Automated Data Visualization from Natural Language via Large Language Models: An Exploratory Study

YANG WU[*†], Huazhong University of Science and Technology, China
YAO WAN[*†‡], Huazhong University of Science and Technology, China
HONGYU ZHANG, Chongqing University, China
YULEI SUI, University of New South Wales, Australia
WUCAI WEI[*], Huazhong University of Science and Technology, China
WEI ZHAO[*], Huazhong University of Science and Technology, China
GUANDONG XU, University of Technology Sydney, Australia
HAI JIN[*], Huazhong University of Science and Technology, China

The *Natural Language to Visualization (NL2Vis)* task aims to transform natural-language descriptions into visual representations for a grounded table, enabling users to gain insights from vast amounts of data. Recently, many deep learning-based approaches have been developed for NL2Vis. Despite the considerable efforts made by these approaches, challenges persist in visualizing data sourced from unseen databases or spanning multiple tables. Taking inspiration from the remarkable generation capabilities of *Large Language Models (LLMs)*, this paper conducts an empirical study to evaluate their potential in generating visualizations, and explore the effectiveness of in-context learning prompts for enhancing this task. In particular, we first explore the ways of transforming structured tabular data into sequential text prompts, as to feed them into LLMs and analyze which table content contributes most to the NL2Vis. Our findings suggest that transforming structured tabular data into programs is effective, and it is essential to consider the table schema when formulating prompts. Furthermore, we evaluate two types of LLMs: finetuned models (e.g., T5-Small) and inference-only models (e.g., GPT-3.5), against state-of-the-art methods, using the NL2Vis benchmarks (i.e., nvBench). The experimental results reveal that LLMs outperform baselines, with inference-only models consistently exhibiting performance improvements, at times even surpassing fine-tuned models when provided with certain few-shot demonstrations through in-context learning. Finally, we analyze when the LLMs fail in NL2Vis, and propose to iteratively update the results using strategies such as chain-of-thought, role-playing, and code-interpreter. The experimental results confirm the efficacy of iterative updates and hold great potential for future study.

CCS Concepts: • **Human-centered computing → Empirical studies in visualization**.

[*]Also with National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China.
[†]Both authors contributed equally to this research.
[‡]Yao Wan is the corresponding author.

Authors' addresses: Yang Wu, Huazhong University of Science and Technology, China, wuyang_emily@hust.edu.cn; Yao Wan, Huazhong University of Science and Technology, China, wanyao@hust.edu.cn; Hongyu Zhang, Chongqing University, China, hyzhang@cqu.edu.cn; Yulei Sui, University of New South Wales, Australia, y.sui@unsw.edu.au; Wucai Wei, Huazhong University of Science and Technology, China, m202273789@hust.edu.cn; Wei Zhao, Huazhong University of Science and Technology, China, m202073277@hust.edu.cn; Guandong Xu, University of Technology Sydney, Australia, guandong.xu@uts.edu.au; Hai Jin, Huazhong University of Science and Technology, China, hjin@hust.edu.cn.

ACM 2836-6573/2024/6-ART115
https://doi.org/10.1145/3654992

Additional Key Words and Phrases: Data Visualization, Data Analysis, Natural Language Processing, Code Generation, Large Language Models, Exploratory Study

ACM Reference Format:
Yang Wu, Yao Wan, Hongyu Zhang, Yulei Sui, Wucai Wei, Wei Zhao, Guandong Xu, and Hai Jin. 2024. Automated Data Visualization from Natural Language via Large Language Models: An Exploratory Study. *Proc. ACM Manag. Data* 2, 3 (SIGMOD), Article 115 (June 2024), 28 pages. https://doi.org/10.1145/3654992

1 INTRODUCTION

Data visualizations, typically presented as charts, plots, and histograms, offer an effective means to represent, analyze, and explore data, as well as enable the identification and communication of valuable insights. Despite the availability of numerous tools (e.g., Tableau's Ask Data [41] and Amazon's QuickSight [1]) and domain-specific programming languages (e.g., Vega-Lite [58] and ggplot2 [47]) for data visualization, crafting effective data visualizations remains a complicated effort for a range of users, particularly those with limited or no prior visualization experience. Moreover, there is a pressing demand for visualizing data on smart devices such as tablets and mobile phones without requiring users to acquire data visualization expertise.

