E^5: Zero-shot Hierarchical Table Analysis using Augmented LLMs via Explain, Extract, Execute, Exhibit and Extrapolate

Zhehao Zhang[1]*, **Yan Gao**[2], **Jian-Guang Lou**[2]
[1]Dartmouth College, [2]Microsoft Research Asia
zhehao.zhang.gr@dartmouth.edu
{yan.gao, jlou}@microsoft.com

Abstract

Analyzing large hierarchical tables with multi-level headers presents challenges due to their complex structure, implicit semantics, and calculation relationships. While recent advancements in large language models (LLMs) have shown promise in flat table analysis, their application to hierarchical tables is constrained by the reliance on manually curated exemplars and the model's token capacity limitations. Addressing these challenges, we introduce a novel code-augmented LLM-based framework, E^5, for zero-shot hierarchical table question answering. This approach encompasses self-explaining the table's hierarchical structures, code generation to extract relevant information and apply operations, external code execution to prevent hallucinations, and leveraging LLMs' reasoning for final answer derivation. Empirical results indicate that our method, based on GPT-4, outperforms state-of-the-art fine-tuning methods with a 44.38 Exact Match improvement. Furthermore, we present F^3, an adaptive algorithm designed for token-limited scenarios, effectively condensing large tables while maintaining useful information. Our experiments prove its efficiency, enabling the processing of large tables even with models having limited context lengths. The code is available at https://github.com/zzh-SJTU/E5-Hierarchical-Table-Analysis.

1 Introduction

In the digital era, structured data, especially in the form of hierarchical tables, plays a crucial role in information representation. Unlike flat tables, hierarchical ones utilize multi-level headers (Lim and Ng, 1999; Chen and Cafarella, 2014; Wang et al., 2021), offering a nuanced organization of complex datasets. They're widely used in areas like finance, scientific research, and business structures. Three main challenges with hierarchical table analysis are: (1) Complex Structure: Their multi-level, bi-dimensional indexing is intuitive for humans but confusing for models. (2) Implicit Calculation Relationships: They often have aggregated rows and columns without obvious indications. (3) Implicit Semantic Relationships: Hierarchical tables contain numerous cross-row, cross-column, and cross-level relationships that aren't clearly indicated (Cheng et al., 2022a).

As large language models (LLMs) (Brown et al., 2020; Chowdhery et al., 2022; Chung et al., 2022; Black et al., 2022; Biderman et al., 2023; Touvron et al., 2023a; OpenAI, 2023a; Touvron et al., 2023b; OpenAI, 2023b; Rozière et al., 2023) have exhibited significant progress in diverse NLP tasks (Qin et al., 2023; Ziems et al., 2024), recent works (Chen, 2023; Ye et al., 2023; Jiang et al., 2023) start to apply LLMs on flat table analysis based on the models' in-context learning ability using few-shot examplers. Ye et al. (2023) introduce the Dater framework, which seeks to decompose table reasoning tasks using Codex (Chen et al., 2021a). Besides, Jiang et al. (2023) presents StructGPT, which utilizes an invoking-linearization-generation approach to apply LLMs on structured data. However, when transitioning to hierarchical table analysis, these methodologies encounter specific constraints: (1) these methods depend heavily on manually curated in-context exemplars, which are extremely **expensive** and inflexible for generalization (2) Given the substantial size of some hierarchical tables, directly inputting them as exemplars for in-context learning can surpass the model's token capacity, making these methods **infeasible**. In light of these challenges, our study emphasizes **zero-shot learning** without the need for exemplars, aiming for a more efficient and universally applicable approach across diverse datasets. To the best of our knowledge, we are the **first** to investigate hierarchical table

*Work done during Zhehao Zhang's internship at Microsoft Research Asia.

Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 1244–1258

analysis using LLMs and study table analysis in a zero-shot manner.

In this work, we propose E^5, a code-augmented framework for hierarchical table question answering (QA) tasks in a zero-shot manner using LLMs. We formulate the process of solving this task as the following steps: (1) self-**Explaining** the hierarchical structures of the complex table to enhance LLMs' comprehension of the underlying semantic associations between headers (2) Generating code to **Extract** related rows and columns and apply appropriate logic operations (*e.g.,* filter, sort, etc) (3) using external code interpreter to **Execute** the generated code and **Exhibit** observations to mitigate hallucinations (4) **Extrapolating** the observations to get the final answer, leveraging the reasoning capabilities of LLMs via natural language. Experiment results on benchmark datasets show that based on GPT-4, our proposal surpasses state-of-the-art (**SOTA**) fine-tuning methods by a significant margin (**44.38** on Exact Match). Besides, compared with another tool-augmented LLM agent, ReAct (Yao et al., 2022), E^{5}achieves noticeable improvement on multiple metrics with a significant increase in **efficiency** and **stability**.

To additionally study the practical token-limited scenario, we propose F^3, an adaptive algorithm to condense large tables while keeping as much useful information as possible within the constraints of the token length. Comprehensive experiments prove that not only does F^{3}save computational resources while maintaining decent performances but also makes it possible to process large tables using models with relatively small context lengths.

2 Related Works

2.1 Tool-Augmented LM

To address complex tasks and mitigate hallucinations, advanced external tools such as web browsers, calculators, search engines, Python interpreters, and diverse modality models have been integrated into LLMs (Nakano et al., 2021; Shuster et al., 2022; Cheng et al., 2022b; Chen et al., 2022; Cobbe et al., 2021; Paranjape et al., 2023; Shen et al., 2023; Lu et al., 2024; Chern et al., 2023; Mialon et al., 2023; Song et al., 2023; Tang et al., 2023). Toolformer (Schick et al., 2023) trains LLMs to use multiple tools in a self-supervised manner. However, it must fine-tune the parameters of the LLM, making it impossible to apply it to close-sourced LLMs like GPT-4. Yao et al. (2022) propose ReAct (Yao et al., 2022), a prompt-based paradigm for integrating reasoning and acting for LLMs, which can be effectively applied to various NLP tasks. However, it heavily relies on meticulously curated exemplars of ($Thought, Act, Obs$) triplets, making the method resource-intensive and constraining its adaptability across diverse tasks.

