Exploring Hybrid Question Answering via Program-based Prompting

Qi Shi[*], **Han Cui**[*], **Haofeng Wang**, **Qingfu Zhu**, **Wanxiang Che**[†], **Ting Liu**
Research Center for Social Computing and Information Retrieval
Harbin Institute of Technology, Harbin, China
{qshi,hcui,hfwang,qfzhu,car,tliu}@ir.hit.edu.cn

Abstract

Question answering over heterogeneous data requires reasoning over diverse sources of data, which is challenging due to the large scale of information and organic coupling of heterogeneous data. Various approaches have been proposed to address these challenges. One approach involves training specialized retrievers to select relevant information, thereby reducing the input length. Another approach is to transform diverse modalities of data into a single modality, simplifying the task difficulty and enabling more straightforward processing. In this paper, we propose HPROPRO, a novel program-based prompting framework for the hybrid question answering task. HPROPRO follows the code generation and execution paradigm. In addition, HPROPRO integrates various functions to tackle the hybrid reasoning scenario. Specifically, HPROPRO contains function declaration and function implementation to perform hybrid information-seeking over data from various sources and modalities, which enables reasoning over such data without training specialized retrievers or performing modal transformations. Experimental results on two typical hybrid question answering benchmarks HybridQA and MultiModalQA demonstrate the effectiveness of HPROPRO: it surpasses all baseline systems and achieves the best performances in the few-shot settings on both datasets[1].

1 Introduction

Question answering systems (Pasupat and Liang, 2015; Rajpurkar et al., 2016; Goyal et al., 2017) have attracted significant attention and made considerable progress in recent years. However, real-world data often exists in diverse formats and originates from multiple sources. Consequently, researchers turn their focus to the hybrid question answering (HQA) task (Chen et al., 2020b; Talmor et al., 2020), which necessitates mixed reasoning across various types of data. The HQA task is challenging due to the vast amount of information and the organic coupling of heterogeneous data sources. Reasoning over such diverse data requires the ability to understand multiple data types simultaneously. For instance, as depicted in Figure 1, the model must engage in reasoning over both the table and the extensive passages and images linked in hyperlinks to make accurate predictions.

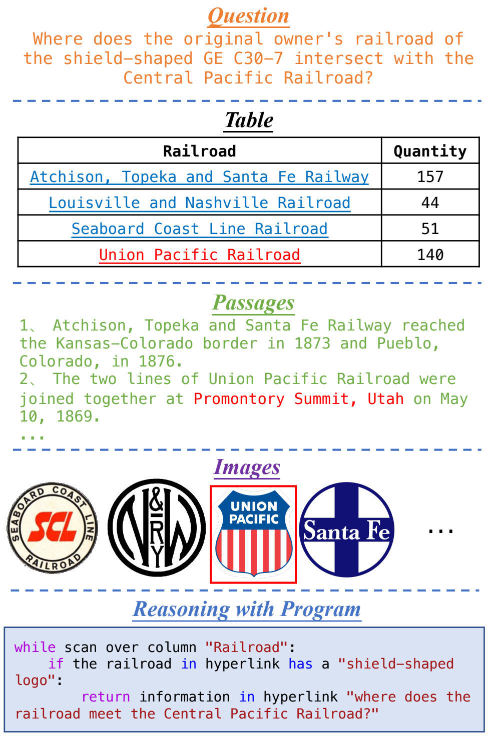

Figure 1: Example of hybrid question answering task with the corresponding program.

To tackle these challenges, recent approaches focus on training domain-specific models to retrieve or rank elements such as table rows, passages, or images, selecting the most relevant ones to

[*] Equal Contributions.
[†] Corresponding author.
[1] Code is available at: https://github.com/qshi95/HProPro

Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 11035–11046
August 11-16, 2024 ©2024 Association for Computational Linguistics

enhance the subsequent reasoning process (Eisenschlos et al., 2021; Kumar et al., 2021; Lei et al., 2023). Since real-world heterogeneous data is vast and constantly updated, even if these approaches demonstrate promising performance on their focused datasets, their applicability to such intricate data is still limited. Furthermore, some existing approaches tend to transform diverse modalities of data into a single modality, such as image captioning (Cheng et al., 2022; Liu et al., 2023), or table-to-text generation (Li et al., 2021), to reduce the task difficulty. However, such approaches are constrained by the performance of modal transformation models, which often result in the loss of information. In a word, these approaches highly rely on data distribution, and the complexity of real-world heterogeneous data makes them exorbitant.

In contrast to previous approaches, we argue that the solution of solving the HQA task should be agnostic to data distribution. Consequently, we advocate for an optimal solution devising a procedure for determining how to find an answer, rather than merely generating the answer itself. Noticing that the program could elucidate the reasoning process employed to arrive at the answer (as depicted in Figure 1), in the current era of large language models (LLMs), leveraging a program can serve as an advantageous solution since LLMs are an excellent program generator. Moreover, the process of program generation necessitates the incorporation of various functions into the program, enabling information-seeking across diverse sources and modalities of data.