To facilitate users in conducting data analytics, there has been an increasing interest in the automated generation of data visualizations from natural-language descriptions, denoted as NL2Vis [26, 27]. Existing approaches to NL2Vis mainly fall into two categories: the rule-based [8, 13, 40] and deep-learning-based [25–28]. DataTone [8], Eviza [40], and Evizeon [13] employed a parser (i.e., the Stanford Core NLP Parser [32]), along with a set of predefined rules, to translate natural-language descriptions into visualization queries. DeepEye [25] introduced a novel approach that enables the generation of visualizations based on keyword queries, akin to search engine functionality. Current methods for deep-learning-based techniques rely mostly on the encoder-decoder paradigm, which, in an end-to-end fashion, encodes the natural-language specification into hidden states and subsequently generates visualization queries.

NL2Vis is similar to the the task of NL2SQL (also referred to as Text2SQL) [15], wherein the objective is to translate natural-language descriptions into *Structured Query Language (SQL)* queries. Generally, the visualization is articulated using the *Visualization Query Language (VQL)*, a language that shares similarities with SQL. Both NL2Vis and NL2SQL are based on input tables with diverse structures and aim to generate queries of various complexities, including selection, comparison, aggregation, and join operations. In comparison to NL2SQL, one distinction is that NL2Vis faces the additional challenge of considering intricate visualization attributes during generation, including the selection of chart types (e.g., bar, pie, line, and scatter). Drawing inspiration from an established NL2SQL dataset (e.g., Spider [53]), Luo et al. [27] proposed the creation of a paired dataset for NL2Vis. Based on this dataset, a benchmark called nvBench [26] is built and a Transformer-based model (named ncNet [28]) is introduced.

LLMs for NL2Vis. Recently, *Large Language Models (LLMs)*, such as GPT-3.5 [10] and LLaMA [45], have demonstrated impressive capabilities for few-shot learning in many *Natural Language Processing (NLP)* tasks, including question answering [12, 44], machine translation [30, 36, 49], and code generation [23, 57]. As LLMs sequentially process the entire large-scale training dataset during the pre-training phase, they face a limitation in directly handling structured data, including tabular data. Alternatively, we can serialize the tabular data, input it into the LLMs, and prompt the LLMs to generate data visualizations, which are in the form of domain-specific query language. To fill this gap, this paper aims to address the following research question: "*Can LLMs be utilized for automating data visualization from natural-language descriptions grounded on a table and how?*".

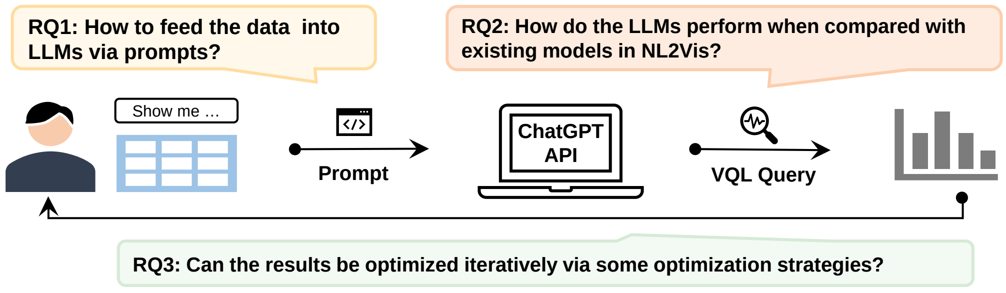

RQ1: How to feed the data into LLMs via prompts?
RQ2: How do the LLMs perform when compared with existing models in NL2Vis?
Show me ...
Prompt
ChatGPT API
VQL Query
RQ3: Can the results be optimized iteratively via some optimization strategies?

Fig. 1. An illustration of NL2Vɪs. The framework presents our investigation into prompt engineering (RQ1), overall performance (RQ2), and iterative updating (RQ3).