2.2 LMs for Table Analysis

Numerous studies (Yin et al., 2020; Herzig et al., 2020; Liu et al., 2021; Deng et al., 2022) have pretrained Language Models (LMs) (such as BART (Lewis et al., 2020)) on combined tabular and textual data to learn integrated representation, which can generally obtain the best-known performance on table-associated tasks (Chen, 2023). Chen (2023) first investigate GPT-3's (Brown et al., 2020) in-context leaning ability over tabular data. Zhang et al. (2023) introduce CRT-QA, the first TableQA dataset featuring complex reasoning questions over flat tables, and propose an effective code-augmented method named ARC to solve this task.Ye et al. (2023) propose the Dater framework to decompose table reasoning tasks using Codex (Chen et al., 2021a). Jiang et al. (2023) propose StructGPT, employing an invoking-linearization-generation procedure to apply LLMs on structured data. However, these methods depend heavily on manually created in-context exemplars, which is extremely **expensive** or even **impractical** when the table is extremely large. To the best of our knowledge, we are the **first** to investigate large hierarchical table analysis using LLMs and study table QA in a zero-shot setting.

3 Task Description

The hierarchical table analysis task is a variant of the table-based QA problem, specifically tailored for hierarchical tables. Formally, the task involves processing a hierarchical table T with multi-level headers, a query Q, and a prompt P that contextualizes the query, through a model M. The model outputs an answer A, succinctly represented by the equation: $A = M(T, Q, P)$.

4 Method

As shown in Figure1, we propose E^5, a code-augmented pipeline for question answering over large hierarchical tables. We first let an LLM self-explain table structures to understand hierarchies. Then, the LLM generates code to extract relevant

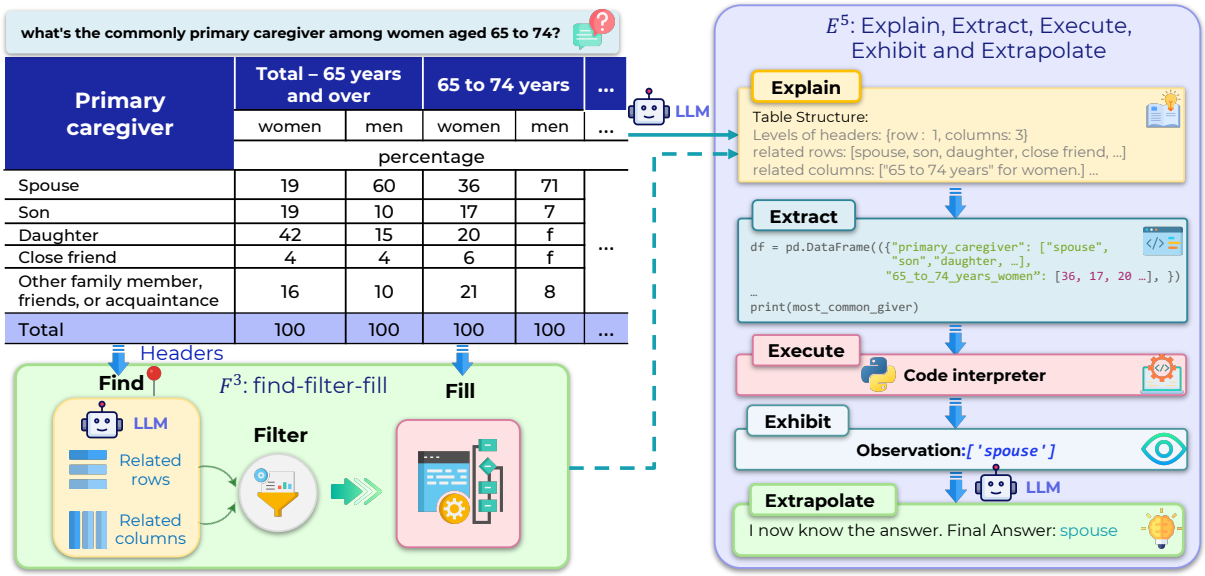


Figure 1: An illustration of our proposed framework of E^{5}and F^3. E^{5}is a pipeline where an LLM explains table hierarchies, generates code to extract relevant data, and then interprets the results post-execution, optimizing question answering. In scenarios with token limitations (dashed line), we first apply F^{3}to compress the large table while maintaining as useful information as possible before feeding to E^5.

table parts and apply operations, making it focus on useful information. An external interpreter then executes the code to avoid hallucination, with results analyzed by the LLM itself. We also propose F^3, an adaptive algorithm to compress large tables in a practical context-limited scenario, which can be applied before E^{5}to compress large tables while keeping as much useful information as possible.

4.1 E^5: Explain, Extract, Execute, Exhibit and Extrapolate

Explain Inspired by CoT (Wei et al., 2022; Kojima et al., 2022) and ReAct (Yao et al., 2022), we make the LLM first explain the structure of the hierarchical table by itself. To be specific, we use the following instructions:

Prompt for Explain

Table Structure: you should describe the table in detail including different levels of headers and their meanings. In the end, you should clearly specify which columns AND rows and their corresponding levels are related to the question.

In this way, the LLM can better understand the table structure, particularly emphasizing the hierarchical header configurations and the latent semantic associations therein. This understanding provides helpful guidance to the subsequent stage of *Extract*. We independently analyze the effectiveness of *Explain* in the main experiment, error analysis (Section 7), and a case study (Section 8).

Extract, Execute and Exhibit Previous works (Shi et al., 2023) find that LLMs are susceptible to distractions from irrelevant information. Meanwhile, the majority of the hierarchical table analysis queries only focus on specific segments of the huge table. In response to this observation, we explicitly ask the LLM to extract relevant cells from the original table based on its internal knowledge and the headers delineated in the *Explain* phase. The *Extract* procedure is integrated into code generation, utilizing `Python Pandas Dataframe` as the container to archive the relevant data. Thereafter, the LLM can devise a range of operations (*e.g.* filter, sort, group) to manipulate the procured `Dataframe`.

During the code generation process, the LLM is explicitly asked to print final or intermediary outcomes(*e.g.,* sorted rows) through instructions. This ensures the acquisition of such displayed output (*Exhibit*) for subsequent scrutiny, post-execution by an external code interpreter (*Execute*). The rationale behind employing an external code interpreter lies in reducing hallucinations in the process of generating the code output by the LLM itself.

Extrapolate During the trial experiments, we observed that LLMs sometimes generate intermediate results instead of the final answer when executing their generated code. Consequently, we introduced an *Extrapolate* phase to further analyze the printed outcomes and then generate the final prediction. This stage makes our framework more flexible and able to be applied to other programming languages like `SQL`, which does not support plenty of logic beyond its database functionalities. Furthermore, the *Extrapolate* stage can combine the reasoning ability of LLMs through both code and natural language in order to perform more complex reasoning tasks. The complete prompt design of E^{5}can be found in Appendix D.