Based on the aforementioned considerations, in this paper, we introduce a novel program-based prompting framework HProPro (**H**ybrid **Pro**gram-Based **Pro**mpting) for HQA task. HProPro considers the solution as a process of code generation and execution, integrating external customized functions under the few-shot setting[2]. To facilitate the utilization of customized functions, HProPro incorporates two key components: **Function Declaration** during the code generation phase and **Function Implementation** during the execution phase, which is shown in Figure 2. During the function declaration stage, HProPro defines the function name and formal parameters, utilizing them as prompts to generate code. Subsequently, in the function implementation stage, HProPro implements the declared functions, serving for the direct execution of the generated code. By defining different functions, HProPro can support reasoning over data from various modalities, making it a flexible and scalable framework. Importantly, HProPro eliminates the need to convert different modalities of data into a single modality beforehand. Instead, it acquires information within the origin modal by the functions themselves. To the best of our knowledge, HProPro is the first work to explore the power of LLMs in handling heterogeneous data without requiring domain-specific retrieval or modal transformation. Experiments demonstrate that HProPro significantly outperforms previous methods.

In summary, our contributions are as follows:

- We introduce HProPro, a program-based prompting framework that enables reasoning over heterogeneous data without domain-specific retrieval and modal transformation.
- We implement a few-shot code generation and execution pipeline, calling various functions by function declaration and implementation to perform information-seeking across data from different sources and modalities.
- Experiments show the effectiveness that HProPro achieves state-of-the-art performances under the few-shot settings on both HybridQA (Chen et al., 2020b) and MultiModalQA (Talmor et al., 2020).

2 Method

2.1 HProPro Framework

Task Formulation In this paper, our focus is on the task of hybrid question answering, which involves answering questions based on heterogeneous information sources such as tables, text, and images. The objective is to provide accurate answers to questions based on the given heterogeneous data. Figure 2 provides the comparison between retrieval-based methods and our proposed approach HProPro. Similar to existing program-based prompting approaches, HProPro follows a paradigm that involves generating code and executing it to obtain the final answer. Unlike the previous approaches with a separate retriever, we deal with the input data with external functions but not the retriever module. As a result, we introduce two key components: function declaration

[2]In this work, we use Python code as the carrier of the program.

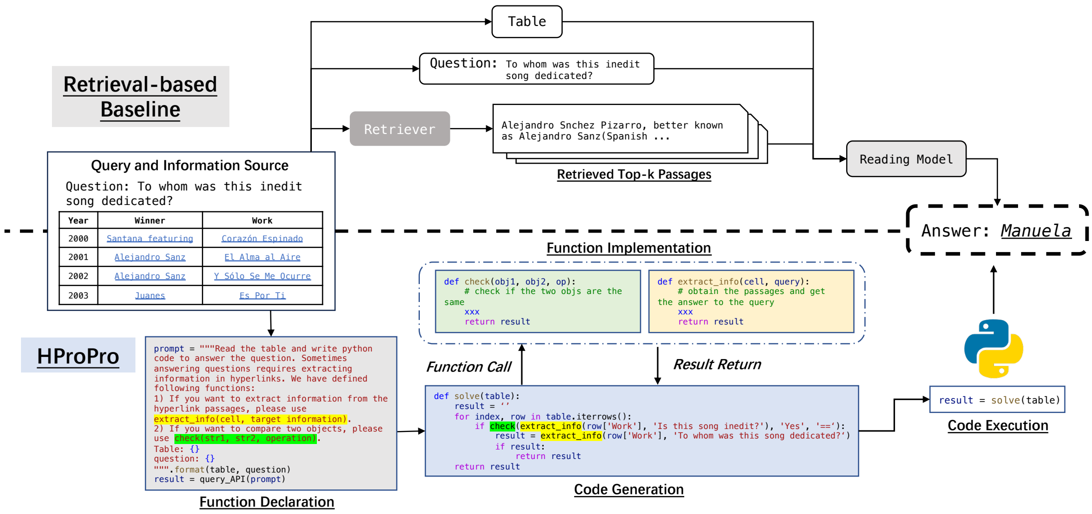

Figure 2: Comparison of HPROPRO with previous retrieval-based methods.

and function implementation, which are required during the code generation stage and code execution stage, respectively. In the following sections, we will delve into both parts of the framework and discuss their roles and functionalities.

Function Declaration The function declaration process in HPROPRO serves the purpose of defining appropriate functions that can be utilized during the code generation phase. During this stage, it is necessary to specify the function name and formal parameters. These declared functions are treated as input prompts for LLMs and are expected to be leveraged to generate code. In Figure 2, the functions with different highlight backgrounds on the left represent the declared functions. Each function has a specific role, which is described briefly alongside the table and query as the input prompts. These prompts are then fed into LLMs to generate the corresponding code. The LLMs will attempt to utilize the declared functions to generate the desired code. By providing function declarations as prompts, HPROPRO enables the LLMs to have a better understanding of the expected structure and behavior of the code to be generated. This allows for more accurate and controllable code generation, ultimately facilitating the HQA task.

Function Implementation The generated code contains formally defined functions, rendering it incapable of direct execution. Consequently, the process of function implementation aims to implement the declared functions to make the code able to be executed by off-the-shelf interpreters. As discussed in Section 1, functions are expected to interact with data from various sources. However, conventional function structures cannot be accommodated in some scenarios, such as extracting information over unstructured texts or images. Therefore, we proceed with the initial implementation of the declared functions integrated with the ability of LLMs to ensure that each function encompasses comprehensive functionality. Specifically, to achieve this process, we pre-design function-related prompts, which are expected to be fed into LLMs when executing the generated code.

2.2 Function Instantiation

In this section, we introduce several functions and elaborate on the declaration and implementation of each function to support HPROPRO.