The challenges of leveraging the LLMs to automate data visualization from natural language are twofold. **C1: Feeding the structural table into LLMs.** Given that LLMs exclusively accommodate sequential prompts, converting a structured grounded table into sequential prompts while preserving semantics poses a notable challenge. Moreover, LLMs are recognized to face limitations due to their restricted token length. Consequently, they are unable to process entire extensive tables, making it challenging to comprehend comprehensive tabular information on a global scale. **C2: Iteratively updating via conversation.** In contrast to traditional neural models for NL2Vɪs that generate the data visualizations in a single attempt, one notable advantage of LLMs lies in their capacity to iteratively refine predicted outputs during conversations. Adapting traditional neural models to the new conversational paradigm also poses a significant challenge.

Our Work. To answer the aforementioned question, we conduct a pioneering empirical study to evaluate the capabilities of LLMs (i.e., T5 [42] and GPT-3.5 [10]) in automating data visualization from natural-language descriptions, comparing them with traditional approaches. Specifically, we structure the empirical study around the following three *Research Questions (RQs)*, as depicted in Figure 1.

RQ1: How to feed the natural-language query as well as the structural table into LLMs via prompting? In this RQ, we first (1) explore the ways (i.e., table serialization, table summarization, table markup formatting, and table programming) to convert structured tabular data into sequential prompts, and subsequently (2) investigate which table content contributes most to the NL2Vɪs in prompting.

RQ2: How do the LLMs perform when compared with several existing models in NL2Vɪs? In this RQ, we first (1) evaluate the performance of LLMs (i.e., finetuned models such as T5-Small and T5-Base, and inference-only models such as `text-davinci-002` and `text-davinci-003`) for NL2Vɪs, against several traditional neural networks (i.e., Sᴇǫ2Vɪs [27], Transformer [46], ncNet [28], RGVisNet [43]), both under the in-domain and cross-domain settings, and (2) analyze how the number of in-context demonstrations affects the performance of LLMs.

RQ3: Can the results be iteratively updated via some optimization strategies? In some cases, we observe that LLMs may fail to generate the correct visualization in a single attempt. In this RQ, we first (1) analyze when the LLMs fail in generating data visualization, and (2) propose to iteratively update the results via optimization strategies such as *Chain-of-Thought (CoT)*, role-playing, self-repair, and code-interpreter.

Key Findings and Implications. In this paper, we observe the following important findings: (1) To feed the structural table into LLMs, converting it into programs is an effective way. This finding inspires us to design programming patterns to encode tables. Additionally, the schema information is sufficient for NL2Vis task, which facilitates exploration of how to encode extra large databases with limited input length to LLMs. (2) LLMs demonstrate significantly superior performance compared to traditional neural models for NL2Vis, highlighting their great potential under both in-domain and cross-domain settings. With in-context learning of LLMs, the demonstrations drawn from diverse tables can further increase performance. (3) The failure results can be further optimized via several iterative optimization strategies, such as CoT, role-playing, self-repair, and code-interpreter. To advance the visualization capability of LLMs, crafting multi-turn dialog prompts for automated optimization offers a promising prospect.

Contributions. The key contributions of this paper are as follows.

- To the best of our knowledge, this paper reports the first empirical study to investigate the capability of LLMs in automating data visualization from natural-language descriptions. Additionally, a benchmark of LLMs for NL2Vis is built for further study.
- This paper systematically studies how to feed the data to visualize into the LLMs via prompts, and explores several optimization strategies for iteratively improving the failure results.
- To facilitate further study for other researchers, we release all the experimental data and source code used in this paper at https://github.com/CGCL-codes/naturalcc/tree/main/examples/explore-LLMs-for-NL2Vis [48].

Organization. The rest of this paper is structured as follows. We first introduce some background knowledge that will be used in this paper in Sec. 2. We then introduce our pipeline for NL2Vis task in Sec. 3 and the evaluation setup in Sec. 4. Sec. 5 reports the experiment results with comprehensive analysis. Sec. 6 is dedicated to discussing the potential threats to validity and the broader impacts of this paper. We review the related work to this paper in Sec. 7, and conclude this paper in Sec. 8.

