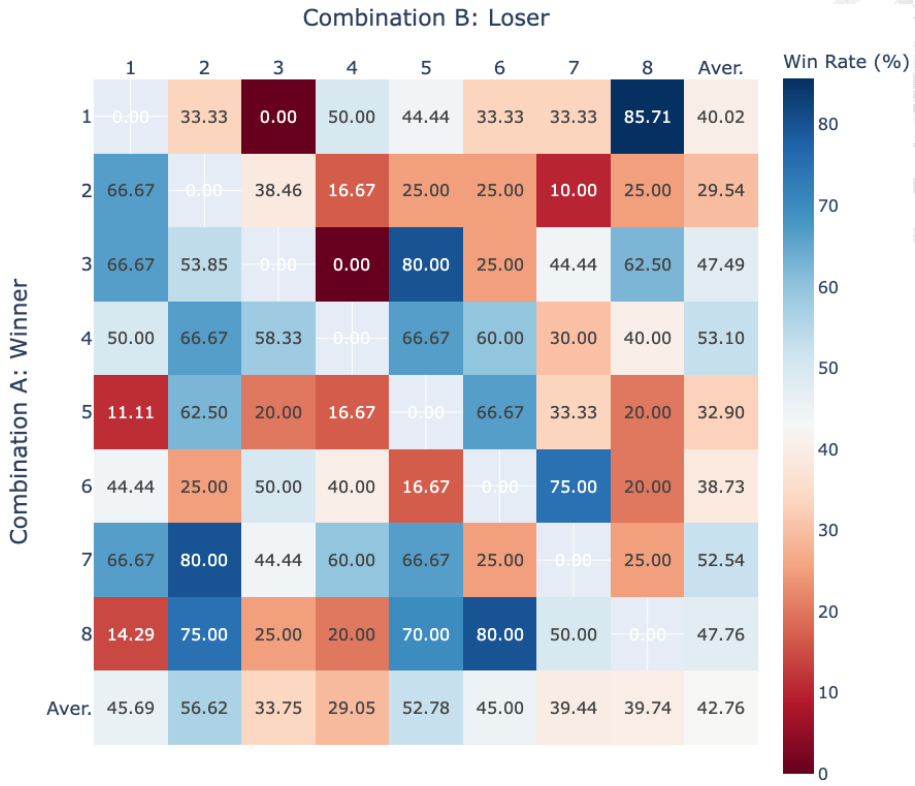


Figure 5.6: Win Rate of Combination A for All Non-tied A vs. B Battles

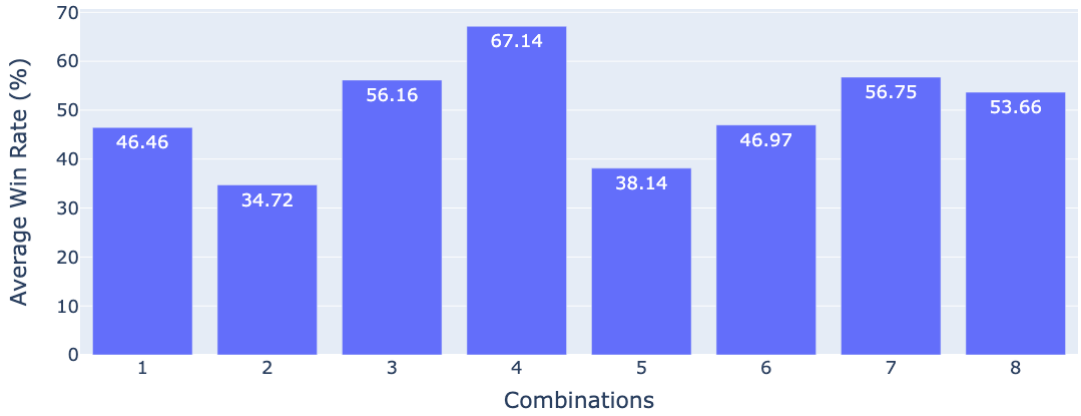


Figure 5.7: Average Win Rate Against All Other Combinations (Without Ties)

tic splitter, IKEC, and ZCoT all show slight performance improvements. However, this evaluation method only considers direct comparison results between pairs, leading to a relatively small number of votes. Consequently, the reliability of the results is limited when the sample size is insufficient.

doi:10.6342/NTU202402417

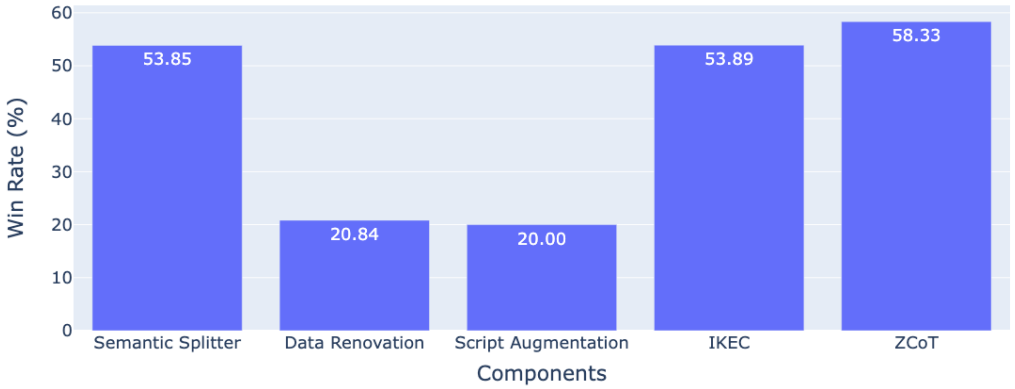


Figure 5.8: Average Win Rate for Each Component (Pairwise Voting)

5.1.3 Pairwise Voting - Even Sample

Using the same direct voting statistics, we applied stratified sampling with the Bootstrap method. The parameters were set to ‘num_sample‘ as 100 and ‘rounds‘ as 1000. This means that stratified sampling was performed on the original sample set, drawing 100 pairwise comparison results each time and recalculating the results. This process was repeated 1000 times. The average win rates of the combinations, as shown in Figure 5.9, are similar to the original win rate results.

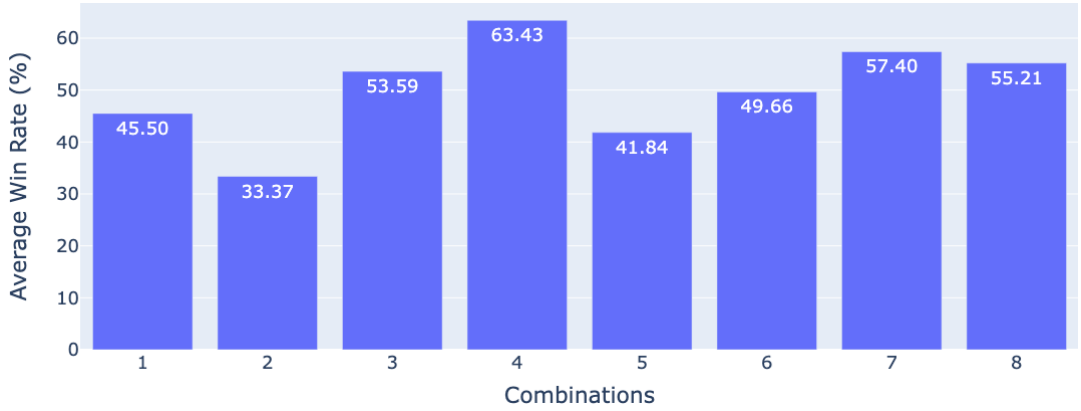


Figure 5.9: Average Win Rate Against All Other Combinations (Even Sample Size of 50 and No Ties)

doi:10.6342/NTU202402417

5.1.4 Elo Rating

Elo rating is a common method for evaluating LLM performance. Unlike direct voting, it also utilizes indirect comparison results to address the issue of insufficient votes, providing reliable relative rankings. In the original pairwise comparison results, one result is randomly selected and replaced each time, repeating this process to obtain 100 results for calculating the Elo ratings of all combinations. This process is repeated 1000 times, resulting in 1000 Elo ratings for each combination. We then use the median of these ratings for subsequent analysis. The overall performance of all combinations is shown in Figure 5.10 and Table 5.1, with combination IDs corresponding to Table 3.1. However, these results only indicate the performance of certain component combinations. We will further analyze the effectiveness of each component by examining the corresponding ablation pairs in subsequent sections.