Extract information from external source To facilitate reasoning across heterogeneous information sources, we introduce the function "`extract_info`". Since data from these sources is often unstructured, the process of extracting information can be likened to a reading comprehension task. In the function declaration, "`extract_info`" is defined as "`extract_info(cell, target_information)`". Here, "`cell`" refers to a specific cell in a table, and "`target_information`" represents the infor-

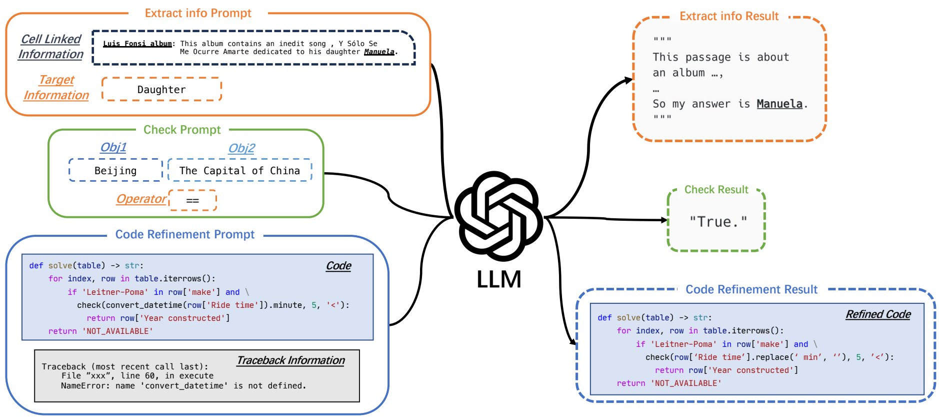

Figure 3: Details of the process of the defined functions and the code refinement.

mation that is required to be extracted, as shown in Figure 3. The function's purpose is to extract the relevant information from the paragraph or image associated with the "cell" based on the specified "target_information". It should return the extracted information as a textual string. All the necessary information, including the function name and its parameters, will be part of the generated code and are expected to be generated by LLMs. During the function implementation process, we utilize an automatically constructed dictionary to locate the corresponding paragraphs or images based on the provided cell. Subsequently, we construct prompts to invoke LLMs based on the data types, which can be categorized into text-based extraction and image-based extraction. The detailed prompts are introduced in Appendix A.

Compare two pieces of information Code often includes rich comparisons between two objects using operators like ">", "<", or "==". However, when dealing with heterogeneous data, the information extracted from various sources may not adhere to a strict format. The form of information obtained from functions like "extract_info" cannot be predetermined. As a result, the traditional comparison operators cannot be directly applied to compare two objects, such as comparing the values ”20, 000” and "ten thousand", or comparing "Beijing" and "the capital of China". To address this issue, we propose a more flexible function called "check". In the function declaration process, "check" can be defined as "check(obj1, obj2, op)". As shown in Figure 3, "obj1" and "obj2" are two strings representing pieces of information. These strings can be the contents of table cells, information obtained from other functions, or directly generated by LLMs based on natural language questions. The "op" parameter represents one of three operators: ">", "<", or "==". The purpose of the "check" function is to compare whether "obj1" and "obj2" are semantically consistent under the specified "op" operator. In other words, it evaluates if the semantic relationship between the two objects aligns with the given operator. Similar to the "extract_info" function, all the relevant information, including the function name and its actual parameters, will be part of the generated code and are expected to be generated by LLMs. During the function implementation process, we provide some few-shot cases as prompts to guide LLMs on how to compare the objects. The detailed prompts are introduced in Appendix A.

2.3 Code Refinement

In HPROPRO, the final answer is obtained by executing the generated code using a standard Python interpreter. Any error in the code will terminate the execution process. However, since the model cannot predict the results returned by each function during code generation, there is a possibility that the model may generate code with mismatched processing methods. This can lead to execution errors or empty results when running the code. Since initial outputs from LLMs can be improved through iterative feedback and refinement (Madaan et al., 2023), we perform code refinement by re-calling the LLMs and incorporating error codes and traceback information into the prompts to generate new code. Figure 3 illustrates the prompts used for code refinement. By providing the above information to LLMs, the models are expected to reconsider the code generation process and generate new code that can alleviate the issues encountered. The detailed prompts are introduced in Appendix A.

2.4 Query Simplification

In the HQA task, code generation is often performed based on the input of a table and a question since including all relevant data as input would result in an extensive input length. However, the reasoning process often involves linked passages or images, which are not directly visible during the code generation phase. This increases the burden of the code generation process. To address this issue, we employ query simplification to simplify the question and establish links between the question and the table cells before conducting code generation. Figure 4 illustrates the schematic diagram of the query simplification process. Taking *"Among animated TV shows, who was the original voice actor on the show whose poster features a character in yellow gloves"* as an example, we utilize a general retriever[3] initially to retrieve relevant information from passages or images in the hyperlinks. Query simplification involves using LLMs that take as input the retrieved passage or image, the original question, and the table. The goal is to replace the span in the question (such as *"the show whose poster features a character in yellow gloves"*) with the corresponding content in the table cell (such as *"Kick Buttowski: Suburban Daredevil"*). The detailed prompts are introduced in Appendix A.

3 Experiments

3.1 Datasets

We conduct experiments on two typical HQA datasets: HybridQA(Chen et al., 2020b) and MultiModalQA(Talmor et al., 2020). Both datasets involve the task of mixed reasoning over diverse sources of data. HybridQA necessitates reasoning over hybrid contexts that consist of both tables and texts. On the other hand, MultiModalQA requires reasoning over tables, texts, and images. To evaluate HPROPRO, we follow the official evaluation metrics provided by the datasets. We report the exact match and F1 score on both HybridQA and MultiModalQA. For more detailed statistics about the datasets, please refer to Appendix B.