2 BACKGROUND

In this section, we present foundational concepts of visualization query languages and LLMs, essential for understanding our work.

2.1 Visualization Query Language (VQL)

In the realm of data visualization, one widely used grammar is Vega-Lite [58], which offers a concise and declarative JSON syntax for creating a diverse range of expressive visualizations suitable for data analysis and presentation. While Vega-Lite is intuitive and straightforward to use, training a sequence-to-sequence model to automatically generate hierarchical outputs, such as JSON format for Vega-Lite specifications, is challenging. Conversely, training a sequence-to-sequence model to generate sequential outputs is relatively more manageable. In response to this challenge, several works [25, 27] introduce the VQL, which empowers users to articulate their data visualization requirements in a structured and efficient manner. In contrast to Vega-lite, VQL queries remove structure-aware symbols such as parentheses, commas, and quotes, effectively transforming a JSON object into a sequence of keywords. This streamlining greatly simplifies the process of generating VQL queries. Moreover, having eliminated all language-specific configurations, VQL is deemed language-agnostic that encompasses necessary data components (e.g., data operations) and vital visualization formats (e.g., visualization types). The VQL query could be easily transformed to diverse specifications (e.g., Vega-Lite, and ggplot2).

Table 1 shows the details of VQL [25] for specifying visualization queries. Specifically, "`VISUALIZE`" specifies the visualization type, including bar, line, scatter, and pie. "`SELECT`" specifies the chosen columns, where X' represents either the original X or its corresponding binned values, and Y'

Table 1. The visualization query language [25].

`VISUALIZE`	$TYPE(\epsilon\{bar, pie, line, scatter\})$
`SELECT`	$X^{'}, Y^{'}\,(X^{'}\epsilon\{X, \mathtt{BIN}(X)\}, Y^{'}\epsilon\{Y, \mathtt{AGG}(Y)\})$
`FROM`	D_1
`JOIN`	$D_1 \bowtie D_2$
`WHERE`	$X'\ OP\ v$
`TRANSFORM`	$X \longrightarrow f(X)$ where $f \in \{\mathtt{BIN}, \mathtt{GROUP}\}$
`ORDER BY`	X', Y'
`AND/OR`	$\phi_1 \land \phi_2\ ,\ \phi_1 \lor \phi_2$
`Nested`	$Q(\text{subquery})$

denotes either the original variable Y or its aggregated value, derived by operations of `SUM`, `AVG`, and `COUNT`. “`FROM`” specifies the originating table. “`JOIN`” operation links tabular data from more than one table. “`WHERE`” filters the values that meet a certain condition, for instance, those greater than 10. “`TRANSFORM`” modifies the chosen columns, typically by binning X into designated buckets or applying a `GROUP` operation. “`BIN`” operation divides the temporal data for visualization into several intervals, e.g., binning by year. “`GROUP`” operation transforms the data into specific groups based on detailed stacked type or classification color. “`ORDER BY`” arranges the selected column in a particular sequence. “`AND/OR`” operation filters data based on multiple conditions. “`Nested`” operations nesting subquery, implements more complex data queries for visual representation.

Example 1. Considering a natural-language query: *List the name of technicians whose team is not “NYY”, and count them by a bar chart, rank x-axis in ascending order*, the corresponding VQL query is as follows.