4.2 F^3: Find, Filter and Fill

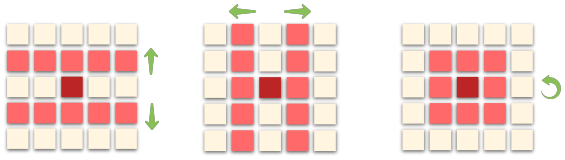
Original E^{5}and all previous table analysis methods (Ye et al., 2023; Jiang et al., 2023) take the whole table as part of the input to LLMs, necessitating a context length that exceeds the table's length. These approaches become infeasible when handling large tables, such as a century-long sales report of a company, or when the available models possess a limited context length. Furthermore, the substantial token count introduced to LLMs can escalate computational costs and augment carbon footprints. Consequently, we introduce F^3, designed to significantly reduce the input token length while trying to retain maximal pertinent information through the subsequent procedures.

Find and Filter In most cases, cells other than headers occupy a significant portion of the table's space and do not contain much semantic information except presenting values. Namely, altering these cells does not necessitate a change in the way to address a specific query in most cases. Consequently, we input only the table headers to the LLM and instruct it to identify the pertinent rows and columns (*Find*). After that, we use a rule-based function to filter the original table and only keep the identified cells.

Fill However, as all our prompt designs are in a zero-shot manner, even GPT-4, the most powerful LLM, can fail to accurately identify the relevant headers, potentially omitting crucial cell values related to the specified query. As a result, we propose an adaptive algorithm to integrate cells from the original table into the filtered version without using LLMs. Our proposed *Fill* algorithm tries to preserve as many cell values as possible, adhering to the context length constraints of the LLM.

To be specific, we iterate all the cells identified in the stage of *Find* and first add all cells from the original table in the same row and column under the assumption that cells in the same row or column may have more similarity. For example, the cells in this column of *women* in Figure 1 table are all statistics about females. After that, we design three different strategies (as shown in Figure 2) to further add potentially useful cells. The row-wise (Figure 2a) and column-wise (Figure 2b) expansion strategy iteratively add neighboring rows and columns respectively. The spiral expansion strategy (Figure 2c) uses a spiral pattern to expand the extracted cells in all directions. Each strategy greedily expands until the maximum token limit is reached or complete filling all the cell values.

Although the *Fill* procedure may bring in unrelated cells that may distract LLMs, this can be mitigated by the *Extract* stage in E^{5}to a certain degree. Besides, we empirically find that the effect of distraction is considerably less than the loss of useful information (described in Figure 3). The complete F^{3}is illustrated in Algorithm 1.



(a) Row-wise (b) Column-wise (c) Spiral

Figure 2: Three different strategies to reintegrate potentially relevant cells into the filtered table. The dark red cell is the one identified in the stage of *Find*. The light red cells are the potentially useful cells that will be added back to the filtered table in the stage of *Fill*. The green arrows indicate the direction of expansion.

5 Experiment Settings

Dataset As we focus on large hierarchical table analysis, we use HiTab (Cheng et al., 2022a) as the primary dataset. HiTab is the only public hierarchical table QA dataset containing complex hierarchical indexing. In contrast to other table QA datasets (Pasupat and Liang, 2015; Chen et al., 2020a,b, 2021b), which predominantly source tables from Wikipedia, HiTab collects tables from Statistics Canada and the National Science Foundation's statistical reports. Not only do these sources offer larger tables, but they also align more closely

Algorithm 1: F^3: Find, Filter and Fill

Input : T: Original hierarchical table, Q: User query, LLM: Language model, N: Token limit, P: Prompt for *Find*

Output : T_{final}: Filtered and filled table with $|T_{\text{final}}| < N$

$T_{\text{header}} \leftarrow \text{ExtractHeaders}(T)$;
$\mathcal{R} \leftarrow \text{LLM}(T_{\text{header}}, Q, P)$ Find
$T_{\text{filtered}} \leftarrow \text{FilterTable}(T, \mathcal{R})$ Filter
$T_{\text{final}} \leftarrow T_{\text{filtered}}$
$i \leftarrow 1$;
while $T_{final} \neq T$ *and* $|T_{final}| < N$ **do**
  $T_{\text{final}} \leftarrow \text{FillTable}(T, \mathcal{R}[i], T_{\text{final}})$ Fill
  $i \leftarrow i + 1$;
return T_{final};

with authentic data analysis scenarios in the industry. We use the test set of HiTab, which has 1584 (table, query) pairs. For table input format, we employ HTML due to its prevalent usage online and its capability to effectively display hierarchical table architectures. To demonstrate the efficacy and adaptability of our proposed approach, we subsequently evaluate E^{5}on an additional two prevalent table QA datasets, WikiTableQuestions (Pasupat and Liang, 2015) and TabFact (Chen et al., 2020a), as detailed in Appendix A.1.

Baselines For comparative analysis, we select the following methods as our baselines[1]:

- **Zero-shot prompting** (Brown et al., 2020): simply prompts LLMs with the given table and query along with instructions.
- **Zero-shot Chain-of-Thought (CoT)** (Kojima et al., 2022): adds *Let's think step by step* in the end of Zero-shot prompting.
- **ReAct** (Yao et al., 2022): employs specific instructions to guide LLMs in producing $(Thought, Act, Obs)$ tuples, facilitating the combination of logical reasoning with actions. Although the original ReAct requires multiple few-shot exemplars, we use the modified zero-shot version from LangChain (Chase, 2023).
- **E^4 (Ours)**: adopts the procedure of *Extract, Execute, Exhibit and Extrapolate* described in Section 4.1.
- **E^5 (Ours)**: adds the step of *Explain* prior to E^4, instructing LLMs to first describe the hierarchical structure and identify relevant headers.

All prompting baselines are built upon GPT-4-32k (OpenAI, 2023b) accessed via the Azure API with the API base of 2023-05-15 for main experiments. We also compare with other LLMs with a smaller context length in Appendix A.2. In addition to LLM-based prompting methods, we also present the results of previous SOTA finetuning-based methods including table pre-trained TaPas (Herzig et al., 2020) and Neural Symbolic Machine (NSM) with MAPO (Liang et al., 2018). Implementation details can be found in Appendix C.