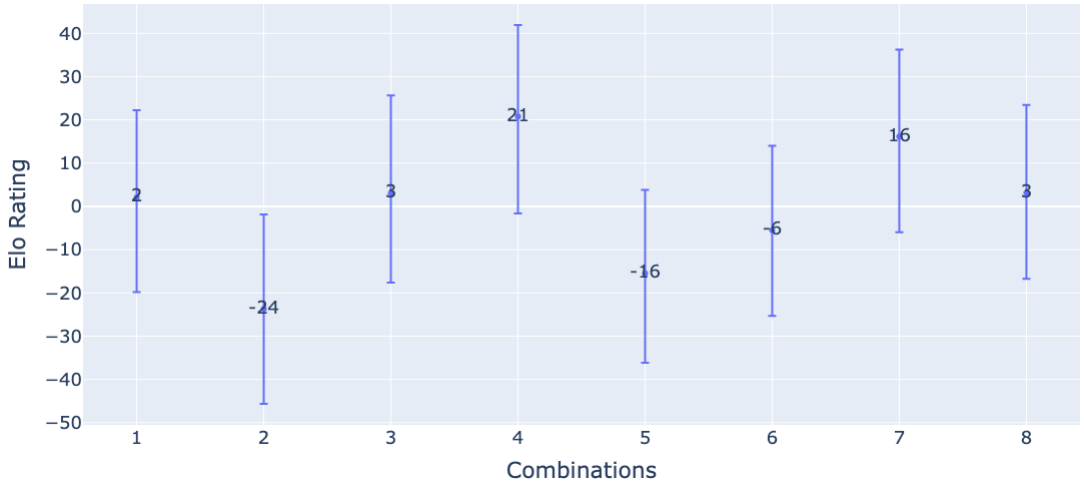


Figure 5.10: Bootstrap of Online Elo Rating Estimates

5.1.5 Elo Rating - Even Sample

The stratified sampling method is similar to what was mentioned in pairwise voting. Stratified sampling is performed on the original data to obtain 100 samples, and the Elo

doi:10.6342/NTU202402417

Table 5.1: Elo Performance of Combinations

Combination ID	Arena Information		
	Elo	***Estimated Win Rate***	***Rank***
1	8	49.05%	3
2	-51	38.59%	8
3	0	52.54%	5
4	58	64.60%	1
5	-49	37.19%	7
6	-2	49.44%	6
7	28	59.15%	2
8	7	49.44%	4

ratings of all combinations are calculated based on these samples. This process is repeated 1000 times, resulting in 1000 Elo ratings for each Combination ID. We then use the median of these ratings for subsequent analysis. The box plot of these ratings is shown in Figure 5.11. After applying the Bootstrap stratified sampling method, it is evident that the error intervals are much smaller, making it easier to use these results for subsequent analysis.

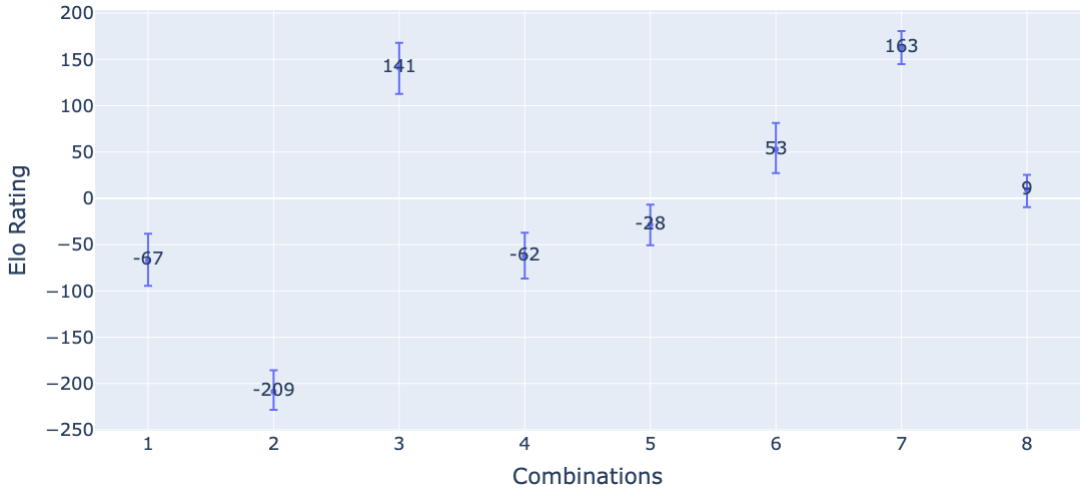


Figure 5.11: Bootstrap of Online Elo Estimates - Even Sample (Size of 100)

5.1.6 Bradley-Terry Model

Using the 187 pairwise voting results, we applied the Bradley-Terry Model to calculate the ability parameters for each combination, and then linearly transformed them into

doi:10.6342/NTU202402417

Elo ratings for easier comparison. This is shown in Figure 5.12. This one-time calculation provides more reliable results, which are generally consistent with the Elo results shown in Figure 5.10.

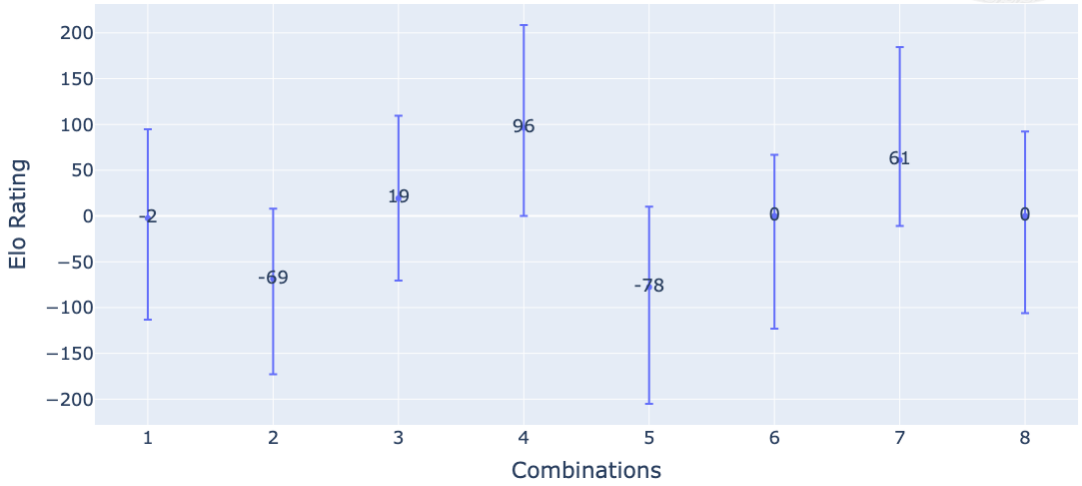


Figure 5.12: Bootstrap of Bradley-Terry Model Elo Rating Estimates

Next, based on these results, we used the Elo formula to calculate the predicted win rates between each combination, as illustrated in Figure 5.13. We then compared these predicted win rates with the actual pairwise voting results, as shown in Figure 5.14, to observe the discrepancies between the two analysis methods. The results indicate that the error between predicted and actual win rates is somewhat large, likely due to the insufficient sample size. However, the average predicted win rates, as depicted in Figure 5.15, show trends that are fully consistent with the average voting win rates.

5.1.7 Bradley-Terry Model - Even Sample

Using the same Bootstrap stratified sampling method described previously, we sampled 100 samples and then used the Bradley-Terry Model to calculate the results, linearly transforming them into Elo ratings. This process was repeated 1000 times. The results are shown in Figure 5.16. Bootstrap is particularly helpful for results with a smaller sample