[3]The general retriever stands for either a naive retriever or a neural retriever trained on a general corpus, rather than a customized retriever trained on a specific task.

3.2 Experimental Settings

In all our experiments, we utilize different versions of the GPT language models for different components. Specifically, we use `gpt-4-0613` as the backbone model for code generation, code refinement, and query simplification. For implementing the function "`extract_info`", we employ `gpt-4-0613` and `gpt-4-vision-preview` to extract information in the passages and images respectively. For implementing the function "`check`", we employ `gpt-3.5-turbo`. In the process of query simplification, for the HybridQA task, we employ a hybrid retriever that combines TF-IDF and longest-substring matching (Chen et al., 2020b) as the retriever. For the MultiModalQA task, we utilize sentence transformers (Reimers and Gurevych, 2020) as the retriever respectively. The temperature parameter for all models is set to 0. All few-shot experiments for code generation are in the settings of 4 shots. Furthermore, in the oracle settings of MultiModalQA, where the golden passage and image are provided, we remove the query simplification module. This allows us to directly feed all relevant data to the language models without encountering issues related to excessive input length.

3.3 Baseline Systems

We compare HPROPRO to various methods on HybridQA and MultiModalQA, which can be mainly divided into with(*w.*) and without(*w.o.*) fine-tuning approaches. For HybridQA, approaches *w.* fine-tuning stand for the method that trained on the training set, including MAFiD (Lee et al., 2023), S^3HQA (Lei et al., 2023), etc, and approaches *w.o.* fine-tuning include the Unsupervised-QG (Pan et al., 2021) and End-to-End QA with retriever on GPT-4. For MultiModalQA, baseline methods consist of approaches *w.* fine-tuning including SKURG(Yang et al., 2023), PReasM-Large(Yoran et al., 2022), etc, and approaches *w.o.* fine-tuning including Binder(Cheng et al., 2022), MMHQA-ICL (Liu et al., 2023), etc.

3.4 Main Results

Results on HybridQA According to the results presented in Table 1, it is evident that HPROPRO outperforms all baseline systems among ap-

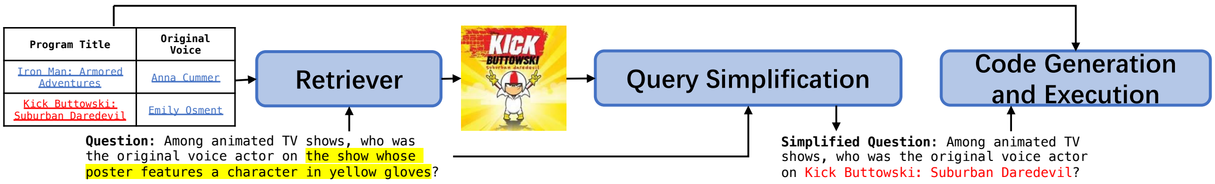

Figure 4: Schematic diagram of query simplification process.

Models	Table				Passage				Total			
	Dev		Test		Dev		Test		Dev		Test	
	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1
Approaches w. Fine-tuning												
HYBRIDER (Chen et al., 2020b)	54.3	61.4	56.2	63.3	39.1	45.7	37.5	44.4	44.0	50.7	43.8	50.6
DocHopper (Sun et al., 2021)	–	–	–	–	–	–	–	–	47.7	55.0	46.3	53.3
MuGER[2] (Wang et al., 2022)	60.9	69.2	58.7	66.6	56.9	68.9	57.1	68.6	57.1	67.3	56.3	66.2
POINTR (Eisenschlos et al., 2021)	68.6	74.2	66.9	72.3	62.8	71.9	62.8	71.9	63.4	71.0	62.8	70.2
DEHG (Feng et al., 2022)	–	–	–	–	–	–	–	–	65.2	**76.3**	63.9	**75.5**
MITQA (Kumar et al., 2021)	68.1	73.3	68.5	74.4	66.7	75.6	64.3	73.3	65.5	72.7	64.3	71.9
MAFiD (Lee et al., 2023)	69.4	75.2	68.5	74.9	66.5	75.5	65.7	75.3	66.2	74.1	65.4	73.6
S[3]HQA (Lei et al., 2023)	**70.3**	**75.3**	**70.6**	**76.3**	**69.9**	**78.2**	**68.7**	**77.8**	**68.4**	75.3	**67.9**	**75.5**
Approaches w.o. Fine-tuning												
Unsupervised-QG (Pan et al., 2021)	–	–	–	–	–	–	–	–	25.7	30.5	–	–
GPT-4 End-to-End QA *w.* Retriever	50.0†	**61.8**†	–	–	11.1†	13.3†	–	–	24.5†	30.0†	–	–
HProPro	**51.4**†	55.9†	**52.9**	**57.6**	**46.8**†	**54.4**†	**46.5**	**57.5**	**48.0**†	**54.6**†	**48.7**	**57.7**

Table 1: Experimental results on HybridQA. † stands for running on 200 sampled cases from the validation set.

proaches *w.o.* fine-tuning. GPT-4 End-to-End QA *w.* Retriever stands for leveraging GPT-4 to generate answers directly along with a retriever. To conduct this experiment, we follow the retrieval approach proposed by Chen et al. (2020b). In comparison to GPT-4 End-to-End QA *w.* Retriever, HProPro achieves more than a 20% improvement in both EM and F1 scores. This result demonstrates the effectiveness of HProPro compared with the approaches relied on retrievers. However, it is important to note that HProPro still exhibits a significant performance gap when compared to the state-of-the-art approaches *w.* fine-tuning. We argue that the main reason for this gap is that these methods are fully trained on the HybridQA dataset. These systems focus on domain-specific training, which includes training a retriever (Wang et al., 2022; Lei et al., 2023), ranker (Kumar et al., 2021), or reasoner (Eisenschlos et al., 2021; Lee et al., 2023). These domain-specific components may lack flexibility and generalization in handling diverse scenarios.

