Relationalizing Tables with Large Language Models: The Promise and Challenges

Zezhou Huang
Columbia University
Email: zh2408@columbia.edu

Eugene Wu
DSI, Columbia University
Email: ewu@cs.columbia.edu

***Abstract*—Tables in the wild are usually not relationalized, making querying them difficult. To relationalize tables, recent works designed seven transformation operators, and deep neural networks were adopted to automatically find the sequence of operators, achieving an accuracy of 57.0%. In comparison, earlier versions of large language models like GPT-3.5 only reached 13.1%. However, these results were obtained using naive prompts. Furthermore, GPT-4 is recently available, which is substantially larger and more performant. This study examines how the selection of models, specifically GPT-3.5 and GPT-4, and various prompting strategies, such as Chain-of-Thought and task decomposition, affect accuracy. The main finding is that GPT-4, combined with Task Decomposition and Chain-of-Thought, attains a remarkable accuracy of 74.6%. Further analysis of errors made by GPT-4 shows that the challenges that about half of the errors are not due to the model's shortcomings, but rather to ambiguities in the benchmarks. When these benchmarks are disambiguated, GPT-4's accuracy improves to 86.9%.**

I. Introduction

In relational databases, standard relational tables have rows representing entities and columns representing attributes. But this standard format doesn't always apply to real-world data. For example, data might include years like "2013", "2014", and "2015" as columns, or cells containing lists that violate the first normal form. To address this, recent works [Li et al.(2023)] propose seven operators to transform these tables into the relational format, as illustrated in Figure 1.

To automatically generate a list of operators to relationalize tables, [Li et al.(2023)] exploits visual patterns, similar to how humans identify row and column patterns, to predict the right operators. Their approach involves Deep Neural Networks (DNN) inspired by computer vision, achieving a 57.0% accuracy rate. In comparison, earlier versions of Large Language Models (LLMs) like GPT-3.5, even with a few-shot in-context learning, only reach an accuracy of 13.1%.

Although Li et al.'s method showed higher accuracy than GPT-3.5, it is arguably difficult to continuously improve and maintain a custom DNN pipeline as compared to designing logic that benefits from the modern LLM ecosystem that is evolving on a daily basis. Two recent LLM advancements could change this comparison. Firstly, the introduction of GPT-4, which has orders of magnitude more parameters and is multimodal, could potentially recognize visual patterns inside tables better. Secondly, recent works have explored different strategies to prompt LLMs, like decomposing a complex task into simple ones [Khot et al.(2022)], [Pourreza and Rafiei(2023)], or using Chain-of-Thought (CoT) [Wei et al.(2022)] to let LLMs first reason about the task. These methods have significantly improved LLM performance but weren't used by Li et al.

This paper revisits table relationalization using the latest LLM architectures (GPT-4), and prompting strategies (decomposition and CoT). Our key finding is that these techniques achieve a remarkable 74.6% accuracy in relationalizing tables, showing great promise. Additionally, an analysis of the 25.4% errors made by GPT-4 revealed that about half were not due to the model's mistakes, but rather because of the ambiguities in the benchmarks. We therefore contribute to the relationalization benchmarks by adding alternative answers. After we disambiguate the benchmarks[1], GPT-4 with prompting strategies achieves the accuracy of 86.9%. We conclude with a proposed architecture of LLM-driven transformation systems.

II. Approach Overview

We define the problem and discuss the prompting strategies.

A. Problem Definition

We study the problem from [Li et al.(2023)]. Given 7 transformation operators illustrated in Figure 1, with parameters:

- ***Transpose*, *Subtitle*:** No parameters.
- ***Pivot*:** `row_frequency` for how many rows to *Pivot*.
- ***Ffill*:** `end_idx` for the ending column index to *Ffill*.
- ***Explode*:** `column_idx` for the column index to *Explode*.
- ***Stack*, *Wide_to_long*:** `start_idx` and `end_idx` for the starting and ending column index of the columns to collapse.