doi:10.6342/NTU202402417

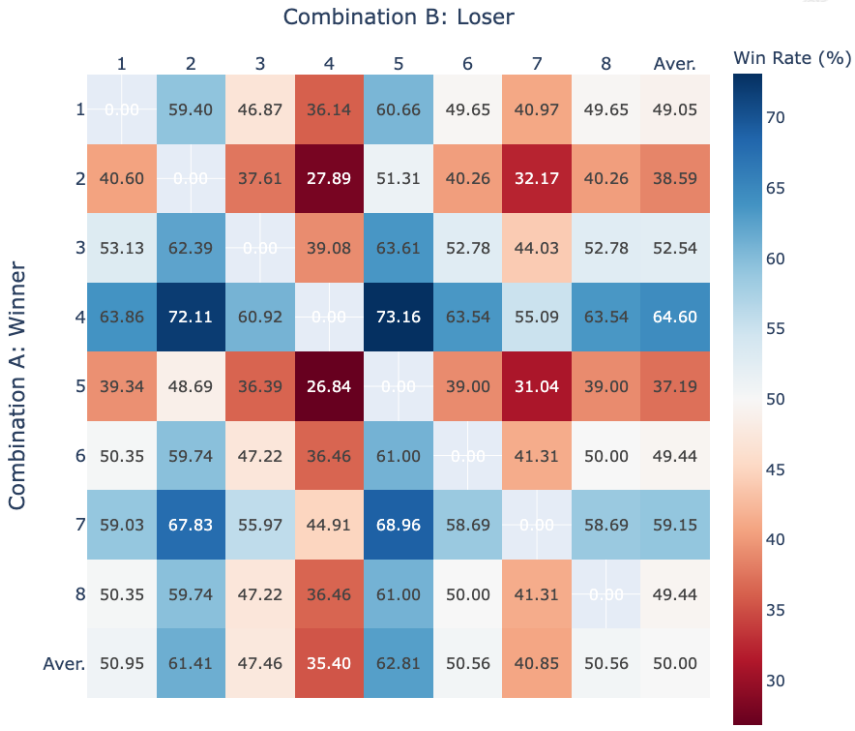


Figure 5.13: Predicted Win Rate for A vs. B Using Elo Ratings

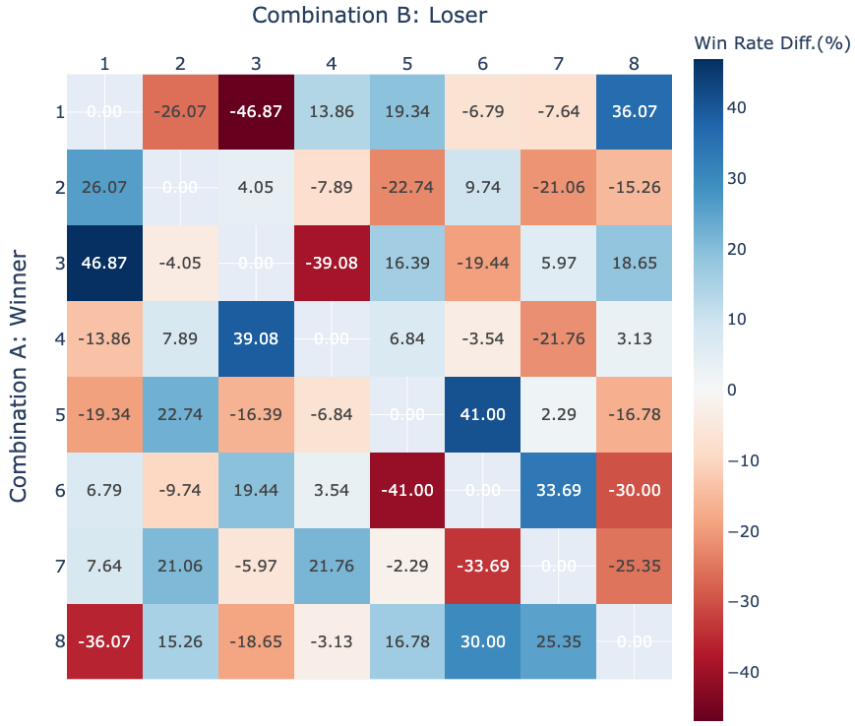


Figure 5.14: Difference Between Elo Predicted and Actual Win Rate for A vs. B

size, maintaining statistical rigor and reliability while reducing the error range. The Elo rating-based win rate table is shown in Figure 5.17, and the differences compared to the pairwise voting win rates are shown in Figure 5.18. The stratified sampling results, as

doi:10.6342/NTU202402417

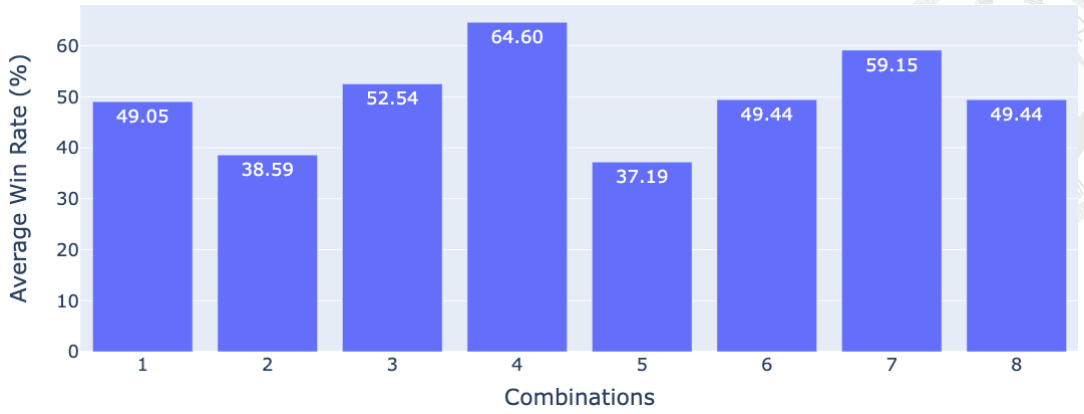


Figure 5.15: Average Predicted Win Rate Using Elo Ratings

depicted in the figure, show a deeper color representation, indicating more confidence in the estimates, although there is still a noticeable difference from the direct voting results.

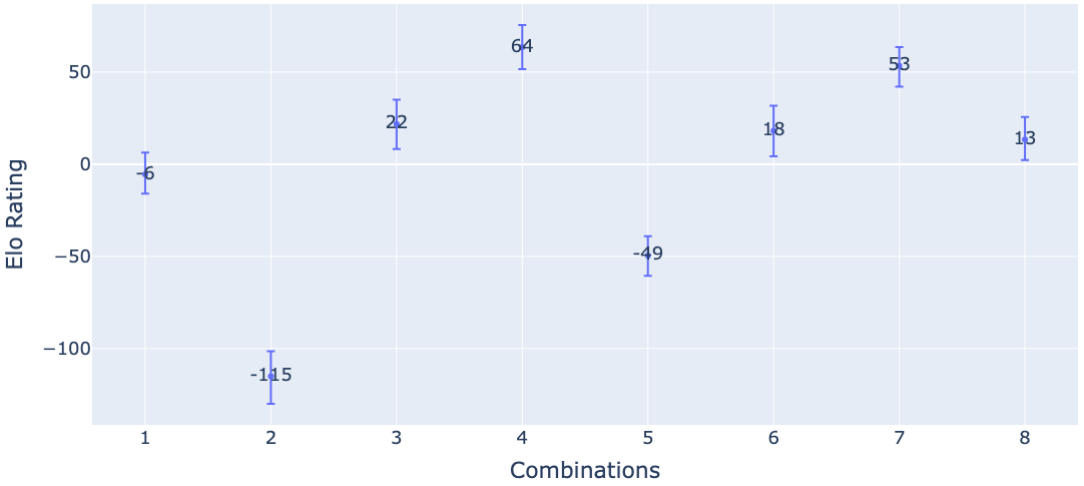


Figure 5.16: Bootstrap of Bradley-Terry Model Elo Estimates - Even Sample (Size of 100)

When observing the average win rates (Figure 5.19), the error rates are consistent with those from pairwise voting and Elo rating. By subtracting the pairwise voting win rates from the average win rates (Figure 5.20), we find that the differences are minimal, with the maximum difference being 3.54%.

Table 5.2 presents the ratings from different indirect comparison methods, with the combination IDs corresponding to Table 3.1. ”BT” stands for the Bradley-Terry Model, while ”_100” indicates that the Bootstrap method was used with 100 resampled iterations for calculation, and ”_50” indicates 50 resampled iterations. In the ablation study, we will

doi:10.6342/NTU202402417

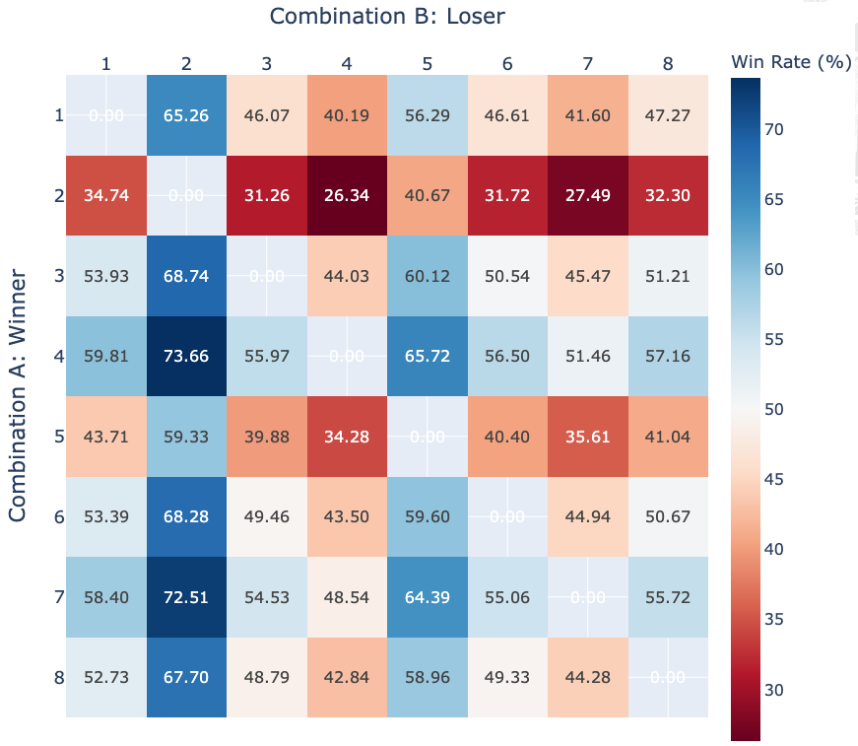


Figure 5.17: Elo Predicted Win Rate for A in A vs. B Battle - Even Sample (Size of 100)