Metrics We use the following two metrics to evaluate hierarchical table QA systems:

- **Exact Match (EM)**: Following previous works (Chen, 2023), we use EM score as the main metric for evaluations. EM is a common metric used in various fields such as information retrieval, machine reading comprehension, and question answering to measure how often the output from a system exactly matches the expected or ground truth value (ranging from 0 to 1). However, EM can not correctly identify semantic equivalence on the prediction and gold label (*e.g.,* "Bhimrao Ramji Ambedkar" and "B. R. Ambedkar.") (Kamalloo et al., 2023).
- **GPT-4-eval**: Inspired by the finding that LLMs have the potential to evaluate NLP models (Chiang and Lee, 2023; Kamalloo et al., 2023), we use GPT-4 as the evaluator to justify the correctness of predictions. We use the following prompt to input GPT-4:

GPT-4-Eval

Please justify whether the model correctly predicted the output based on the given question and label.
Question: Question
Label (Correct Answer): Label
Model Prediction: Prediction

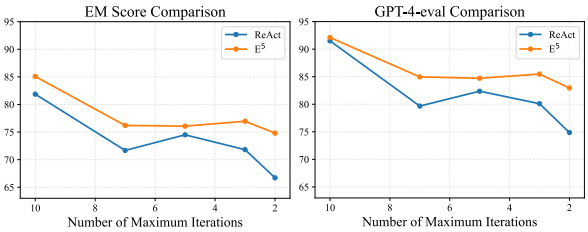
As the finetuning-based methods generate logic forms, we use execution accuracy (EA) to measure the accuracy.

[1]All the prompt design can be found in Appendix D

Token-limited Scenario In order to simulate conditions where the token capacity of the LLM is relatively constrained, we deliberately set a limit on the maximum token length, ranging from 2,000 to 6,000. This range is compatible with numerous LLMs, including LLaMA (Touvron et al., 2023a), Pythia (Biderman et al., 2023) and so on. This maximum token length serves as a parameter in our proposed adaptive algorithm. In this token-limited setting, we first apply $F^3$to get filtered-and-filled tables and subsequently feed them into E^5.

6 Experiment Results

Table 1 presents the primary experimental findings on HiTab. The observations are as follows: (1). Compared with earlier SOTA finetuning-based methods, GPT-4 with instructional prompts can substantially increase the accuracy on HiTab even in a zero-shot manner. This suggests that prompting LLMs has potential superior efficacy in hierarchical table question-answering tasks. (2). Zero-shot CoT brings about a subtle increase in EM score and a noticeable improvement on GPT-4 Eval. (3). Among all the GPT-4-based methods, those augmented with tools consistently surpass naive prompting methods, indicating that incorporating external code interpreter can effectively help LLMs for table analysis by reducing hallucinations. (4). Our proposed E^5achieves the best performances on both metrics. Between our proposals, E^5outperforms E^4 on both metrics by a noticeable margin. This combined with the case study in Figure 8 denotes that the stage of *Explain* is useful to the system by helping the LLM better understand the tables' hierarchical structures and implicit semantic relationships.



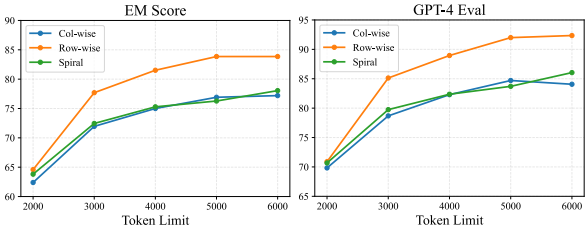
(a) EM Score (b) GPT-4-eval

Figure 3: Performance variation of $E^5$and ReAct in relation to maximum iteration limit. E^5has more stable performances than ReAct on both metrics.

Comparison with ReAct From Table 1, it is evident that E^5surpasses ReAct on both metrics. Furthermore, during the experimentation

Method	EM Score	GPT-4 Eval
Finetuning-based methods		
TaPas	38.90	-
NSM w/ MAPO	40.70	-
Prompting using GPT-4		
Zero-shot	78.56	87.84
Zero-shot CoT	78.74	91.36
ReAct	81.87	91.50
E^4(Ours)	83.19	92.89
E^5(Ours)	**85.08**	**93.11**

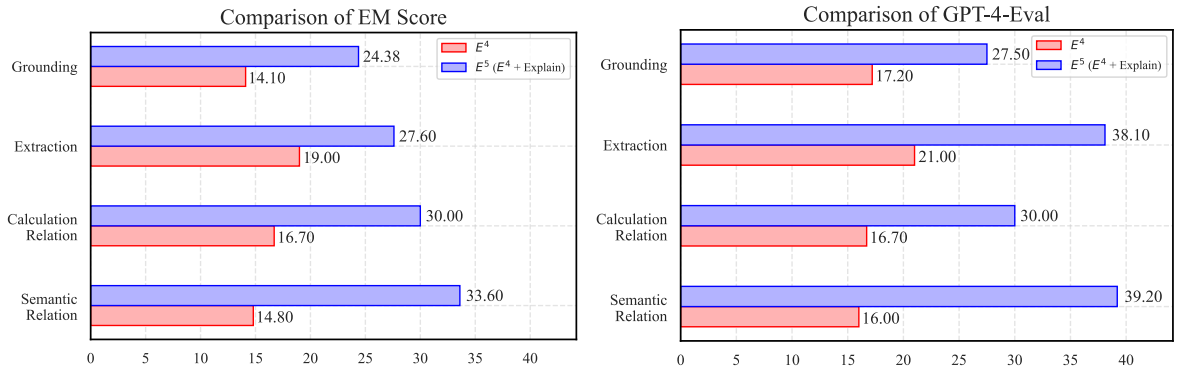
Table 1: The experiment results on HiTab. The mean p-values for the paired t-test between $E^5$and other top-performing baselines is 0.038, indicating significant differences. We report the results averaged over three runs.