Results on MultiModalQA Table 2 summarizes the results obtained on the MultiModalQA dataset, where HProPro achieves state-of-the-art performances across all experimental settings. When considering systems *w.o.* fine-tuning, HProPro outperforms the previous system MMHQA-ICL by 4.2% and 0.9% in terms of EM and F1 scores, respectively. In comparison to the baseline systems Binder and MMHQA-ICL, which utilize modal transformation modules to convert images into texts, HProPro employs various functions to directly extract information from different modalities. This approach avoids information loss and is more suitable for real-world scenarios involving heterogeneous data. It is important to note that the improvements achieved by HProPro are nontrivial, considering that MMHQA-ICL leverages domain-specific fine-tuned classifiers and retrievers to obtain the type and relevant passages of each question respectively, which heavily relies on the distribution of the targeted benchmark. In contrast, HProPro is performed without any supervised signals from the training set, resulting in a more universal approach.

In the oracle setting which golden passages and images are provided as the input, HProPro achieves comparable results to the previous state-of-the-art system MMHQA-ICL in terms of EM. Demonstrating that regardless of the retriever (only focus on the reasoning part), the results prove that their work highly relies on the retrievers to gain the performances. Besides, compared to their approach, HProPro follows a code generation and execution paradigm, which provides enhanced interpretability and generalizability.

Models	EM	F1
Approaches w. Fine-tuning		
Implicit-Decomp (Talmor et al., 2020)	48.8	55.5
AutoRouting (Talmor et al., 2020)	42.1	49.5
SKURG (Yang et al., 2023)	59.4	63.8
PReasM-Large (Yoran et al., 2022)	59.0	65.5
Approaches w.o. Fine-tuning		
Binder (Cheng et al., 2022)	51.0	57.1
MMHQA-ICL (Liu et al., 2023)	54.8	65.8
HPROPRO	**59.0**	**66.7**
Approaches in Oracle Setting		
Binder$_{oracle}$ (Cheng et al., 2022)	58.1	64.5
MMHQA-ICL$_{oracle}$ (Liu et al., 2023)	65.0	**75.9**
HPROPRO	**65.1**	73.1

Table 2: Experimental results on MultiModalQA.

Models	HybridQA		MultiModalQA	
	EM	F1	EM	F1
HPROPRO	**48.0**	**54.6**	**56.0**	**62.8**
– "check"	44.5	52.5	35.5	38.0
– Question Simplification	43.0	50.0	37.0	39.4
– Code Reflection	43.5	50.9	54.0	60.5

Table 3: Ablation studies on HybridQA and MultiModalQA. All ablation studies are performed on 200 randomly sampled subsets from validation sets.

3.5 Ablation Study

Effect of the function "check" The function "check" is designed to compare the semantic relations between two objects, offering greater flexibility compared to arithmetic operators such as ">", "<", and "==". To demonstrate the effectiveness of the "check" function, we conduct ablation studies by removing its definition in both the function declaration and function implementation processes. Table 3 presents the results of these ablation studies, highlighting the impact of the "check" function. When the "check" function is removed, there is a noticeable drop of 3.5% and 2.1% points in terms of EM and F1 scores, respectively, in the HybridQA dataset. Moreover, the removal of the "check" function has an even more substantial impact on the MultiModalQA dataset. Specifically, the results drop by more than 20% for both EM and F1 scores. This is because constraints from images are weaker than those from passages since LLMs can copy spans from the passages as the answer, which improves the need for the "check" function in this set of experiments.

Effect of query simplification The purpose of query simplification is to alleviate the burden of the code generation process by simplifying the question and establishing links between the question and the table cells. In Table 3, we present the effectiveness of query simplification on both the HybridQA and MultiModalQA datasets. When query simplification is removed, the results demonstrate a decrease of approximately 2% on the HybridQA dataset and a substantial drop of about 20% on the MultiModalQA dataset. These findings highlight the effectiveness of query simplification in the HQA task. It is important to note that the removal of the query simplification module leads to a significant drop specifically in the MultiModalQA dataset. We posit that this drop is due to the presence of passages and images that are necessary to answer the question but are not linked in the table, which couldn't be accessed by the model from the prompt. Therefore, performing query simplification becomes crucial in handling such scenarios in the MultiModalQA task.

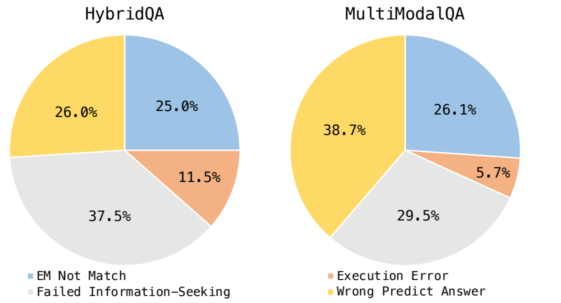

Figure 5: Error percentage of HPROPRO on HybridQA and MultiModalQA.