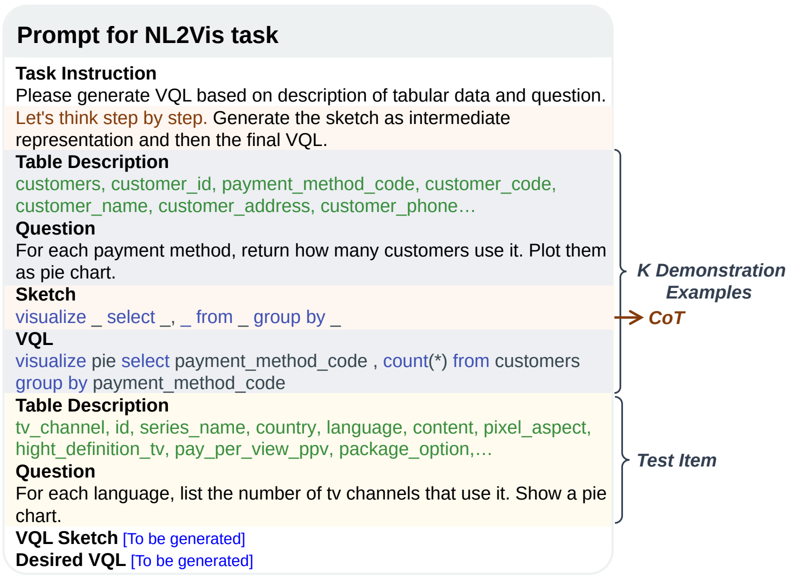
```
VISUALIZE bar SELECT name , COUNT(name) FROM technician WHERE team != “NYY” GROUP BY
name ORDER BY name asc
```

We can see that this example is to select a name with its count from the technician table, where team ! = “NYY”. Then, it groups the results by name with an ascend order, and finally visualizes the results using a bar chart.

2.2 Large Language Models

Over the past year, we have observed a growing proliferation of LLMs, including GPT-3 [4] and LLaMA [45]. These LLMs have spearheaded a revolution in the field of NLP, especially in the related tasks of text generation [30, 49] and code generation [23, 36, 57]. As pre-training LLMs on a large-scale dataset is a time-consuming and computationally expensive process (e.g., ChatGPT requires approximately 4,000 GPU hours for pre-training, at an estimated cost of 2 million dollars [16]), numerous prompting techniques (e.g., in-context learning and chain-of-thought) have been developed with the goal of maximizing the effective utilization of LLMs.

2.2.1 Prompting. With the emergence of LLMs, the learning paradigm is undergoing a transformation from the conventional “*pre-train and fine-tune*” paradigm to a more innovative “*pre-train, prompt, and predict*” framework [24]. In this new paradigm, instead of extensively fine-tuning LLMs to accommodate various downstream tasks, there is a shift towards reformulating these tasks to align more closely with the tasks for which LLMs are initially trained, with textual prompts guiding the process. For instance, when assessing the emotion in a customer review like “*I like the book I have read today.*”, we may proceed with a prompt like “*It was__.*” and request that LLM


K Demonstration Examples
CoT
Test Item

Prompt for NL2Vis task

Task Instruction
Please generate VQL based on description of tabular data and question.
Let's think step by step. Generate the sketch as intermediate representation and then the final VQL.

Table Description
customers, customer_id, payment_method_code, customer_code, customer_name, customer_address, customer_phone…

Question
For each payment method, return how many customers use it. Plot them as pie chart.

Sketch
visualize _ select _, _ from _ group by _

VQL
visualize pie select payment_method_code , count(*) from customers group by payment_method_code

Table Description
tv_channel, id, series_name, country, language, content, pixel_aspect, hight_definition_tv, pay_per_view_ppv, package_option,…

Question
For each language, list the number of tv channels that use it. Show a pie chart.

VQL Sketch [To be generated]
Desired VQL [To be generated]

Fig. 2. An example to illustrate the usage of prompt in in-context learning of LLMs, the k demonstration examples, the test item, and the CoT prompt.

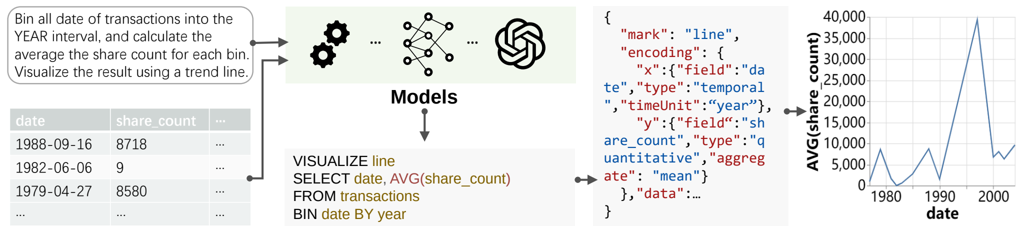
complete the sentence with an emotion-laden word. In our specific scenario, involving a table and a natural-language query, we can construct a prompt as follows: "`[TABLE]`, `[QUESTION]`, *please generate VQL based on the description of tabular data and question.*", where `[TABLE]` signifies the structured tabular data, and `[QUESTION]` represents the natural-language query provided by the end-user for exploring the data.