(a) EM Score (b) GPT-4-eval

Figure 4: Performances of three *Fill* strategies' with changing token limits. All three strategies achieve decent performances when the input token length is constrained. Among the three strategies, the row-wise diffusion strategy outperforms the others.

phase, we empirically find that ReAct requires a significantly higher number of iterations for the (*Thought*, *Act*, *Obs*) cycle, which consequently leads to an increased time and resource expenditure. To provide a more holistic comparison, we analyze the performance variation of both ReAct and E^5in relation to the maximum iteration limit. As illustrated in Figure 3, when the maximum number of iterations is reduced, E^5exhibits a more marginal performance decline compared to ReAct. The disparity in performance between the two methods becomes more pronounced as the maximum number of iterations is reduced. Specifically, when the maximum number of iterations is limited to 2, this discrepancy is most evident. Under this condition, E^5demonstrates decent performances on both evaluation metrics, achieving scores of **74.81** on EM and **82.95** on GPT-4-eval. Notably, the variance for E^5(9.92) is substantially lower than that of ReAct (29.99), which indicates superior stability



(a) EM Score (b) GPT-4-eval

Figure 5: Comparative analysis of $E^5$and E^4 performance across error categories. We report the averaged results over ten runs. E^5outperforms E^4 in all error categories by considerable margins, indicating the effectiveness of self-explaining tables' hierarchical structures.

of E^5concerning the maximum iteration number. Additionally, in the course of our experiments, we observe that in the absence of instructions to extract pertinent cells using `Pandas`, ReAct frequently attempts to load the complete table via multiple tools, leading to recurrent syntax errors.

Token-limited Scenario Figure 4 illustrates the performance of three *Fill* strategies in relation to the input token length limit. We find that GPT-4 can still achieve decent results even with token limit constraints for hierarchical table analysis, indicating the effectiveness of our proposed F^3. As the token limit increases from 2000 to 6000, all three strategies have considerable performance improvements. notably, the row-wise diffusion strategy outperforms the others across both evaluation metrics.

7 Error Analysis

Besides the main experiment, we also conduct a comprehensive error analysis of our proposals. Specifically, we run E^4 ten times, selecting instances with at least one recorded error. These errors were then classified into the following four categories and visualized in Figure 6:

- **Semantic Relationship Errors:** The LLM struggles to discern implicit semantic relationships between hierarchical headers, as illustrated in Figure 7.
- **Extraction Errors:** The LLM is occasionally unsuccessful in extracting relevant segments of the table to address the query.
- **Calculation Relationship Errors:** In instances where computation is required, the LLM fails to accurately determine the computational relationships between entities.
- **Grounding Errors:** The LLM exhibits challenges in aligning the intent of the query with the corresponding operations on tables.

To further investigate how well E^5can address these error cases through *Explain* phase, we separately evaluate E^4 and E^5on them, repeating the experiment ten times. We visualize the average EM score and GPT-4-eval over ten runs in Figure 5. From this figure, it is evident that E^5exhibits considerable improvement across all four error categories, with a particularly notable enhancement in the semantic relationship category (an increase of over 50%). These findings suggest that LLM can significantly benefit from self-explaining the hierarchical table structure in our tasks.

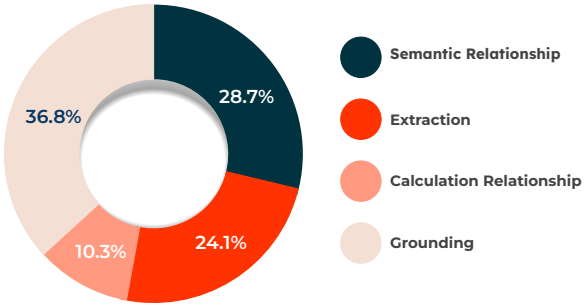


Figure 6: Proportional distribution of four categories of error cases generated by E^4

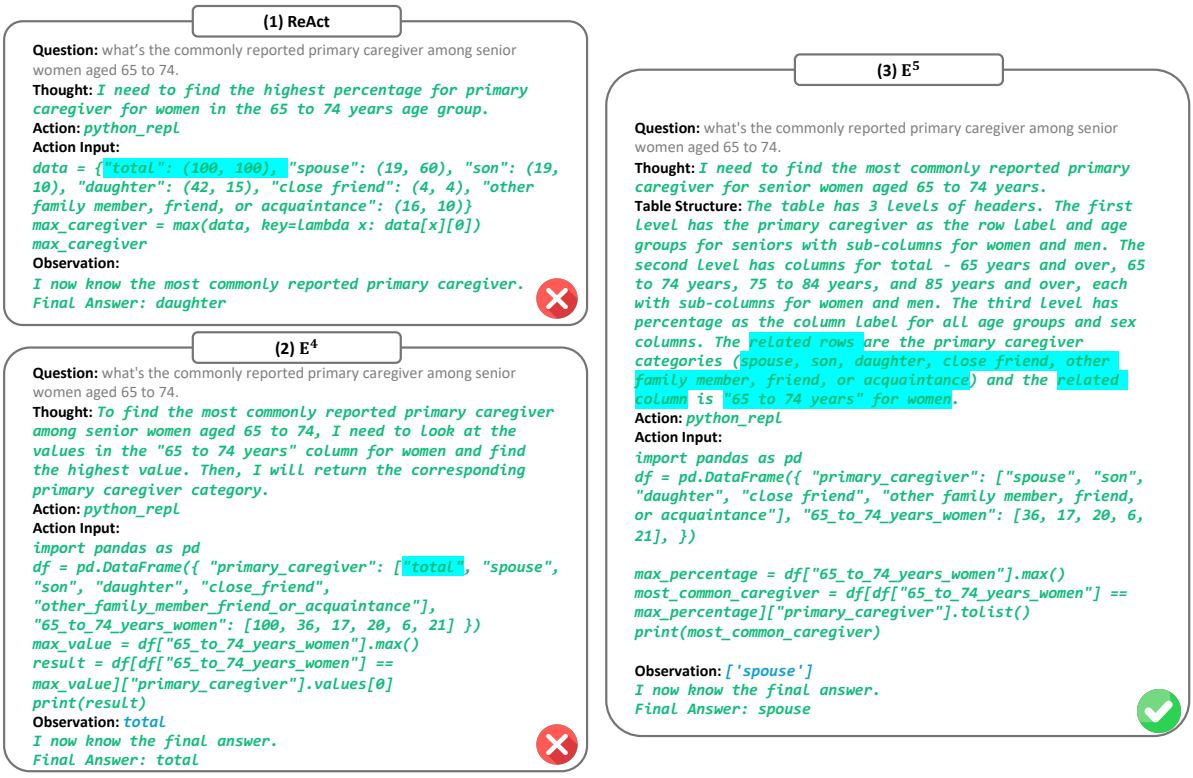


Figure 7: The complete reasoning path and generated code of ReAct, E^4 and E^{5}for the (Table, Query) pair in Figure 1. The highlighted segments are related to how different methods' reasoning over the semantic relationship between the 'total' row and the remaining rows. Compared with ReAct and E^4, E^{5}exhibits an enhanced ability to understand the implicit relationship between headers through self-explaining. In this particular instance, E^{5}can recognize that *Total* is not a category of caregiver and generate the correct answer.