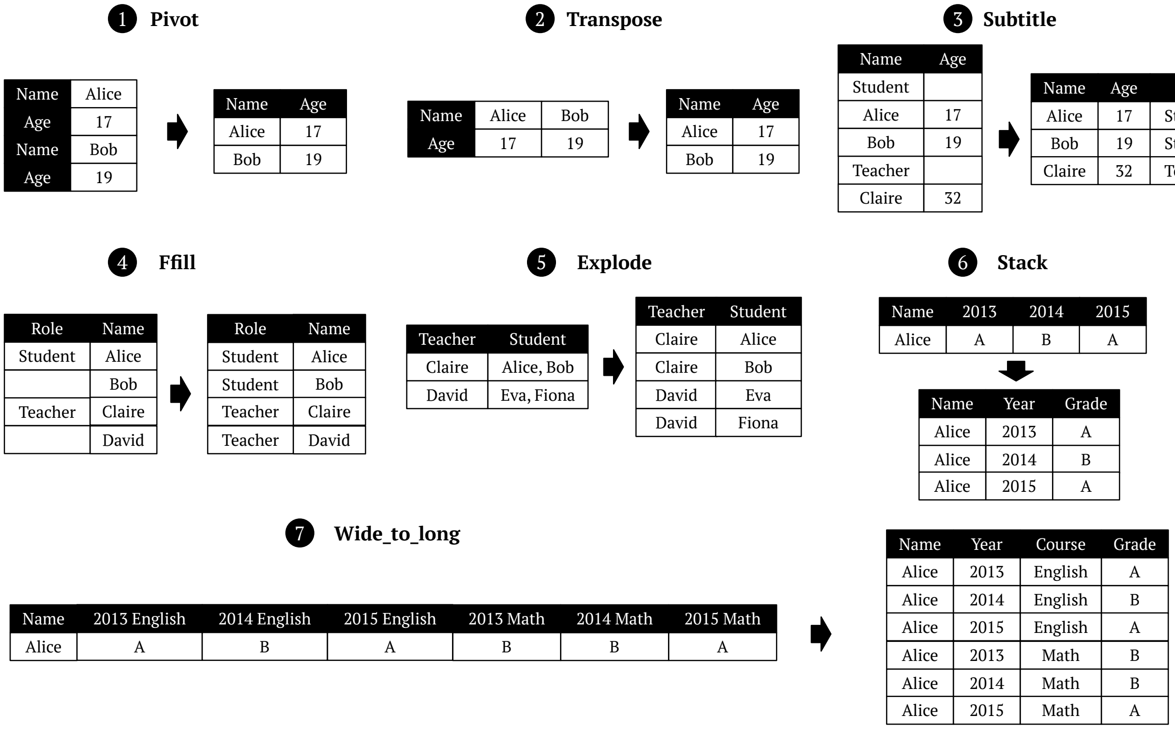
The problem is then defined as:

Problem 1. Given an input table T and the seven operators, the task is to construct a list $M = [\mathrm{o}_1(p_1), \mathrm{o}_2(p_2), \ldots]$, where o_i represents an operator and p_i denotes its parameters for each index i in the range $1 \leq i \leq k$. This list of parameterized operators M, when applied sequentially to T, should transform T into a relational table.

B. Prompting Strategies

To solve Problem 1, we explore different prompting strategies.

[1] The disambiguated benchmarks can be found in https://github.com/zachary62/auto_table_correction/



1 Pivot

Name	Alice
Age	17
Name	Bob
Age	19

Name	Age
Alice	17
Bob	19

2 Transpose

Name	Alice	Bob
Age	17	19

Name	Age
Alice	17
Bob	19

3 Subtitle

Name	Age
Student	
Alice	17
Bob	19
Teacher	
Claire	32

Name	Age	Role
Alice	17	Student
Bob	19	Student
Claire	32	Teacher

4 Ffill

Role	Name
Student	Alice
	Bob
Teacher	Claire
	David

Role	Name
Student	Alice
Student	Bob
Teacher	Claire
Teacher	David

5 Explode

Teacher	Student
Claire	Alice, Bob
David	Eva, Fiona

Teacher	Student
Claire	Alice
Claire	Bob
David	Eva
David	Fiona

6 Stack

Name	2013	2014	2015
Alice	A	B	A

Name	Year	Grade
Alice	2013	A
Alice	2014	B
Alice	2015	A

7 Wide_to_long

Name	2013 English	2014 English	2015 English	2013 Math	2014 Math	2015 Math
Alice	A	B	A	B	B	A

Name	Year	Course	Grade
Alice	2013	English	A
Alice	2014	English	B
Alice	2015	English	A
Alice	2013	Math	B
Alice	2014	Math	B
Alice	2015	Math	A

Fig. 1: Seven types of transformations to relationalize tables. (1) *Pivot*: Turns key-value pairs into a table, with keys as column headers and their corresponding values as rows. (2) *Transpose*: Swaps the rows and columns of a table. (3) *Subtitle*: Move rows of *Subtitle*s as a new column. (4) *Ffill*: Copies the value from the cell above into any empty cell in a column. (5) *Explode*: Splits cells that contain multiple entries into separate rows. (6) *Stack*: Combines homogeneous columns into one, *Stack*ing rows on top of each other. (7) *Wide_to_long*: Similar to *Stack*, but it splits complex homogeneous columns by different dimensions.