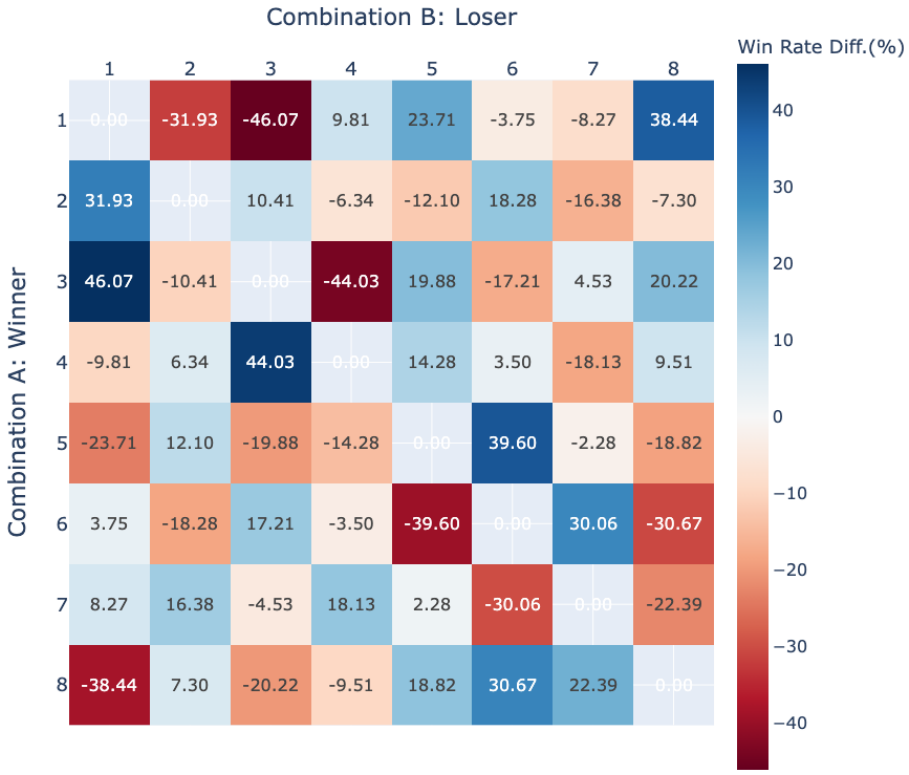


Figure 5.18: Difference Between Predicted and Actual Win Rates for A - Even Sample (Size of 100)

use this table to evaluate the effectiveness of each component.

doi:10.6342/NTU202402417

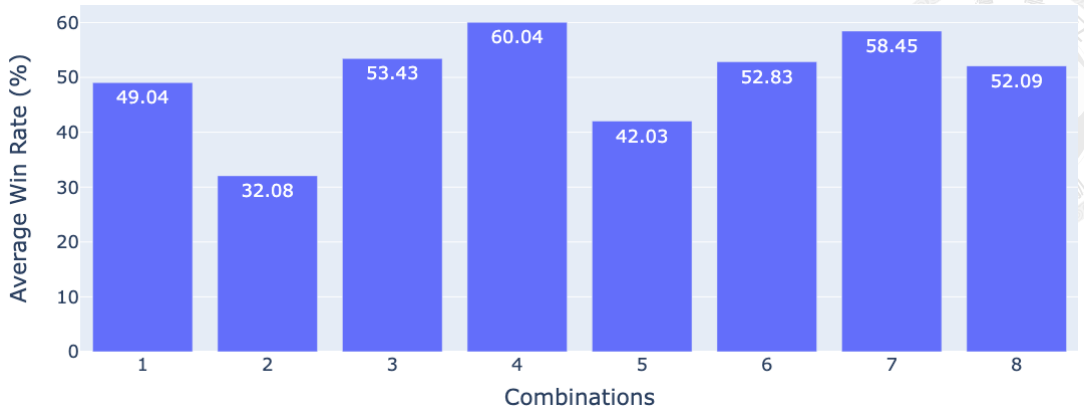


Figure 5.19: Average Win Rate for Each Combination Based on Elo Ratings - Even Sample (Size of 100)

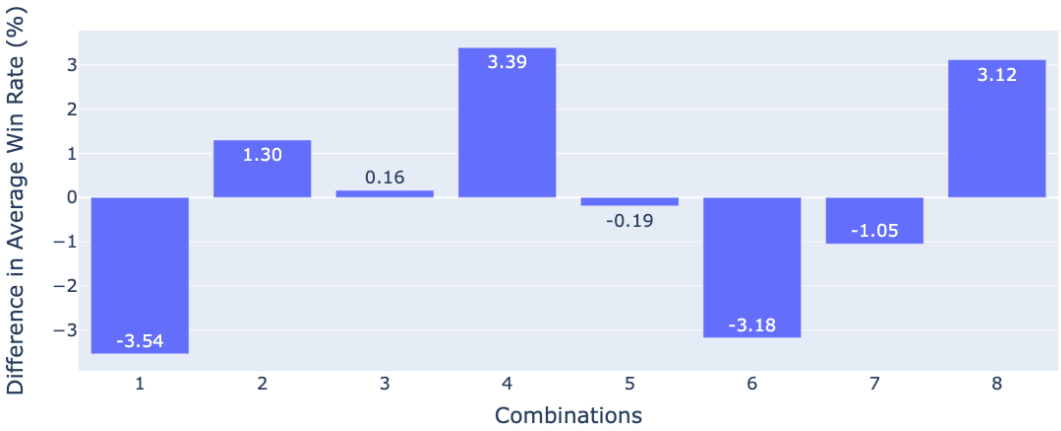


Figure 5.20: Difference in Average Win Rate (Pairwise and Bradley-Terry Model) - Even Sample (Size of 100)

Table 5.2: Overall Pairwise Comparison of Elo Ratings and Bradley-Terry Converted Elo Ratings

Combination ID	Arena Information			
	Elo	***BT***	***Elo_100***	***BT_100***
1	8	-9	-67	-6
2	-51	-80	-209	-115
3	0	18	141	22
4	58	85	-62	64
5	-49	-68	-28	-49
6	-2	-28	53	18
7	28	74	163	53
8	7	8	9	13

doi:10.6342/NTU202402417

5.1.8 Ablation Study

Previously, we discussed various calculation methods and their corresponding overall rankings. However, those rankings alone do not reveal whether a single component is effective for code generation. Therefore, we extended the overall rankings to extract ablation pairs to evaluate the effectiveness of each component.

5.1.8.1 Semantic Splitter

Table 5.3 shows that using the semantic splitter, the win rate estimated by Elo is 57.29% ($p \approx 10^{-317}$), and by pairwise voting is 53.85%. Both calculation methods indicate a relatively high and similar win rate, suggesting the effectiveness of this component. As shown in Figure 5.21, the win-loss probability ratios are very close, further supporting the component's effectiveness.

Table 5.3: Semantic Splitter - Elo Ratings and Pairwise Voting Results

Combination ID	Elo		Pairwise Voting	
	Rating	***Estimated Win Rate***	***Win Rate***	***Tie Rate***
2	-51	42.71%	38.46%	7.69%
3	0	57.29%	53.85%	-

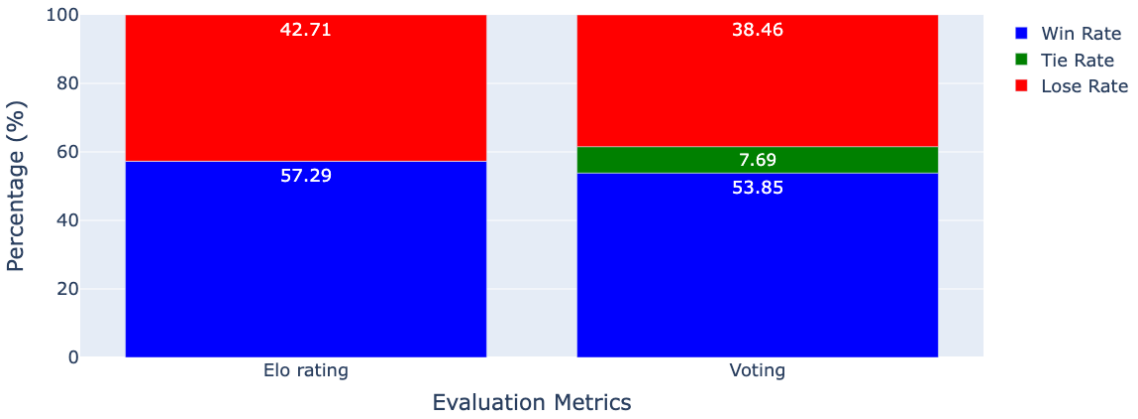


Figure 5.21: Semantic Splitter - Elo Ratings and Pairwise Voting Results

doi:10.6342/NTU202402417

5.1.8.2 Data Renovation

Similarly, for data renovation, we extracted ablation pairs from the overall ranking table to individually compare and verify the effectiveness of this single component. The two sets of ablation pairs are shown in Tables 5.4 and 5.5. After applying data renovation, the Elo ratings consistently showed performance improvements, such as for combination ID 6 with an estimated win rate of 56.72% ($p \approx 10^{-62}$) and ID 7 with an estimated win rate of 53.02% ($p \approx 10^{-96}$). However, the pairwise voting results indicated a decrease in performance (ID 6 with a 16.67% win rate and 16.67% tie rate, and ID 7 with a 25.00% win rate and 25.00% tie rate), likely due to the insufficient sample size. This suggests that while data renovation has not demonstrated a significant improvement, it remains a promising direction for potential performance enhancement. The comparisons are illustrated in Figure 5.22.