8 Case Study

In order to qualitatively investigate how the stage of self-**Explaining** helps the E^{5}system, we conduct a case study to compare three different tool-augmented approaches, ReAct, E^4, and E^5. Based on the table and query depicted in Figure 1, the reasoning trajectories of ReAct, E^4, and E^{5}are illustrated in Figure 7. From this example, we observe that both ReAct and E^4 simply retrieve all row headers in the given table and neglect the underlying semantic correlations among them, leading to an incorrect answer. Although it is intuitive for humans that the row labeled *Total* represents the cumulative sum of preceding rows, both ReAct and E^4, employing GPT-4, fail to recognize this, mistakenly interpreting *Total* as a separate caregiver category. On the contrary, as highlighted in the paragraph of *Table Structure*, E^{5}accurately identifies the appropriate caregiver categories, excluding the row labeled *Total* in the generated code. This suggests that integrating a phase for self-explaining table structures can enhance the capacity of LLMs to discern implicit relationships within hierarchical headers. Besides, in this case, ReAct does not explicitly print its output, resulting in an absence of observation. Consequently, the LLM produces an inaccurate answer, a phenomenon referred to as "hallucination." Such outcomes suggest that merely instructing LLMs to use tools may not be sufficiently effective for tool-augmented agents. Alternatively, prompting LLMs to appropriately use tools in a structured manner proves to be of paramount importance and the design of such prompts should be meticulously tailored to each distinct task.

9 Conclusion and Future work

We introduce E^5, a code-augmented pipeline for zero-shot question answering over large hierarchical tables. Through comprehensive experiments, we empirically demonstrate that built upon GPT-4, E^{5}can significantly outperform previous SOTA finetuning-based methods by a substantial margin (44.38 on Exact Match) and other prompting strategies with superior stability and adaptability. We separately analyze the phase of self-explaining through extensive error analysis and case study,

proving its effectiveness in helping LLMs get a deeper insight into hierarchical table structures. We also investigate the practical token-limited scenario and propose an adaptive algorithm, F^3, making it possible to analyze large tables using models even with models constrained by a limited context length. We believe that our contributions hold substantial promise for practical applications in real-world table analysis, such as financial statements, in the industry.

Limitations

We identify two limitations in our work that need further investigation and improvement. (1) As our research focuses on leveraging LLMs' hierarchical table analysis ability, we only try three strategies (illustrated in Figure 2) in the *Fill* stage of F^3. Other similarity-based approaches such as clustering or embedding-based search might also be applicable in this context. We leave the exploration of more *Fill* strategies as future works. (2) In this work, as HiTab is the only public hierarchical table QA dataset comprising exclusively single-table analysis data, we do not consider hierarchical table analysis spanning multiple tables.

References

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR.

Sidney Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, Usvsn Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. 2022. GPT-NeoX-20B: An open-source autoregressive language model. In *Proceedings of BigScience Episode #5 – Workshop on Challenges & Perspectives in Creating Large Language Models*, pages 95–136, virtual+Dublin. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Harrison Chase. 2023. Langchain. Accessed: 2023-07-17.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021a. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Wenhu Chen. 2023. Large language models are few(1)-shot table reasoners. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 1120–1130, Dubrovnik, Croatia. Association for Computational Linguistics.

Wenhu Chen, Ming-Wei Chang, Eva Schlinger, William Wang, and William W. Cohen. 2021b. Open question answering over tables and text.

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

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2020a. Tabfact: A large-scale dataset for table-based fact verification.

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, Online. Association for Computational Linguistics.

Zhe Chen and Michael Cafarella. 2014. Integrating spreadsheet data via accurate and low-effort extraction. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, page 1126–1135, New York, NY, USA. Association for Computing Machinery.

Zhoujun Cheng, Haoyu Dong, Zhiruo Wang, Ran Jia, Jiaqi Guo, Yan Gao, Shi Han, Jian-Guang Lou, and Dongmei Zhang. 2022a. HiTab: A hierarchical table dataset for question answering and natural language generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1094–1110, Dublin, Ireland. Association for Computational Linguistics.

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

I Chern, Steffi Chern, Shiqi Chen, Weizhe Yuan, Kehua Feng, Chunting Zhou, Junxian He, Graham Neubig, Pengfei Liu, et al. 2023. Factool: Factuality detection in generative ai–a tool augmented framework for multi-task and multi-domain scenarios. *arXiv preprint arXiv:2307.13528*.

Cheng-Han Chiang and Hung-yi Lee. 2023. Can large language models be an alternative to human evaluations? In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15607–15631, Toronto, Canada. Association for Computational Linguistics.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. Palm: Scaling language modeling with pathways.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling instruction-finetuned language models.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2022. Turl: Table understanding through representation learning. *ACM SIGMOD Record*, 51(1):33–40.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. 2020. TaPas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333, Online. Association for Computational Linguistics.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Wayne Xin Zhao, and Ji-Rong Wen. 2023. Structgpt: A general framework for large language model to reason over structured data. *arXiv preprint arXiv:2305.09645*.

Ehsan Kamalloo, Nouha Dziri, Charles Clarke, and Davood Rafiei. 2023. Evaluating open-domain question answering in the era of large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5591–5606, Toronto, Canada. Association for Computational Linguistics.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Chen Liang, Mohammad Norouzi, Jonathan Berant, Quoc V Le, and Ni Lao. 2018. Memory augmented policy optimization for program synthesis and semantic parsing. *Advances in Neural Information Processing Systems*, 31.

Seung-Jin Lim and Yiu-Kai Ng. 1999. An automated approach for retrieving hierarchical data from html tables. In *Proceedings of the Eighth International Conference on Information and Knowledge Management*, CIKM '99, page 466–474, New York, NY, USA. Association for Computing Machinery.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2021. Tapex: Table pre-training via learning a neural sql executor. In *International Conference on Learning Representations*.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2024. Chameleon: Plug-and-play compositional reasoning with large language models. *Advances in Neural Information Processing Systems*, 36.

Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, Edouard Grave, Yann LeCun, and Thomas Scialom. 2023. Augmented language models: a survey.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

OpenAI. 2023a. Chatgpt.

OpenAI. 2023b. Gpt-4 technical report.

Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. 2023. Art: Automatic multi-step reasoning and tool-use for large language models. *arXiv preprint arXiv:2303.09014*.