2.2.2 In-Context Learning (ICL). ICL is a special form of prompt-based learning that leverages demonstration examples in prompts to promote the model's performance. In ICL, in addition to describing the current question in the prompt, a few demonstration examples that have the same form as the question are also included in the prompt. For instance, to generate a VQL query to count the number of television channels based on the language of each individual channel, a demonstrative example of counting customers by their preferred payment method is provided. The model is expected to learn the pattern hidden in the demonstration, infer downstream tasks from examples, and accordingly make the right prediction.

Specifically, given a task instruction I and a test question x_t, ICL retrieves k examples related to x_t from the task dataset as demonstration examples, and transforms these examples using the prompt function f to form a demonstration example set $D_k = \{f(x_1, y_1), \ldots, f(x_k, y_k)\}$. The task description I, the example set D_k, and the problem is then fed into the language model for predicting $\hat{y}_t$. The ICL process can be expressed as follows.

$$\hat{y}_t = \mathrm{LLM}(I, \underbrace{f(x_1, y_1), \ldots, f(x_k, y_k)}_{k \text{ demonstration examples}}, \underbrace{f(x_t, \bullet)}_{\text{test query}}), \quad (1)$$

where k stands for the number of demonstration examples in the ICL prompt, which typically ranges from 0 to 20. In particular, for the special case of $k = 0$, the ICL constructs the prompt without demonstration example, referred to as zero-shot learning.


Bin all date of transactions into the YEAR interval, and calculate the average the share count for each bin. Visualize the result using a trend line.

date	share_count	...
1988-09-16	8718	...
1982-06-06	9	...
1979-04-27	8580	...
...	...	...

```
VISUALIZE line
SELECT date, AVG(share_count)
FROM transactions
BIN date BY year
```

```
{
  "mark": "line",
  "encoding": {
    "x":{"field":"da
te","type":"temporal
","timeUnit":“year”},
    "y":{"field“:"sh
are_count","type":"q
uantitative","aggreg
ate": "mean"}
  },"data":…
}
```

(a) Feeding Data into LLMs via Prompting (b) Visualization Query Generation (c) Language-Aware Rending

Fig. 3. The pipeline of NL2Vis. The task process flows from (a) feeding data into the LLMs through prompts, to (b) generating visualization queries, and finally to (c) rendering language-aware visual charts.

2.2.3 Chain-of-Thought (CoT) Prompting. CoT [52] is an improved prompting strategy that, when employed in conjunction with ICL, significantly enhances the capabilities of LLMs in tackling complex reasoning tasks. In addition to simply constructing the prompts with a few demonstration examples as in ICL, CoT enriches these prompts by integrating intermediate reasoning steps, which guide the reasoning process to the final output. Specifically, the CoT prompting strategy augments each demonstration example $\langle x, y\rangle$ in ICL with a chain-of-thought prompt CoT, constructing a triplet prompt $\langle x, CoT, y\rangle$. The design of the chain-of-thought prompt CoT involves both hand-crafted prompts that are independent from the problem, and the VQL sketch of each problem, which is inspired by the logical execution process of SQL queries to establish step-by-step infilling the VQL statements with the LLM.

Example 2. Figure 2 presents an example of in-context learning with chain-of-thought strategy in NL2Vis. LLMs like ChatGPT, take the input text and infer the answer based on the task description, demonstration, and the problem. In the few-shot scenario, we add the most relative samples from the training dataset as examples in the demonstration. The demonstration part of the prompt for the NL2Vis task consists of a table description, NL question, and golden VQL. In particular, we select the most relevant three rows of the table by calculating the Jaccard similarity correlation.