Table 5.4: Data Renovation (Pair 1) - Elo Ratings and Pairwise Voting Results

Combination ID	**Elo**		**Pairwise Voting**	
	Rating	***Estimated Win Rate***	***Win Rate***	***Tie Rate***
5	-49	43.28%	66.67%	16.67%
6	-2	56.72%	16.67%	-

Table 5.5: Data Renovation (Pair 2) - Elo Ratings and Pairwise Voting Results

Combination ID	**Elo**		**Pairwise Voting**	
	Rating	***Estimated Win Rate***	***Win Rate***	***Tie Rate***
8	7	46.98%	50.00%	25.00%
7	28	53.02%	25.00%	-

doi:10.6342/NTU202402417

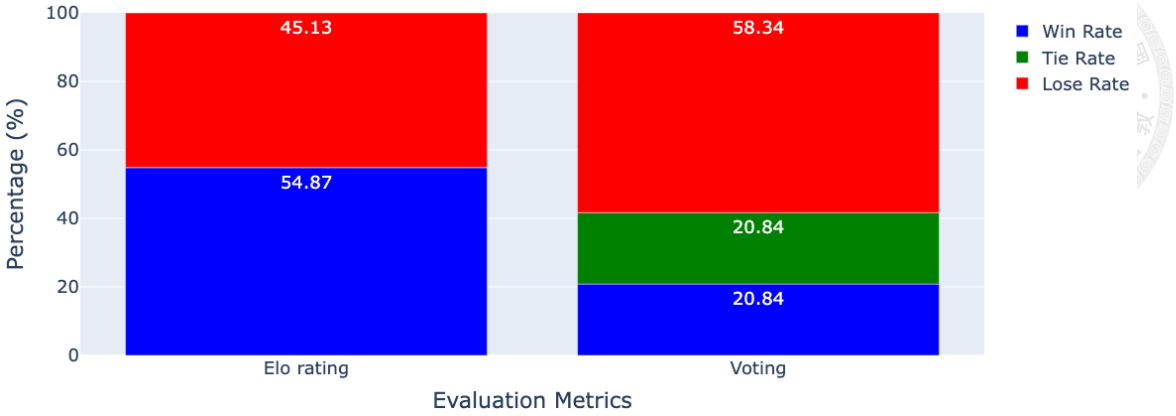


Figure 5.22: Data Renovation - Elo Ratings and Pairwise Voting Results

5.1.8.3 Script Augmentation

The results for script augmentation are shown in Table 5.6 and Figure 5.23. The Elo rating indicates a win rate of 42.99% ($p \approx 10^{-286}$), while the pairwise voting results show an even lower win rate of only 20.00%. This suggests that the use of this component has led to a decrease in performance.

Table 5.6: Script Augmentation - Elo Ratings and Pairwise Voting Results

	Elo		Pairwise Voting	
Combination ID	***Rating***	***Estimated Win Rate***	***Win Rate***	***Tie Rate***
3	0	57.01%	80.00%	0.00%
5	-49	42.99%	20.00%	-

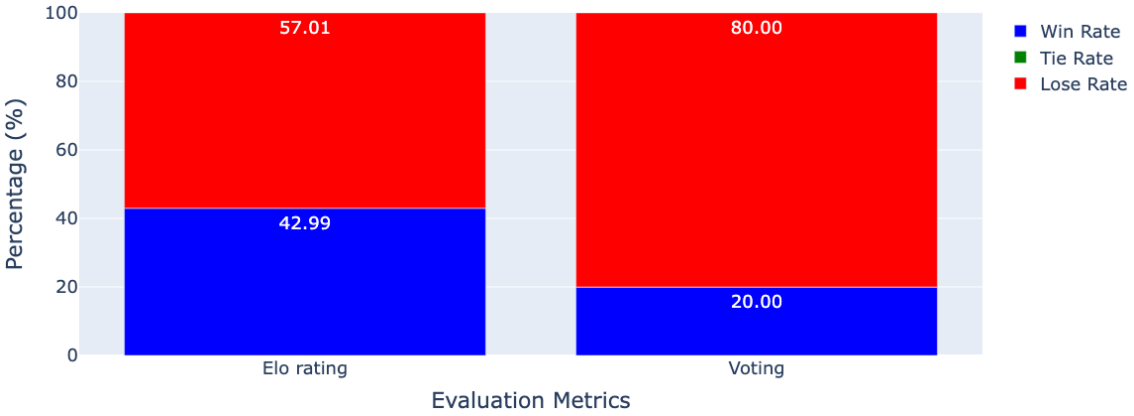


Figure 5.23: Script Augmentation - Elo Ratings and Pairwise Voting Results

Script augmentation aims to create new scripts based on existing ones, generating

doi:10.6342/NTU202402417

high-quality scripts without syntax errors. Despite this, the augmented scripts still have a high similarity to the original scripts, meaning that both pre- and post-augmentation scripts might be used.

5.1.8.4 Implicit Knowledge Expansion and Contemplation (IKEC)

There are three ablation pairs for IKEC (Tables 5.7, 5.8, 5.9). After applying IKEC, the Elo results consistently show a significant decrease in performance, such as for combination ID 2 with an estimated win rate of 41.59% ($p \approx 10^{-213}$), combination ID 5 with an estimated win rate of 42.01% ($p \approx 10^{-286}$), and Combination ID 6 with an estimated win rate of 45.69% (p-value extremely small, effectively 0).

However, this is not the case for pairwise voting. In the second-highest voting sample size combination (total: 9), IKEC has a win rate of nearly 70.00%, and in another combination, it has a win rate of 75.00%. Conversely, in the highest sample size combination (total: 10), it only has a win rate of 20.00%. This suggests that the performance of IKEC is questionable and might be advantageous in specific combinations, winning in two out of the three pairs, but overall, its performance is declining. Additionally, as shown in Figure 5.24, if we include ties in the win rate for pairwise voting, the results are exactly opposite to the Elo distribution, further confirming the instability of IKEC's performance.

Table 5.7: IKEC (Pair 1) - Elo Ratings and Pairwise Voting Results

Combination ID	Elo		Pairwise Voting	
	Rating	***Estimated Win Rate***	***Win Rate***	***Tie Rate***
1	8	58.41%	33.33%	0.00%
2	-51	41.59%	66.67%	-

doi:10.6342/NTU202402417

Table 5.8: IKEC (Pair 2) - Elo Ratings and Pairwise Voting Results

Combination ID	Elo		Pairwise Voting	
	Rating	*Estimated Win Rate*	*Win Rate*	*Tie Rate*
8	7	57.99%	70.00%	10.00%
5	-49	42.01%	20.00%	-

Table 5.9: IKEC (Pair 3) - Elo Ratings and Pairwise Voting Results

Combination ID	Elo		Pairwise Voting	
	Rating	*Estimated Win Rate*	*Win Rate*	*Tie Rate*
7	28	53.31%	25.00%	0.00%
6	-2	45.69%	75.00%	-

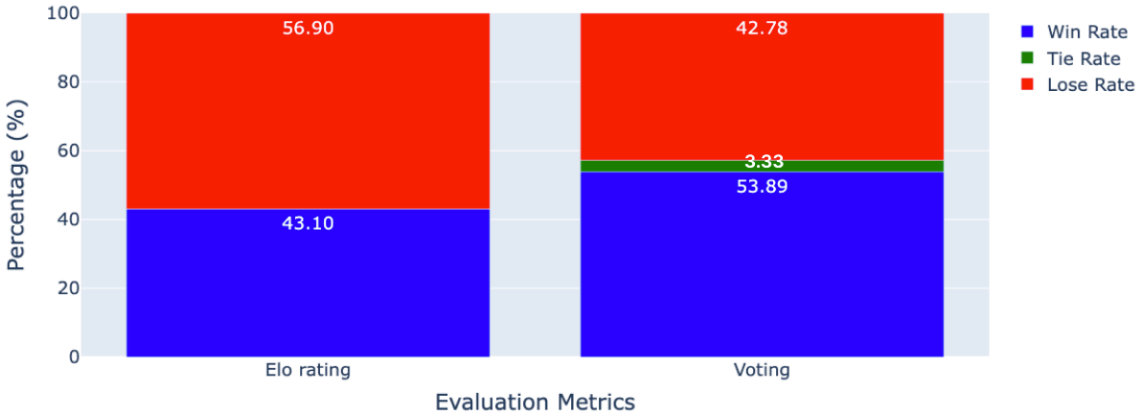


Figure 5.24: IKEC - Elo Ratings and Pairwise Voting Results

5.1.8.5 Zero-shot Chain-of-Thought (ZCoT)

Table 5.10 and Figure 5.25 demonstrate the significant effectiveness of ZCoT in both Elo and pairwise voting. For combination ID 4, which uses ZCoT, the estimated win rate by Elo is 58.27% ($p < 10^{-162}$), and the win and tie rates together account for a full 100.00% in pairwise voting. An interesting phenomenon here is that the win and loss rates in Elo are almost identical to the win and tie rates in pairwise voting. A tie indicates that both sides have some wins, suggesting that even in the worst-case scenario, ZCoT maintains an improved performance as estimated by the Elo win rate. On average, the effect is even better.

doi:10.6342/NTU202402417

Table 5.10: ZCoT - Elo Ratings and Pairwise Voting Results

Combination ID	Elo		Pairwise Voting	
	Rating	*Estimated Win Rate*	*Win Rate*	*Tie Rate*
3	0	41.74%	0.00%	41.67%
4	58	58.27%	58.33%	-