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, Beijing, China. Association for Computational Linguistics.

Chengwei Qin, Aston Zhang, Zhuosheng Zhang, Jiaao Chen, Michihiro Yasunaga, and Diyi Yang. 2023. Is ChatGPT a general-purpose natural language processing task solver? In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1339–1384, Singapore. Association for Computational Linguistics.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *arXiv preprint arXiv:2303.17580*.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H. Chi, Nathanael Schärli, and Denny Zhou. 2023. Large language models can be easily distracted by irrelevant context. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 31210–31227. PMLR.

Kurt Shuster, Mojtaba Komeili, Leonard Adolphs, Stephen Roller, Arthur Szlam, and Jason Weston. 2022. Language models that seek for knowledge: Modular search & generation for dialogue and prompt completion. *arXiv preprint arXiv:2203.13224*.

Yifan Song, Weimin Xiong, Dawei Zhu, Cheng Li, Ke Wang, Ye Tian, and Sujian Li. 2023. Restgpt: Connecting large language models with real-world applications via restful apis. *arXiv preprint arXiv:2306.06624*.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. 2021. Tuta: Tree-based transformers for generally structured table pre-training. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, page 1780–1790, New York, NY, USA. Association for Computing Machinery.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023. Large language models are versatile decomposers: Decomposing evidence and questions for table-based reasoning. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 174–184, New York, NY, USA. Association for Computing Machinery.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8413–8426, Online. Association for Computational Linguistics.

Zhehao Zhang, Xitao Li, Yan Gao, and Jian-Guang Lou. 2023. CRT-QA: A dataset of complex reasoning question answering over tabular data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2131–2153, Singapore. Association for Computational Linguistics.

Caleb Ziems, William Held, Omar Shaikh, Jiaao Chen, Zhehao Zhang, and Diyi Yang. 2024. Can Large Language Models Transform Computational Social Science? *Computational Linguistics*, pages 1–55.

A Ablation Studies

A.1 Experiments on Other Datasets

Although we focus on the analysis of hierarchical tables, to further demonstrate the effectiveness and flexibility of our proposal, we directly apply E^{5}on another two commonly-used flat table QA datasets, WikiTableQuestions (Pasupat and Liang, 2015) and TabFact (Chen et al., 2020a) using the exact same prompt design in the main experiments. We follow the exact same experiment setups with Chen (2023) for these two datasets. WikiTableQuestions is characterized by complex questions derived from Wikipedia tables. We evaluate the standard test set, encompassing approximately 4,000 questions. Within this dataset, the evaluation metric employed is the EM score. TabFact, on the other hand, has both simple and complex claims. These claims are annotated by crowd workers using Wikipedia tables as references. The simple subset predominantly lacks higher-order operations like max/min/count and the like. Conversely, the complex subset is saturated with claims that incorporate these higher-order operations. This dataset requires the QA system to determine if the provided claim is true or false based on the table. Our evaluation spans the original test set, which comprises 12,779 examples, and we present the binary classification accuracy for this dataset.

Method	EM Score
SOTA table pre-trained model with finetuning	
TAPAS (Herzig et al., 2020)	48.80
TAPEX (Liu et al., 2021)	57.50
Prompting using GPT-4	
ReAct (Yao et al., 2022)	57.32
E^5(Ours)	**65.54**

Table 2: Experiment results on WikiTableQuestions

Method	Accuracy
SOTA table pre-trained model with finetuning	
TAPAS (Herzig et al., 2020)	81.00
TAPEX (Liu et al., 2021)	84.20
Prompting using GPT-4	
ReAct (Yao et al., 2022)	83.66
E^5(Ours)	**88.77**

Table 3: Experiment results on TabFact

Tables 2 and 3 present the performance metrics of various baselines on WikiTableQuestions and TabFact, respectively. It is evident that our proposed E^{5}consistently surpasses prior SOTA table pre-trained models and ReAct, yielding notable enhancements on both datasets. These results indicate the efficacy of our approach and its superior generalization capability across different datasets.

A.2 Experiments with Models with Smaller Context Length

As we use F^{3}to compress large hierarchical tables within the input token limit, it is possible to analyze these tables using models with limited context length limit. As a result, we experiment with another three models: text-davinci-002 (Brown et al., 2020), text-davinci-003, and GPT-3.5-turbo (OpenAI, 2023a). These three models' context length limits are all 4,096. Following the practice in the main experiment, we first apply F^{3}to get filtered-and-filled tables and subsequently feed them into E^5.

Model	EM Score	GPT-4 Eval
text-davinci-002	26.81	32.13
text-davinci-003	35.23	43.11
GPT-3.5-turbo	36.78	41.39

Table 4: Experiment results of three models with relatively small context length

Table 4 shows the experiment results of these three models on HiTab. Combined with the results in Table 1, we have the following observations: (1) Compared with GPT-4, these three models can not reach comparative performances on both metrics, indicating a huge gap in ability between GPT-4 and the other LLMs. (2) Despite their constrained context length, these models still manifest performance metrics in close proximity to the preceding

state-of-the-art (SOTA) table pre-trained models. This suggests that F^3, facilitating extensive table analysis even with models of limited context length, possesses significant promise for this task.

B Token Length Distribution

To count the tables' number of tokens in HiTab, we tokenize them using the same tokenizer as OpenAI LLMs with `tiktoken` library. We then visualize the token length distribution of HiTab datasets' tables in Figure 8. From this figure, we find that there are a proportion of tables in HiTab that contain more than 3,000 tokens. Besides, we also need to reserve enough token length for prompt instructions and model generation. As a result, it is impossible to input these large tables entirely to some LLMs with a smaller context length. To solve such issues, the stage of *Filter* in F^3can compress 93.28% of tables with more than 3,000 tokens to within 3,000. From Table 8, we can also conclude that as many hierarchical tables already exceed 3000 tokens, adding these tables with additional prompts is very likely to surpass the capacity of most LLMs, including widely used models like vanilla GPT-3.5 and GPT-4, as well as open-source models such as LLaMA, which have a limited context length of several thousand tokens. While our F3 algorithm can compress table contexts effectively, the addition of instructional exemplars, which encompass instructions and codes, is impractical for many scenarios due to these context limitations. This also makes our zero-shot setting more practical and well-motivated.