1) Single Large Prompt with Few-shot In-context Learning: [Li et al.(2023)] use a single, large prompt and provide one example for each of the seven operators, as demonstrated in the example template in Figure 2. However, this prompt could be challenging for LLMs to execute effectively: (1) The prompt requires the LLM to generate a complete list for all seven transformation operations in one go. This would be complex because the LLM needs to not only comprehend each transformation operator and its parameters but also consider how each step influences the parameters of subsequent transformations, since earlier transformations can alter the column indices for later ones. (2) Additionally, the prompt asks the LLM to produce the output directly without allowing any reasoning process. This complexity could be too overwhelming for LLMs [Shi et al.(2023)], [Liu et al.(2023)].

2) Task Decomposition: To manage the complexity of the task, we exploit the relationships among the transformation operators. We observe that in real-world transformation processes, like those in data wrangling tools [Integrate.io(2023)], [Kandel et al.(2011)] and Kaggle [EHR(2023)], [End(2020)], [Int(2021)], [Name(2019)], people often start with operators that modify the table structure that changes the entire table shape. Following that, modifications are made to columns or rows, and finally individual cells. Taking inspiration from this, we decompose Problem 1 into a list of subproblems[Khot et al.(2022)], [Pourreza and Rafiei(2023)]:

```
Task: Predict transformation operators for table.
=========
Operator descriptions:
- {\it Stack}: collapse homogeneous cols into rows.
@param (int): start_idx:
zero-based starting column index of the column-group.
@param (int): end_idx:
zero-based ending column index of the column-group.
... (the rest 6 operators)
=========
OUTPUT the transformations and parameters in JSON. E.g.,
[{"operator": "{\it Ffill}", "end_idx": 1},
{"operator": "{\it Stack}", "start_idx": 1, "end_idx": 5}].
No explanation is needed.
=========
Here are some example inputs and outputs
## Input
|id|date|items|
|1|2021-01-01|apple, banana, orange|
|2|2022-12-12|banana, carrot|
## Output
[{"operator": "{\it Explode}", "column_idx": 2}]
... (the rest 6 operators)
=========
Your task:
## Input:
(input_table_parsed_string)

## Output:
```

Fig. 2: Single large prompt with few-shot in-context learning, as used by previous work, could be overwhelming.

- **Structural Change of All Rows and Columns (*Pivot*, *Transpose*):** Both *Pivot* and *Transpose* modify the row-column structure of the table to replace all column headers with rows; such correction is the prerequisites to all other operators. Additionally, these structural errors can't arise from other operators. Therefore, we start with two sub-problems to evaluate these two operators.
- **Structual Change of Subset of Rows and Columns (*Subtitle*):** After *Pivot* and *Transpose*, *Subtitle* turns extra non-entity rows to columns, which is the prerequisite to the remaining operators. Therefore, we solve *Subtitle* next.
- **Column change (*Stack*, *Wide_to_long* - Shared Tasks):** Once the extra rows are eliminated, the next priority for the rest operators is to make sure that the columns correctly represent attributes. *Stack* and *Wide_to_long* are similar operators in that both group homogeneous column headers into column values, but they differ in the number of dimensions of homogeneity (*Stack* for one dimension, *Wide_to_long* for multiple). Therefore, we break this down into two tasks: first, identify the homogeneous columns, and then determine the number of dimensions in their homogeneity.
- **Row and Cell Change (*Ffill*, *Explode*):** Following the prior steps, where column headers indicate attributes and rows denote entities, we address issues at the cell level. We use *Ffill* for missing values and *Explode* for lists within cells.

In summary, we break down Problem 1 into simpler sub-problems, as shown in Figure 3, with each sub-problem addressed by a single LLM call. After each sub-problem, we execute Python code to apply the transformation to update the table. This relieves the LLM from having to account for how previous transformations might affect the indices of subsequent columns in multi-step transformations.