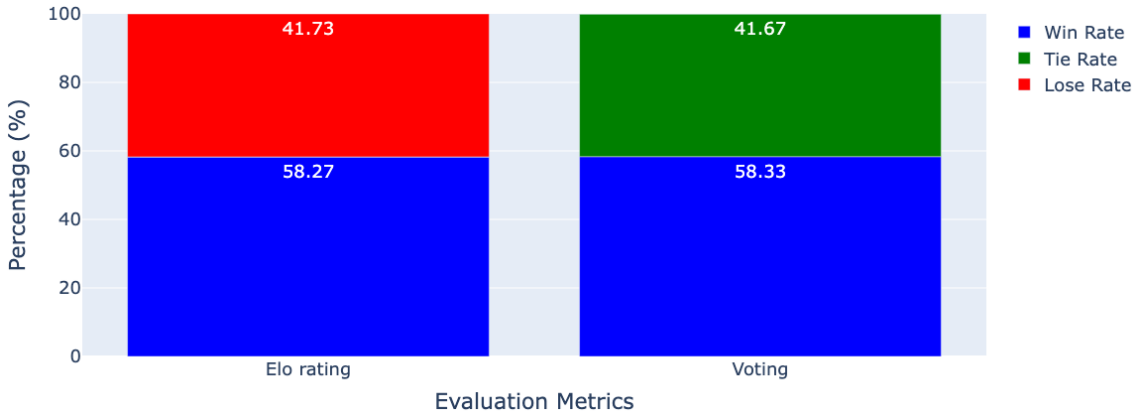


Figure 5.25: ZCoT - Elo Ratings and Pairwise Voting Results

5.1.9 Comprehensive Explanation

In experiment 1, three different evaluation methods were used, along with stratified sampling, to comprehensively assess the effectiveness of five different constructive components.

Based on previous discussions, we found that the semantic splitter is effective and consistently performs well across multiple methods. ZCoT also shows effectiveness in various results, but it has a negative impact in the Elo stratified sampling results, which warrants further investigation. Data renovation is moderately effective, while the augmented scripts from script augmentation do not improve RAG performance for this task. IKEC generally shows negative results in most evaluations, though it has positive effects in Pairwise voting.

From the overview figures 5.26 and 5.27, it is evident that the semantic splitter is the

doi:10.6342/NTU202402417

most effective component, followed by data renovation. While ZCoT is also effective, it shows some negative effects in certain cases. Both script augmentation and IKEC, on the other hand, significantly lead to negative effects.

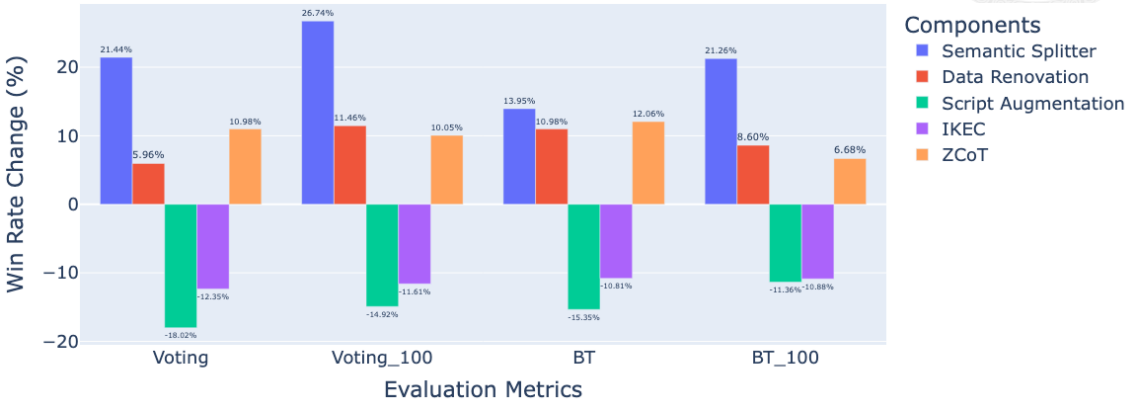


Figure 5.26: Impact of Component Ablation on Win Rate

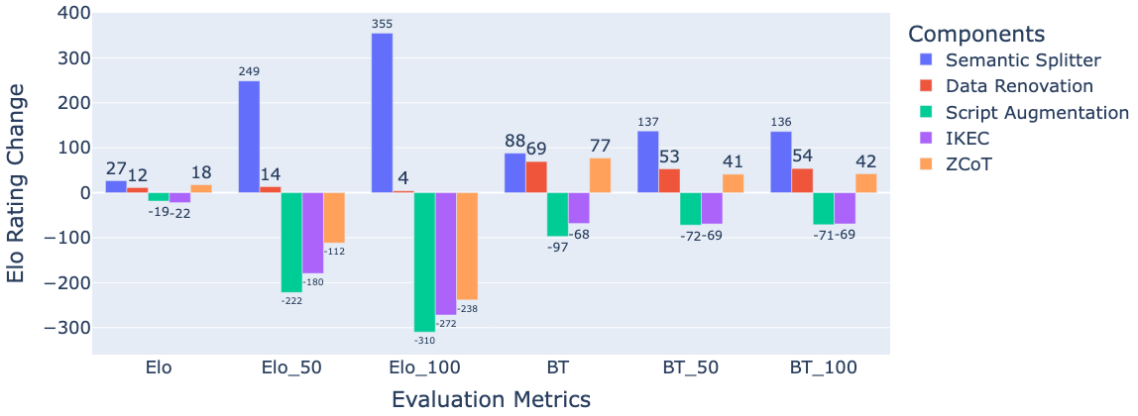


Figure 5.27: Impact of Component Ablation on Elo Rating

Next, we discuss the stratified sampling results from the most reliable Bradley-Terry Model, focusing on numerical analysis (Figure 5.26). The semantic splitter improves performance win rates by 21.26%. Data renovation shows an 8.60% win rate improvement, while ZCoT improves the win rate by 6.68%. Conversely, script augmentation and IKEC exhibit significant negative effects, with win rate decreases of 11.36% and 10.88%, respectively.

Based on the results, two of our proposed components outperform the existing ZCoT component. According to the results of this ablation study, combining semantic splitter,

doi:10.6342/NTU202402417

data renovation, and ZCoT components could potentially yield the best performance.

Additionally, in the original RAG framework, each chunk was set to contain 1024 characters by default. After applying our semantic 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 the splitter-processed chunks.

5.2 Experiment 2: Comparison of RAG Data Source Proportions in Different Component Combinations

The goal of experiment 2 is to investigate which data is most helpful in improving performance. When components are applied, the proportion of the corresponding RAG reference data changes along with the original rating variations. By analyzing these changes, we can infer which data is most beneficial for RAG in generating scripts for specific domains. The summary of all proportions is shown in Table 5.11.

5.2.1 Data Preprocessing

The data preprocessing techniques applied alter the data itself, but the prompts used to generate scripts remain unchanged. Therefore, changes in the proportion of data sources are a significant factor affecting performance. The semantic splitter fundamentally changes the division of data texts, leading to substantial changes in proportions. An unusual phenomenon observed is that the proportions of the three technical documents significantly

doi:10.6342/NTU202402417

Table 5.11: Distribution of Reference Percentage in RAG Across Document Types

Combination ID	Reference Average (Source Count Percentage (%)				
	Manual	***API***	***MapReduce***	***Script***	***Augmented Script***
1	19.58	27.08	42.08	11.25	-
2	14.58	22.50	52.50	10.42	-
3	6.67	5.42	0.42	87.50	-
4	7.09	6.67	0.84	85.42	-
5	2.08	3.75	0.00	37.50	56.67
6	1.25	12.09	0.00	35.42	51.25
7	1.67	14.59	0.00	37.92	45.83
8	3.33	4.59	0.00	38.34	53.75
9	8.75	5.42	0.42	85.42	-
10	3.34	29.59	0.42	66.67	-
11	4.59	27.50	0.84	67.09	-
12	5.00	30.00	0.84	64.17	-
13	0.42	11.25	0.00	35.84	52.50
14	1.67	2.92	0.00	32.92	62.50
15	10.84	3.33	14.59	28.75	42.50
16	16.67	17.92	57.08	8.34	0

decreased, while the proportion of scripts greatly increased (Table 5.11). This indicates that the chunks of technical documents used in traditional RAG data division methods may not be relevant to the current tasks. Figure 5.28 illustrates the dramatic changes in data proportions, with all three technical documents showing decreased proportions, while the proportion of scripts increased significantly. Since the semantic splitter improves ratings, this suggests that the increased proportion of scripts is likely a major factor contributing to the performance enhancement.