Effect of code refinement The code refinement module aims to enable LLMs to reconsider the code generation process based on previous execution traceback. In Table 3, we can observe the impact of removing the code refinement module on both the HybridQA and MultiModalQA datasets. When the code refinement module is removed, there is a noticeable decrease in performance. In the HybridQA dataset, the EM and F1 scores drop by 4.5% and 3.7% respectively. Similarly, in the MultiModalQA dataset, the EM and F1 scores drop by 2.0% and 2.3% respectively. The drop in performance demonstrates the effectiveness of the code refinement module in HPROPRO. By enabling LLMs to refine their code generation process based on previous execution errors, the code refinement module plays a vital role in generating accurate code, thereby enhancing the overall ability of HPROPRO in the HQA task.

3.6 Error Analysis

We analyze the errors that occurred within randomly selected subsets of 200 cases from the validation sets of HybridQA and MultiModalQA. Our examination reveals that the main errors can be classified into four distinct types, with the corresponding percentages depicted in Figure 5.

The first type of error involves predicted answers that possess similar meanings to the golden answers but differ in their expressions (25.0% for HybridQA and 26.1% for MultiModalQA). For instance, an instance may present the predicted answer as "the Southeastern Conference (SEC)", while the correct answer is simply "Southeastern Conference". From a technological perspective, we contend that such cases have already been resolved, as the underlying code solution is entirely accurate.

The second type of error observed is related to execution failure (11.5% for HybridQA and 5.7% for MultiModalQA). This error arises due to the inherent complexity of the heterogeneous data, which lacks a standardized format and therefore cannot be effectively addressed using a uniform solution.

The third type of error pertains to failures in the information-seeking from heterogeneous data sources (37.5% for HybridQA and 29.5% for MultiModalQA). These errors occur when the "`extract_info`" function fails to produce a valid result. This may be attributed to a mismatch between the generated code and the expected solution for answering the given question, or it could be indicative of instability in the implementation of the "`extract_info`" function.

The last type of error involves wrong predicted answers (26.0% for HybridQA and 38.7% for MultiModalQA). Due to the similarity between contents in different columns, the model encounters difficulty in discerning the appropriate location to locate the answer when generating code solely based on the provided table. Addressing this challenge remains a topic for future research.

For the detailed visualization results of this analysis, please refer to Appendix C.

4 Related Work

4.1 Hybrid Question Answering

The first line of our related work introduces the HQA task, which focuses on answering questions that require reasoning over diverse information sources. Currently, HQA can be broadly categorized into three subtasks based on the nature of the information sources: table-text question answering (Chen et al., 2020b,a; Zhu et al., 2021), image-text question answering (Reddy et al., 2022; Singh et al., 2021), and table-image-text question answering (Hannan et al., 2020; Talmor et al., 2020). Numerous approaches have been explored for reasoning over heterogeneous data in the context of HQA. Many of these methods primarily focus on supervised fine-tuning over specific benchmarks. This includes training dedicated retrievers (Wang et al., 2022; Kumar et al., 2021; Lei et al., 2023), rankers (Kumar et al., 2021), reasoners (Wang et al., 2022; Kumar et al., 2021; Eisenschlos et al., 2021; Lee et al., 2023; Lei et al., 2023), or transforming different modalities of data into a unified modality (Cheng et al., 2022; Liu et al., 2023; Li et al., 2021). In contrast to existing works, HProPro performs reasoning over heterogeneous data without relying on domain-specific retriever and modal transformation modules. Instead, it integrates various functions to facilitate information-seeking across data from different sources and modalities. Parallel to HProPro, Glenn et al. (2024) propose BlendSQL, which also focuses on HQA task and shares the similar point with HProPro that encodes the full decomposed reasoning roadmap into a single interpretable program-based query.

4.2 Program-based Prompting

The second line of our related work focuses on the program-based prompting strategy, with two closely related works: Program-of-Thought-Prompting (Chen et al., 2022; Gao et al., 2023) and Binder (Cheng et al., 2022). Program-of-Thought-Prompting (Chen et al., 2022; Gao et al., 2023) generates code and executes it using an interpreter. However, their approach is not designed to handle heterogeneous data. In contrast, HProPro integrates function declaration and implementation to specify different functions, enabling effective handling of heterogeneous data. On the other hand, Binder (Cheng et al., 2022) converts images into passages and pre-retrieves relevant passages. These passages, along with the table and question, are then fed into LLMs to generate SQL and Python code for solving the question. In comparison, HProPro does not rely on a modal transformation module or a retriever. Instead, it utilizes various functions to directly interact with data from different sources and modalities.

5 Conclusion

In this work, we propose HPROPRO, a novel program-based prompting framework for HQA tasks, which does not require domain-specific retriever and modal transformation, but integrates various functions to interact with heterogeneous data instead. Experimental results on two typical HQA benchmarks HybridQA and MultiModalQA show the effectiveness of HPROPRO that HPROPRO achieves the best performances under the few-shot settings. For future work, we hope to further utilize the coding capabilities of the LLMs, allowing the model to judge and self-create more customized functions based on different scenarios.

Limitations

The main limitation of this paper is that the performance of HPROPRO relies on the abilities of LLMs, which vary according to the different choices of LLMs. Model updates and server status may affect our experimental results. In addition, the existing benchmarks only focus on heterogeneous data containing tables, passages, and images. More types of data including knowledge graphs and charts are expected to be explored in the future.