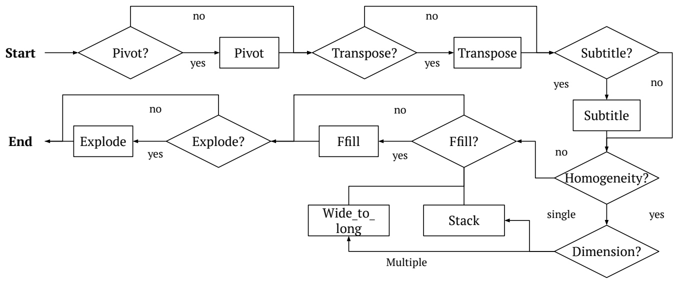


Fig. 3: Decomposed Workflow for Table Relationalization: Each diamond represents a single call to LLM to determine control flow. Each rectangle indicates a Python program that applies the transformation.

3) CoT: Prior work [Wei et al.(2022)] suggests that allowing LLMs to reason before producing an output can enhance their performance. Following this approach, an example of this template for the decomposed task of *Pivot* is shown in Figure 4.

III. Experiments

In this section, we report the experimental findings using the table relationalization benchmark provided by [Li et al.(2023)]. This benchmark comprises 244 tests, with each test including a dataset and one or more lists of parameterized transformation operators as the ground truth. We evaluate various combinations of prompting strategies as follows:

```
Task: Decide if the table needs {\it Pivot}
=========
Example Table:
|Name|Alice|
|Age|17|
|Name|Bob|
|Age|19|

This table rows follow a pattern:
| Attribute Name | Attribute Value |
"Name" and "Age" should be column headers.
Thus, {\it Pivot}ing is needed every 2 row.
=========
Input Table:
(input_table_parsed_string)

1. Identify if the pattern exists in the rows.
2. If yes, count the attributes in each group.
=========
Now, return your answer in JSON:
{
    "reasoning": "The rows mean ...",
    "pattern_exist": true/false,
    "number_of_attributes_per_group": integer
}
```

Fig. 4: Prompt for the decomposed sub-problem of *Pivot* with in-context learning and CoT.

Method	Acc (%)
Auto-Tables from [Li et al.(2023)]	57.0
Large Prompt (GPT-3.5)	13.1
Large Prompt + CoT (GPT-3.5)	8.2
Decomposed Prompts (GPT-3.5)	0
Decomposed Prompts + CoT (GPT-3.5)	3.2
Large Prompt (GPT-4)	46.3
Large Prompt + CoT (GPT-4)	39.3
Decomposed Prompts (GPT-4)	60.1
Decomposed Prompts + CoT (GPT-4)	**74.6**

TABLE I: Accuracy Across Different Models and Methods.

- **Large Prompt:** Via correspondence, we receive the original prompt from the authors of [Li et al.(2023)].
- **Large Prompt + CoT:** This uses the original prompt, but additionally, we prompt the LLM first to provide reasoning.
- **Decomposed Prompts:** This follows our Task Decomposition method in Section II-B2, but we ask the LLM to directly provide the JSON output without the ”reasoning” field.
- **Decomposed Prompts + CoT:** This uses the Task Decomposition method and we instruct the LLM to provide the reasoning as a part of the JSON output.

We use GPT-4 Turbo as the LLM, and accuracy as the performance metric, which corresponds to *Hit@1* in [Li et al.(2023)].

A. Experiment Results

Table I presents the outcomes. The key insights are as follows:

- **Notable Improvement with GPT-4 Over GPT-3.5:** We find that GPT-3.5 frequently mixes up rows and columns,

Category	Count	Percentage (%)
Transpose Ambiguity	3	1.2
Homogeneity Ambiguity	18	7.4
Explode Ambiguity	4	1.6
Explode + Homogeneity Ambiguity	4	1.6
Content Filtering	1	0.4
Mistake	32	13.1
Correct	182	74.6

TABLE II: Breakdown of Answer Categories

which corroborates [Li et al.(2023)]. Interestingly, the use of prompting strategies decreases accuracy for GPT-3.5. These strategies seem to prompt GPT-3.5 to generate longer lists of operators despite the ground truth typically involving no more than two operators. Switching from GPT-3.5 to GPT-4 led to a notable accuracy increase (19.6% → 46.3%). This underscores the importance of larger model sizes in performance. Despite this progress, GPT-4 with Large Prompts still falls short of the specialized Auto-Tables model.