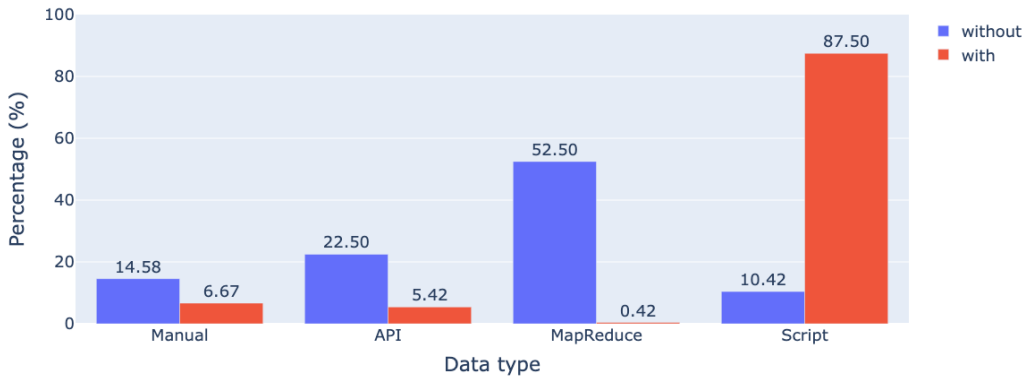


Figure 5.28: Semantic Splitter - Distribution of Reference in RAG

doi:10.6342/NTU202402417

Additionally, as shown in Table 5.11, combinations 14 and 15 differ only by the inclusion of the semantic splitter. After applying the semantic splitter, the proportions of MapReduce, Manual, and API were significantly reduced, and the proportion of scripts increased. This implies that the original RAG method could not adequately reflect the topic of the chunks, leading to the retrieval of irrelevant chunks. After applying the semantic splitter, it is more likely to find relevant content from the technical documents. Compared to irrelevant content, the scripts provide more substantial assistance.

Data renovation focuses on refreshing the text within predefined chunks, so it is expected that the changes in data proportions would not be substantial. This technique helps the chunks to be more focused on specific topics. Figure 5.29 shows that the proportion of API references increased while the proportion of script references decreased. Despite the decrease in the proportion of the most beneficial reference source, scripts, the performance ratings were maintained or even improved, indicating the usefulness of the data renovation method. However, improving the ”post-renovation” validation process will be crucial for further enhancements.

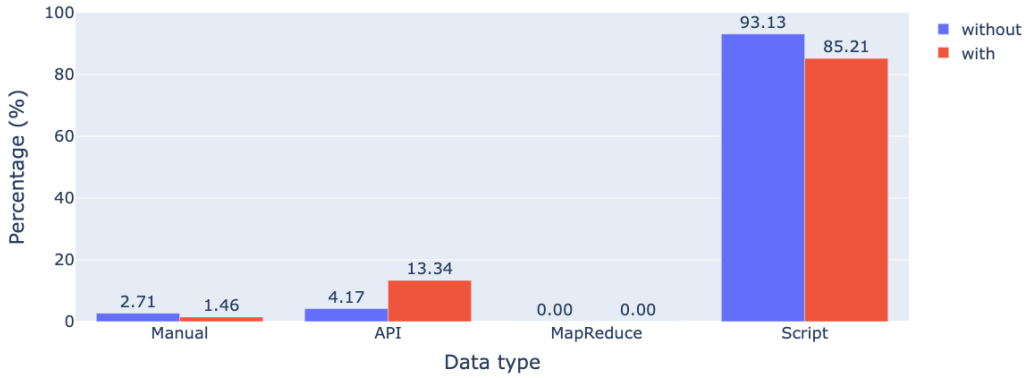


Figure 5.29: Data Renovation - Distribution of Reference in RAG

As seen in Table 5.11, the application of data renovation consistently results in a decrease in the proportion of manual references, a significant increase in the proportion of API references, and a decrease in the proportion of script references. The decrease in

doi:10.6342/NTU202402417

the proportion of high-value script references, yet maintaining or improving performance ratings, suggests that data renovation effectively helps the data embedding to reflect its intended content better.

After script augmentation, the proportion of script references increased as shown in Figure 5.30, since the augmented scripts were included. However, despite the increased proportion, there was a significant decline in performance. This suggests that while the augmented scripts add some novelty based on the original scripts, their relative value is much lower. Because the similarity between pre- and post-augmentation scripts is generally high, the RAG process has a high likelihood of referencing either script, leading to redundant use of reference slots. This observation indicates that instead of referencing similar scripts, it is more beneficial to reference different scripts, even if they are not highly relevant, to improve performance.

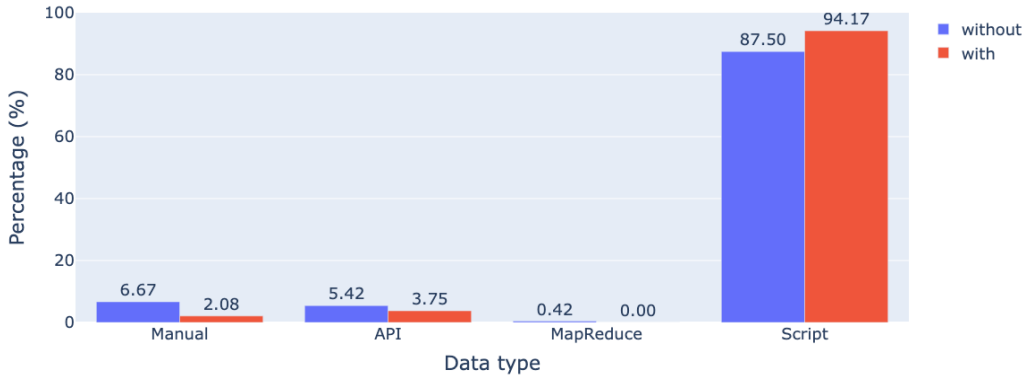


Figure 5.30: Script Augmentation - Distribution of Reference in RAG

5.2.2 Prompt Techniques

Next, we examine the impact of prompt techniques on the proportion of RAG reference data. It is anticipated that prompt techniques should not significantly affect the proportion of RAG reference data. The main reason is that the query remains the same, and any differences arise only from the slightly different script summaries generated by us-

doi:10.6342/NTU202402417

ing different prompt techniques. These summaries become the subsequent queries, hence only slightly influencing the results. Figures 5.31 and 5.32 show that the proportion differences before and after using the prompt techniques are minimal. This indicates that the performance variations are related to the prompt technique itself, rather than the data.

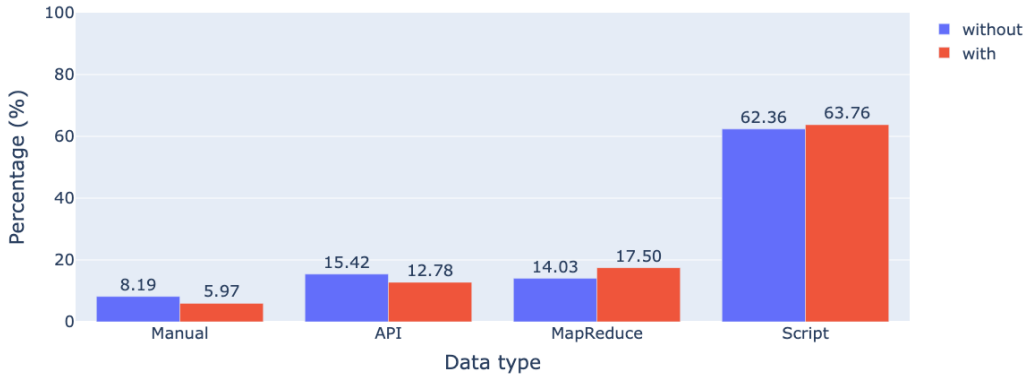


Figure 5.31: IKEC - Distribution of Reference in RAG

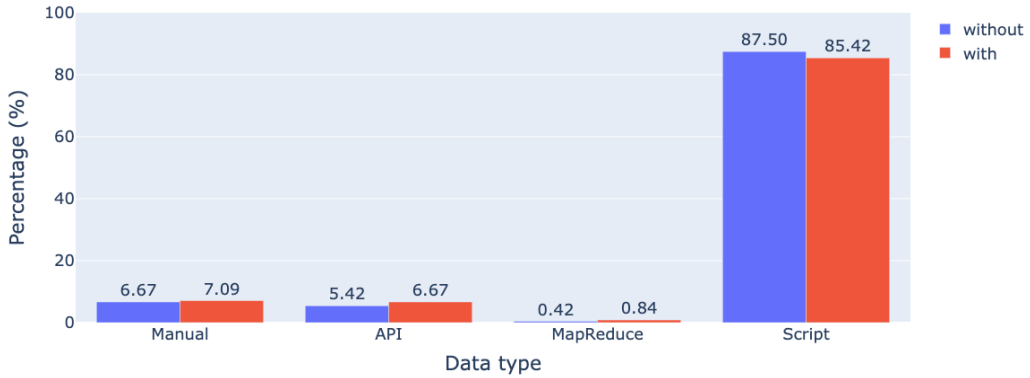


Figure 5.32: ZCoT - Distribution of Reference in RAG

doi:10.6342/NTU202402417

Chapter 6 Conclusions

We propose four constructive components to improve RAG performance for Large Language Models (LLMs) in specific-domain problems: semantic splitter, data renovation, script augmentation, and IKEC. Additionally, we reference one existing constructive component, ZCoT, to facilitate comparison with current methods. Semantically segmenting text and renovating content with high LLM confidence levels facilitate the improvement of topic-focused embeddings during the data retrieval process for RAG. The novel application of semantic 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 187 pairwise comparisons. The results demonstrate that the semantic splitter and data renovation components notably enhance the code generation performance for RedHawk-SC in MapReduce applications within specialized domains. Specifically, the improvement attributed to these components is quantifiably greater than the ZCoT (which shows a 6.68% win rate improvement). The semantic splitter component achieves a 21.26% win rate improvement, and the data renovation component achieves an 8.60% win rate improvement, significantly outperforming the ZCoT.

doi:10.6342/NTU202402417

6.1 Summary of Findings

Expert evaluations determined that three constructive components are effective for domain-specific code generation. The performance improvements indicate that the techniques proposed components can help RAG find more relevant chunks, suggesting that data preprocessing techniques will be a promising direction.