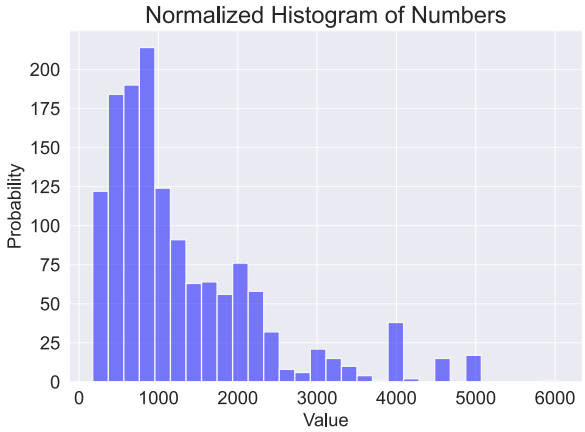


Figure 8: Token length distribution of the HiTab datasets.

C Implementation Details

To balance the randomness and creativity of LLMs' generations, we set the temperature of the LLMs to a small value of 0.3 for experiments. We repeat every experiment three times and report the average performances in the tables. As we find some of the answers in Hitab are incorrectly annotated during the process of error analysis, we filter them out (172 in total) manually before all experiments. We randomly sampled 100 instances for human evaluation, and a difference of less than 1% indicates that GPT-4-eval is effective in our task.

D Propmt Design

The complete prompt designs for all prompting baselines are presented in Table 7. Table 5 and Table 6 shows the prompt design of *Find* in $F^3$and GPT-4-eval respectively.

E Ablation Analysis

Besides the separate analysis of the **Explain** stage in E^5in the main experiment and the study of the effectiveness of **Execute** in previous works (Gao et al., 2023), we also qualitatively analyze the remaining stages in E^5. Empirically, we find that **Extract** is crucial for focusing the LLM on the relevant parts of large tables, avoiding the handling of the entire table in the code, which is prone to syntax errors. For example, the LLM is more likely to generate code to load the entire table's content for the extremely long `Pandas Dataframe` initialization process which greatly increases the change of syntax error in the code. The introduction of **Extract** contrasts with ReAct, which can struggle with larger tables, leading to performance improvement and stability as depicted in Figure 3. The **Exhibit** stage also effectively aids in obtaining observations from the actions performed, as illustrated in Figure 7. In cases where ReAct fails to obtain observations from the code, it can lead to incorrect answers due to hallucination, highlighting the effectiveness of our **Exhibit** stage. For the stage of **Extrapolate**, we empirically observe that as the code frequently outputs some intermediate results (e.g., sorted table, filtered table) that can not be used as the final answer, the final stage can effectively accomplish the final step of reasoning to output the answer in a desirable format.

Prompt Design for the stage of *Find*

Zero-shot	
Instruction:	Given the HTML table's headers and a question, please specify which columns and rows are useful for answering the question. Keep in mind that column indexes or row indexes are the order after the header, starting from 1. For example, if there is a three-column left header and the related column is right next to the header, the index is 1 (do not include the span of the header). Begin!
Title:	Table_title
Table:	Html_table
# of header rows:	top_header_row_num (This is just for reference so you do not count this in the answer's index)
# of header cols:	left_header_columns_num (This is just for reference so you do not count this in the answer's index)
Question:	Question

Table 5: Prompt design of the stage of *Find* in F^3

Prompt Design for GPT-4-eval

Zero-shot	
Instruction:	Please justify whether the model correctly predicted the output based on the given question and label.
Question:	Question
Label (Correct Answer):	Label
Model Prediction:	Prediction If the model makes a correct prediction (Only output Yes or No with nothing else):

Table 6: Prompt design of GPT-4-eval

Zero-shot

```
Table       Answer the question based on the following html of a hierarchical table
Title:      Table_title
Table:      Html_table
Question:   Question
Answer:
```

Zero-shot-CoT

```
Table       Answer the question based on the following html of a hierarchical table
Title:      Table_title
Table:      Html_table
Question:   Question
Explanation: Let's think step-by-step and output the final answer in the end.
```

ReAct

```
Instruction Answer the following questions based on the html table. You have access to the following tools:
            python_repl
            Use the following format:
            Question: the input question you must answer
            Thought: you should always think about what to do
            Action: the action to take, should be one of [python_repl]
            Action Input: the input to the action (python code)
            ... (this Thought/Action/Action Input/Observation can repeat N times)
            Thought: I now know the final answer
            Final Answer: I now know the final answer
            Begin!
Title:      Table_title
Table:      Html_table
Question:   Question
```

E[4]

```
Instruction Answer the following questions based on the html table. You have access to the following tools:
            python_repl
            Use the following format:
            Question: the input question you must answer
            Thought: you should always think about what to do
            Action: the action to take, should be one of [python_repl]
            Action Input: the input to the action (python code), you should first use pandas to first create a dataframe
            with relevant data and write code to accomplish the goal. To accomplish that, you should not load the raw HTML
            data. Instead, you should use df = pd.DataFrame(dict) as the following:
            df = pd.DataFrame( column1 : [],  column2 : [] ) # dict's keys are related column names and values are cell
            values. Note that you should not load the entire table (such as pd.read_html) and only load the related part.
            ...
            print(...) # Finally, you MUST explicitly print the final result. For example: print(df)
            ... (this Thought/Action/Action Input/Observation can repeat N times)
            Thought: I now know the final answer
            Final Answer: I now know the final answer
            Begin!
Title:      Table_title
Table:      Html_table
Question:   Question
```

E[5]

```
Instruction Answer the following questions based on the html table. You have access to the following tools:
            python_repl
            Use the following format:
            Question: the input question you must answer
            Thought: you should always think about what to do
            Table Structure: you should describe the table in detail including different levels of headers and their meanings.
            In the end, you should clearly specify which columns AND rows and their corresponding levels are related to the
            question.
            Action: the action to take, should be one of [python_repl]
            Action Input: the input to the action (python code), you should first use pandas to first create a dataframe
            with relevant data and write code to accomplish the goal. To accomplish that, you should not load the raw HTML
            data. Instead, you should use df = pd.DataFrame(dict) as the following:
            df = pd.DataFrame( column1 : [],  column2 : [] ) # dict's keys are related column names and values are cell
            values. Note that you should not load the entire table (such as pd.read_html) and only load the related part.
            ...
            print(...) # Finally, you MUST explicitly print the final result. For example: print(df)
            ... (this Thought/Action/Action Input/Observation can repeat N times)
            Thought: I now know the final answer
            Final Answer: I now know the final answer
            Begin!
Title:      Table_title
Table:      Html_table
Question:   Question
```

Table 7: Prompt designs of all prompting baselines in the experiment