Ethics Statement

In this paper, we propose HPROPRO, a program-based prompting framework for the HQA task. We conduct experiments on two benchmarks, namely, HybridQA and MultiModalQA. Both datasets are free and open for research use, which means no ethics issues.

Acknowledgements

We gratefully acknowledge the support of the National Natural Science Foundation of China (NSFC) via grant 62236004, 62206078 and 62441603.

References

Wenhu Chen, Ming-Wei Chang, Eva Schlinger, William Yang Wang, and William W Cohen. 2020a. Open question answering over tables and text. In *International Conference on Learning Representations*.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2022. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*.

Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Yang Wang. 2020b. Hybridqa: A dataset of multi-hop question answering over tabular and textual data. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1026–1036.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, et al. 2022. Binding language models in symbolic languages. In *The Eleventh International Conference on Learning Representations*.

Julian Eisenschlos, Maharshi Gor, Thomas Mueller, and William Cohen. 2021. Mate: Multi-view attention for table transformer efficiency. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7606–7619.

Yue Feng, Zhen Han, Mingming Sun, and Ping Li. 2022. Multi-hop open-domain question answering over structured and unstructured knowledge. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 151–156.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.

Parker Glenn, Parag Pravin Dakle, Liang Wang, and Preethi Raghavan. 2024. Blendsql: A scalable dialect for unifying hybrid question answering in relational algebra.

Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. 2017. Making the v in vqa matter: Elevating the role of image understanding in visual question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6904–6913.

Darryl Hannan, Akshay Jain, and Mohit Bansal. 2020. Manymodalqa: Modality disambiguation and qa over diverse inputs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7879–7886.

Vishwajeet Kumar, Saneem Chemmengath, Yash Gupta, Jaydeep Sen, Samarth Bharadwaj, and Soumen Chakrabarti. 2021. Multi-instance training for question answering across table and linked text. *arXiv preprint arXiv:2112.07337*.

Sung-Min Lee, Eunhwan Park, Daeryong Seo, Donghyeon Jeon, Inho Kang, and Seung-Hoon Na. 2023. Mafid: Moving average equipped fusion-in-decoder for question answering over tabular and textual data. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 2337–2344.

Fangyu Lei, Xiang Li, Yifan Wei, Shizhu He, Yiming Huang, Jun Zhao, and Kang Liu. 2023. S

S^3 hqa: A three-stage approach for multi-hop text-table hybrid question answering. *arXiv preprint arXiv:2305.11725*.

Xiao Li, Yawei Sun, and Gong Cheng. 2021. Tsqa: tabular scenario based question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 13297–13305.

Weihao Liu, Fangyu Lei, Tongxu Luo, Jiahe Lei, Shizhu He, Jun Zhao, and Kang Liu. 2023. Mmhqa-icl: Multimodal in-context learning for hybrid question answering over text, tables and images. *arXiv preprint arXiv:2309.04790*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*.

Liangming Pan, Wenhu Chen, Wenhan Xiong, Min-Yen Kan, and William Yang Wang. 2021. Unsupervised multi-hop question answering by question generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5866–5880.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1470–1480.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392.

Revant Gangi Reddy, Xilin Rui, Manling Li, Xudong Lin, Haoyang Wen, Jaemin Cho, Lifu Huang, Mohit Bansal, Avirup Sil, Shih-Fu Chang, et al. 2022. Mumuqa: Multimedia multi-hop news question answering via cross-media knowledge extraction and grounding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 11200–11208.

Nils Reimers and Iryna Gurevych. 2020. Making monolingual sentence embeddings multilingual using knowledge distillation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Hrituraj Singh, Anshul Nasery, Denil Mehta, Aishwarya Agarwal, Jatin Lamba, and Balaji Vasan Srinivasan. 2021. Mimoqa: Multimodal input multimodal output question answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5317–5332.

Haitian Sun, William W Cohen, and Ruslan Salakhutdinov. 2021. End-to-end multihop retrieval for compositional question answering over long documents. *arXiv preprint arXiv:2106.00200*.

Alon Talmor, Ori Yoran, Amnon Catav, Dan Lahav, Yizhong Wang, Akari Asai, Gabriel Ilharco, Hannaneh Hajishirzi, and Jonathan Berant. 2020. Multimodalqa: complex question answering over text, tables and images. In *International Conference on Learning Representations*.

Yingyao Wang, Junwei Bao, Chaoqun Duan, Youzheng Wu, Xiaodong He, and Tiejun Zhao. 2022. Muger2: Multi-granularity evidence retrieval and reasoning for hybrid question answering. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 6687–6697.

Qian Yang, Qian Chen, Wen Wang, Baotian Hu, and Min Zhang. 2023. Enhancing multi-modal multi-hop question answering via structured knowledge and unified retrieval-generation. In *Proceedings of the 31st ACM International Conference on Multimedia*, pages 5223–5234.

Ori Yoran, Alon Talmor, and Jonathan Berant. 2022. Turning tables: Generating examples from semi-structured tables for endowing language models with reasoning skills. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6016–6031.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. Tat-qa: A question answering benchmark on a hybrid of tabular and textual content in finance. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3277–3287.

A Detail Prompts of of HProPro

The system prompt and detail prompts of the function "`extract_info`", function "`check`", code refinement, and query simplification are shown in Figure 7, Figure 8, Figure 9 and Figure 10.