Additionally, this study highlights that providing high-quality scripts is the most critical factor in improving performance, far outweighing the other three types of technical documentation, such as manuals, APIs, and MapReduce. This indicates that in the domain of code generation using large language models, providing practical, executable content is much more effective than merely conceptual information.

6.2 Future Prospects

We aim to experiment with our approach on a wider variety of LLMs in the future, particularly as recent models have demonstrated the ability to maintain exceptional performance even with fewer parameters (e.g., Gemma2 [34]). Even though computational resources may not be abundant, open-source models' capabilities gradually align with those of closed-source models, and the number of parameters is steadily decreasing. Thus, focusing on open-source models is feasible and crucial for advancing academic research in this field.

In the task of Register Transfer Language (RTL) code generation, datasets designed by human experts have shown that the average simulation pass rate for code generated by LLMs stands at 5.30%, with an average of 26.70% [37]. Our research dataset, com-

doi:10.6342/NTU202402417

prising medium to difficult problems designed by experts, generates code for the more resource-constrained RedHawk-SC, likely resulting in even lower simulation pass rates. This poses significant challenges for evaluation, as any changes to processes or parameters necessitate reevaluation by human experts. Therefore, future work could involve designing simpler datasets to increase the simulation pass rate from 50-60% to 80-90%, with the entire evaluation process being automated. This represents a promising direction for future efforts.

Data augmentation is crucial for data-scarce-specific domains and directly impacts the generative performance. We can take inspiration from the data augmentation methods used by RTLCoder [27] and Llama3.1 [6]. To address the scarcity of benchmarks and the difficulty of testing, we could shift our generation target to Verilog, which has relatively more resources and a standardized VerilogEval [26] benchmark. This would allow us to compare our work with various studies focusing on RTL code.

From a technical standpoint, integrating fine-tuning with RAG [1] or GraphRAG [7] could be explored. Additionally, using Compile and Link (CL) Call to leverage reflection for self-correction could further enhance the performance of our research.

Our study concludes that post-renovation could also be a significant point for performance improvement. Future efforts could focus on this aspect, utilizing the best combination of semantic splitter, data renovation, and ZCoT, and then evaluating their performance.

doi:10.6342/NTU202402417

doi:10.6342/NTU202402417

References

[1] A. Balaguer, V. Benara, R. L. de Freitas Cunha, R. de M. Estevão Filho, T. Hendry, D. Holstein, et al. Rag vs fine-tuning: Pipelines, tradeoffs, and a case study on agriculture. arXiv preprint arXiv:2401.08406, 2024.

[2] R. A. Bradley and M. E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. Biometrika, 39(3/4):324–345, 1952.

[3] 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. arXiv preprint arXiv:2403.04132, 2024.

[4] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. Communications of the ACM, 51(1):107–113, 2008.

[5] D. Dua, S. Gupta, S. Singh, and M. Gardner. Successive prompting for decomposing complex questions. arXiv preprint arXiv:2212.04092, 2022.

[6] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, et al. The llama 3 herd of models. arXiv preprint arXiv:2407.21783, 2024.

[7] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, et al. From local

doi:10.6342/NTU202402417

to global: A graph rag approach to query-focused summarization. arXiv preprint arXiv:2404.16130, 2024.

[8] B. Efron. Bootstrap methods: another look at the jackknife. In Breakthroughs in statistics: Methodology and distribution, pages 569–593. Springer, 1992.

[9] A. Eliseev and D. Mazur. Fast inference of mixture-of-experts language models with offloading. arXiv preprint arXiv:2312.17238, 2023.

[10] A. Elo. The Rating of Chessplayers: Past and Present. Ishi Press International, 2008.

[11] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, et al. Retrieval-augmented generation for large language models: A survey. arXiv preprint arXiv:2312.10997, 2024.

[12] M. E. Glickman and A. C. Jones. Rating the chess rating system. CHANCE-BERLIN THEN NEW YORK-, 12:21–28, 1999.

[13] A. Gu and T. Dao. Mamba: Linear-time sequence modeling with selective state spaces. arXiv preprint arXiv:2312.00752, 2024.

[14] 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), pages 1–6. IEEE, 2023.

[15] C.-Y. Hsieh, C.-L. Li, C.-K. Yeh, H. Nakhost, Y. Fujii, A. Ratner, et al. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. arXiv preprint arXiv:2305.02301, 2023.

[16] D. Huang, Q. Bu, J. M. Zhang, M. Luck, and H. Cui. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation. arXiv preprint arXiv:2312.13010, 2024.

doi:10.6342/NTU202402417

[17] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, et al. Mixtral of experts. arXiv preprint arXiv:2401.04088, 2024.

[18] X. Jiang, Y. Dong, L. Wang, Z. Fang, Q. Shang, G. Li, et al. Self-planning code generation with large language models. arXiv preprint arXiv:2303.06689, 2023.

[19] Z. Jiang, F. F. Xu, L. Gao, Z. Sun, Q. Liu, J. Dwivedi-Yu, et al. Active retrieval augmented generation. arXiv preprint arXiv:2305.06983, 2023.

[20] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. Large language models are zero-shot reasoners. arXiv preprint arXiv:2205.11916, 2023.

[21] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. arXiv preprint arXiv:2005.11401, 2021.

[22] C. Li, J. Liang, A. Zeng, X. Chen, K. Hausman, D. Sadigh, et al. Chain of code: Reasoning with a language model-augmented code emulator. arXiv preprint arXiv:2312.04474, 2023.

[23] C. Li, J. Wang, Y. Zhang, K. Zhu, W. Hou, J. Lian, et al. Large language models understand and can be enhanced by emotional stimuli. arXiv preprint arXiv:2307.11760, 2023.

[24] Y.-C. Lin, A. Kumar, N. Chang, W. Zhang, M. Zakir, R. Apte, et al. Novel pre-processing technique for data embedding in engineering code generation using large language model. In 1st IEEE International Workshop on LLM-Aided Design, 2024.

[25] Y.-C. Lin, A. Kumar, N. Chang, W. Zhang, M. Zakir, R. Apte, et al. Novel pre-

doi:10.6342/NTU202402417

processing technique for data embedding in engineering code generation using large language model. arXiv preprint arXiv:2311.16267, 2024.

[26] M. Liu, N. Pinckney, B. Khailany, and H. Ren. Verilogeval: Evaluating large language models for verilog code generation. arXiv preprint arXiv:2309.07544, 2023.

[27] S. Liu, W. Fang, Y. Lu, Q. Zhang, H. Zhang, and Z. Xie. Rtlcoder: Outperforming gpt-3.5 in design rtl generation with our open-source dataset and lightweight solution. In 1st IEEE International Workshop on LLM-Aided Design, 2024.

[28] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, et al. Large language models: A survey. arXiv preprint arXiv:2402.06196, 2024.

[29] A. Mitra, L. D. Corro, S. Mahajan, A. Codas, C. Simoes, S. Agarwal, et al. Orca 2: Teaching small language models how to reason. arXiv preprint arXiv:2311.11045, 2023.

[30] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, et al. Codegen: An open large language model for code with multi-turn program synthesis. arXiv preprint arXiv:2203.13474, 2023.

[31] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, et al. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023.

[32] B. Peng, C. Li, P. He, M. Galley, and J. Gao. Instruction tuning with gpt-4. arXiv preprint arXiv:2304.03277, 2023.

[33] M. Schäfer, S. Nadi, A. Eghbali, and F. Tip. An empirical evaluation of using large language models for automated unit test generation. arXiv preprint arXiv:2302.06527, 2023.

doi:10.6342/NTU202402417

[34] G. Team, M. Riviere, S. Pathak, P. G. Sessa, C. Hardin, S. Bhupatiraju, et al. Gemma 2: Improving open language models at a practical size. arXiv preprint arXiv:2408.00118, 2024.

[35] 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.

[36] 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.

[37] Y.-D. Tsai, M. Liu, and H. Ren. Rtlfixer: Automatically fixing rtl syntax errors with large language models. In Proceedings of the 61th ACM/IEEE Design Automation Conference (DAC'24), 2024.

[38] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, et al. Attention is all you need. Advances in neural information processing systems, 30, 2017.

[39] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis. Efficient streaming language models with attention sinks. arXiv preprint arXiv:2309.17453, 2023.

[40] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, et al. React: Synergizing reasoning and acting in language models. arXiv preprint arXiv:2210.03629, 2023.

[41] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, et al. A survey of large language models. arXiv preprint arXiv:2303.18223, 2023.

[42] H. S. Zheng, S. Mishra, X. Chen, H.-T. Cheng, E. H. Chi, Q. V. Le, et al. Take a step

doi:10.6342/NTU202402417

back: Evoking reasoning via abstraction in large language models. arXiv preprint arXiv:2310.06117, 2023.

[43] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. arXiv preprint arXiv:2306.05685, 2023.

doi:10.6342/NTU202402417