- **Reduced Accuracy with CoT for Large Prompt:** Surprisingly, incorporating CoT with the large prompt actually decreased GPT-4 accuracy (46.3% → 39.3%). Upon reviewing the model's reasoning, we observed that GPT-4 tends to hallucinate when given a large prompt [Liu et al.(2023)]: it generates incorrect assumptions about the input table, leading to wrong outputs.
- **Remarkable Improvement with Task Decomposition:** Breaking down the complex relationalization task into smaller, manageable sub-problems allowed GPT-4 to perform more effectively. The accuracy of 60.1% even surpasses that of the specialized Auto-Tables model (57.0%).
- **Boosted Accuracy Using CoT with Decomposed Prompts:** The combination of decomposed prompts and the CoT technique significantly raised the accuracy (60.1% → 74.6%). Although CoT by itself didn't enhance accuracy, its integration with Task Decomposition allowed GPT-4 to more effectively solve each simplified sub-problem.

B. Error Analysis

While GPT-4, combined with Task Decomposition and CoT, achieves a promising accuracy of 74.6%, what are the nature of the 25.4% errors? Thanks to CoT, the outputs from GPT-4 are interpretable, allowing for detailed manual inspections of these errors.

Key Takeaways: (1) Of the 62 errors by GPT-4, 29 were not due to the model's inaccuracies but rather from ambiguities in the relationalization benchmarks. (2) We contribute to the benchmark by providing alternative answers to these ambiguous queries. With disambiguations, GPT-4, combined with prompting strategies, achieves an accuracy of 86.9%.

We next categorize the ambiguities and provide representative examples for each. The categories are in Table II:

1) Ambiguity in Transpose: Certain tables present a *Pivot* view, leading to ambiguity in *Transpose*. In these cases, both the table header and the first column could be considered as attributes. For instance, `Stack_48` table describes the demographic statistics:

Unnamed: 0	Asian	Black	Hispanic	...
Brown	14%	6%	10%	...
Columbia	15%	8%	13%	...
Cornell	17%	6%	10%	...

The ground truth decided not to *Transpose*, while GPT-4 opted to do so. However, in such scenarios, both choices could be considered valid. The ambiguity lies in determining whether the header or the first column represents the primary set of attributes, making the decision to *Transpose* or not subjective.

2) Ambiguity in Homogeneity: From Task Decomposition, identifying Homogeneous Column Groups is crucial for both *Stack* and *Wide_to_long* operators. Yet, defining 'Homogeneity' is often unclear. Generally, the ground truth views columns with sequential patterns (e.g., "2013", "2014", "2015") or obvious categories (e.g., "red", "blue", "orange") as homogeneous. However, there are instances where the classification of homogeneity is debatable.

- **Less Obvious Categories:** Consider the `Explode_14` table: The columns "Heat", "Drought", "Shade", and "Flood" could be seen as various categories related to crop conditions. GPT-4 classifies them as homogeneous for *Stack*, while the ground truth doesn't.

Species	Heat	Drought	Shade	Flood	...
Annual ryegrass	1	1	3	3	...
Barley	0	3	2	1	...
Oats	1	1	1	2	...
Cereal rye	1	3	3	2	...

- **Small Column Group:** For example, for the `Explode_45` table: The columns "Win" and "Loss" can be considered as different categories of game results. While GPT-4 opts to *Stack* them, the ground truth does not. Generally, the ground truth doesn't *Stack* small groups of only two columns, while GPT-4 does.

Teams	Win	Loss	...
Boston Celtics	17	4	...
Minneapolis/Los Angeles Lakers	16	15	...

3) Ambiguity in Explode: The basic principle of the first normal form in database design involves ensuring that each cell contains a single atomic value. The *Explode* operator is used to map one row to multiple rows, implying a "one-to-many" relationship. However, there are instances where cells with multiple values do not semantically represent a "one-to-many" mapping, and thus, using the '*Explode*' operator might not be appropriate:

- **Single Value with Multiple Components:** For `Explode_32`: Although the "Name" column has multiple

Name	Population	...
Carroll Gardens, Columbia Street, Red Hook	38327	...
Battery Park City, Lower Manhattan	20088	...

values separated by commas, they represent different parts of a single address in a geographical hierarchy. Splitting these into multiple rows using the *Explode* might not make sense semantically. Originally, each row indicates the population of the most specific location, but if *Explode*d, it could be wrongly interpreted as representing populations at various levels of granularity. In this case, the ground truth opts for explosion while GPT-4 does not. A more suitable method might be to introduce a new operator that expands these cell values into multiple columns rather than rows.