B Data Statistics for Each Dataset

The statistics of HybridQA and MultiModalQA are presented in Table 4 and Table 5.

C Error Analysis

The error analysis results are presented in Figure 6.

```
SYSTEM_PROMPT = """Read the table and write python
code to answer the question. Sometimes answering
questions requires extracting information in
hyperlinks. We have defined following functions:
1) If you want to extract information from the
hyperlink passages, please use extract_info(cell,
target information).
2) If you want to compare two objects, please use
check(str1, str2, operation).
Table: {}
question: {}
"""
```

Figure 6: Details of the system prompt.

```
EXTRACT_INFO_PROMPT = """Read the following
passages and find the answer from the passage
according to the given query.
Passage: [CELL_CONTENT]
[PASSAGES]
Query: [QUERY]
Let's find the Answer. The answer should be as
short as possible, like a word, a number or a shot
span. If the answer is not available in the
passage, the information should be marked as
NOT_AVAILABLE.
Please return the information in this format:
"So my answer is xxxx."

Your answer:"""
```

Figure 7: Details of the prompt of "extract_info" function.

```
CHECK_PROMPT = """Please verify whether the
semantics of the two strings meet the given
conditions.
For example:
Q: ten > 9
Check: True
Q: 21 Nov, 2030 < 09-31-2021
Check: False
Q: Beijing == The capital of China
Check: True
Q: 2022/10/01 == Oct 1st, 2022
Check: True

Q: [STRING1] [REL] [STRING2]
Check:"""
```

Figure 8: Details of the prompt of "check" function.

```
CODE_REFINEMENT_PROMPT = """There is something
wrong in your code, that I can't run it.
Orginal Prompt: [PROMPT]
Generated Code: [CODE]
Execution Error Information: [TRACEBACK]
Can you generate the code again and fix it?
Your answer:"""
```

Figure 9: Details of the prompt of the code refinement process.

Split	Train	Dev	Test	Total
In-Passage	35,215	2,025	20,45	39,285 (56.4%)
In-Table	26,803	1,349	1,346	29,498 (42.3%)
Computed	664	92	72	828 (1.1%)
Total	62,682	3,466	3,463	69,611

Table 4: Data Statistics for HybridQA.

```
QUERY_SIMPLIFY_PROMPT = """I have a question:
[QUERY]
I want to solve it step by step. Now I get some
information:
[KNOWLEDGE]
If I substitute this information into my question
(only replace the corresponding entity in the
question), what will my question be ?
Attention: You must make sure that you replace the
completely same part, no more or less.
Only give me the question without any explanation
or description."""
```

Figure 10: Details of the prompt of the query simplification process.

Measurement	*Value*
# Distinct Questions	29,918
Train multimodal questions	34.6%
Dev.+test multimodal questions	40.1%
Train compositional questions	58.8%
Dev.+test compositional questions	62.3%
Average question length (words)	18.2
Average # of answers per question	1.16
List answers	7.4%
List answers per intermediate question	18.9%
Average answer length (words)	2.1
# of distinct words in questions	49,649
# of distinct words in answers	20,820
# of distinct context tables	11,022

Table 5: Data statistics for MultiModalQA.

<table>
<tr><th>Type</th><th>Question</th><th>Generated Code</th><th>Result Comparison</th></tr>
<tr><td>EM Not Match</td><td>What conference does the team won in 2016 belong to?</td><td><pre>def solve(table) -> str:
    for index, row in table.iterrows():
        if '2016' in row['Years won']:
            return extract_info(row['Team'],
'What conference does this team belong to?’)</pre></td><td>The predicted answer is "the Southeastern Conference (SEC)", while the golden answer is "Southeastern Conference". The code has the completely correct solution but has different expressions with the golden answer.</td></tr>
<tr><td>Execution Error</td><td>Who was the older person involved in writing the book from 2000?</td><td><pre>def solve(table) -> str:
  older_author, older_age = None, 0
  for index, row in table.iterrows():
    if check(row['Year'], '2000', '==‘):
      for author in row['Author'].split(',’):
        age = int(extract_info(author, 'age’))
          if check(older_age, age, '<'):
            older_age,older_author=age,author
            return older_author</pre></td><td>The golden answer is "Sally Jenkins", while the existence of function "<code>split</code>" and "<code>int</code>" make the code fail to execute since the heterogeneous data is complex.</td></tr>
<tr><td>Failed Information Seeking</td><td>What shipbuilder designed the ship that the delivery voyage was the longest by a patrol boat?</td><td><pre>def solve(table) -> str:
    for index, row in table.iterrows():
        if 'HMAS Fremantle' in row['Name']
and 'patrol boat' in row['Type']:
            return extract_info(row['Name'],
'Who is the shipbuilder?’)</pre></td><td>The golden answer is "Brooke Marine", while the code cannot locate the answer since the "<code>extract_info</code>" returns "None".</td></tr>
<tr><td>Wrong Predicted Answer</td><td>At which air force base did the pilot who graduated in 1968 serve ?</td><td><pre>def solve(table) -> str:
  for index, row in table.iterrows():
    if check(row['ClassYear'],'1968','==‘):
      result = extract_info(row['Notability'],
'At which air force base did the pilot
serve?’)
      return result</pre></td><td>The prediction answer is "Hurlburt Field", while the golden answer is "Hickam". The model cannot distinguish where to find the answer between the columns "ClassYear" and "Notability" since the information in the two columns is similar.</td></tr>
</table>

Table 6: Main error types of HProPro in HybridQA and MultiModalQA.