- **Single Value with Synonyms:** Consider `Explode_10`: While the ”OCCUPATION” field includes multiple values separated by slashes, they likely represent a single occupation, inferred from the semantic similarity of these occupations, but is ”also known as” various synonyms. This does not necessarily suggest a ”one-to-many” relationship between a person and occupations. In this case, GPT-4 opted not to use the *Explode*, whereas the ground truth did.

OCCUPATION	ALL	PHONE	EMAIL	...
Driver/Bus Driver	21043	13480	6142	...
Broker/Stock/Trader	1461924	885050	536957	...

4) Content Filtering: Finally, one test (`Subtitle_21`) failed due to content filtering. Its input table contains album titles with humanity-threatening themes (e.g., ”Animals Killing People - Human Hunting Season”); the server denies the request as a measure of AI safety.

In summary, as shown in Table II, out of 244 test cases, only 32 (13.1%) are clear errors committed by GPT-4. Approximately half of these errors arise from ambiguities: In 26 cases (10.7%), GPT-4 differs due to ambiguities in either the *Explode* function, Homogeneity, or both. Additionally, 3 errors (1.2%) are attributed solely to ambiguity in the *Transpose* function. Our manual evaluation finds that GPT-4's answers in these ambiguous scenarios are also reasonable. We disambiguate the benchmarks by providing alternative answers and GPT-4's accuracy improves to 86.9%.

IV. Conclusion

We investigated how different models (GPT-3.5 and GPT-4) and prompting strategies (CoT and Task Decomposition) affect the accuracy of table relationalization tasks. Our findings show that GPT-4, combined with Task Decomposition and Chain of Thought, achieves a remarkable accuracy of 74.6%. However, upon examining the errors, we discovered that approximately half of them are due to ambiguities in the benchmark itself, rather than the model's limitations. By resolving these ambiguities, the accuracy of GPT-4 improves to 86.9%.

While LLMs show significant promise in table relationalization, there is a need for more operators like column split, and ambiguities are currently identified manually. In future work, we plan to explore the architecture of LLM-driven transformation systems with self-improving components [Wang et al.(2023)]: LLMs will learn and enhance (1) a library of operator functions, (2) a list of dependencies between operators (e.g., prioritizing structural operators before others), and (3) a human-supervised and document-aided prompt metadata system [Huang et al.(2023)] to help uncover and resolve ambiguities.

References

[End(2020)] 2020. End to end data wrangling & simple random forest. https://www.kaggle.com/code/beezus666/end-to-end-data-wrangling-simple-random-forest

[Int(2021)] 2021. Intermittent Sales Forecasting: 5 TimeSeries Models. https://www.kaggle.com/code/chandrimad31/intermittent-sales-forecasting-5-timeseries-models

[EHR(2023)] 2023. Electronic Health Records (EHRs) Data Exploration. https://www.kaggle.com/code/gpreda/electronic-health-records-ehrs-data-exploration

[Huang et al.(2023)] Zezhou Huang, Pavan Kalyan Damalapati, and Eugene Wu. 2023. Data Ambiguity Strikes Back: How Documentation Improves GPT's Text-to-SQL. In *NeurIPS 2023 Second Table Representation Learning Workshop*.

[Integrate.io(2023)] Integrate.io. 2023. *Data Transformation: Explained*. https://www.tibco.com/reference-center/what-is-data-transformation4

[Kandel et al.(2011)] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the sigchi conference on human factors in computing systems*. 3363–3372.

[Khot et al.(2022)] Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2022. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406* (2022).

[Li et al.(2023)] Peng Li, Yeye He, Cong Yan, Yue Wang, and Surajit Chauduri. 2023. Auto-Tables: Synthesizing Multi-Step Transformations to Relationalize Tables without Using Examples. *arXiv preprint arXiv:2307.14565* (2023).

[Liu et al.(2023)] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172* (2023).

[Name(2019)] Author's Name. 2019. CORD-19: EDA, parse JSON and generate clean CSV. https://www.kaggle.com/code/xhlulu/cord-19-eda-parse-json-and-generate-clean-csv

[Pourreza and Rafiei(2023)] Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *arXiv preprint arXiv:2304.11015* (2023).

[Shi et al.(2023)] 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 *International Conference on Machine Learning*. PMLR, 31210–31227.

[Wang et al.(2023)] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291* (2023).

[Wei et al.(2022)] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems* 35 (2022), 24824–24837.