3 LLMS FOR NL2VIS

In this section, we first formulate the problem of NL2Vis with a pipeline provided, and subsequently detail each individual module.

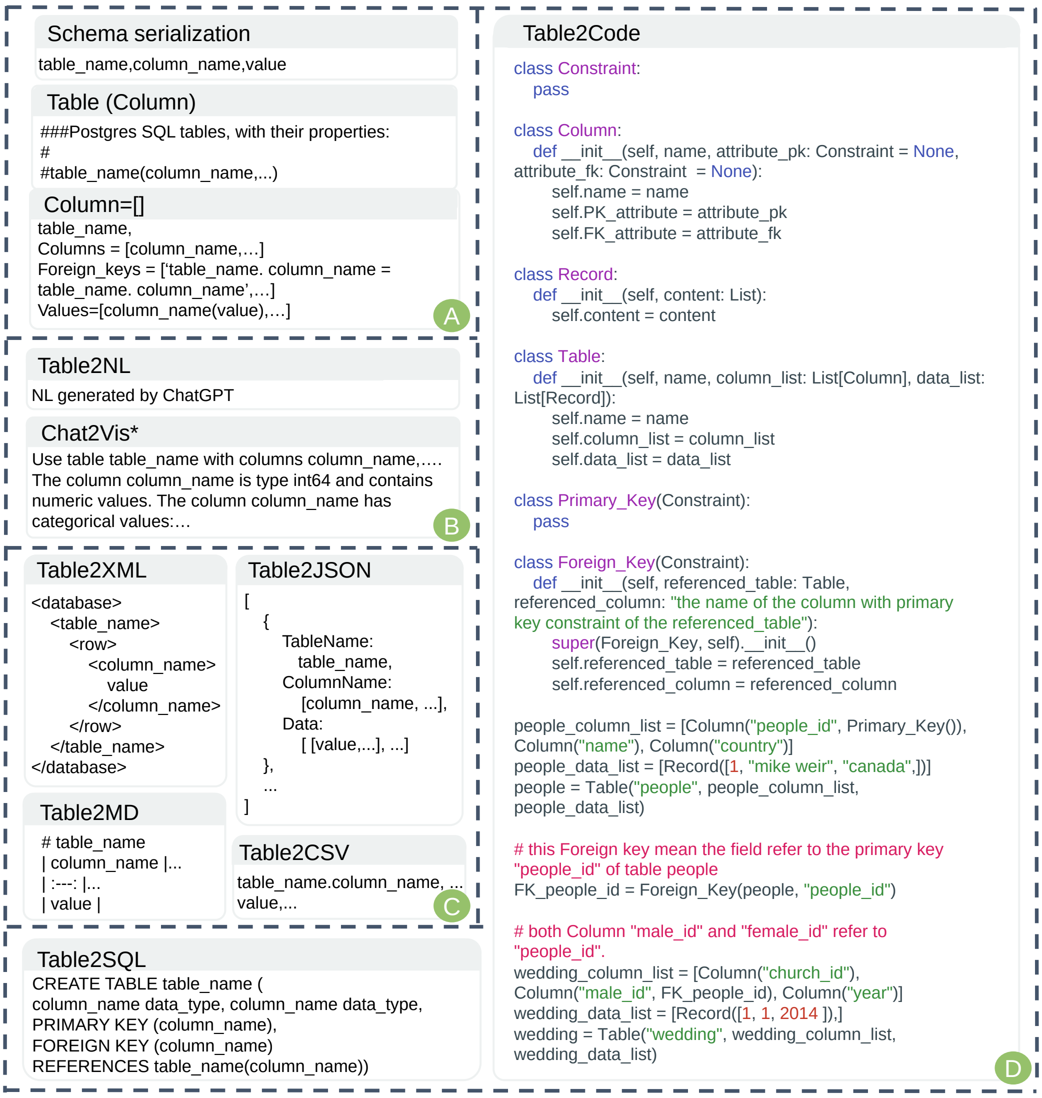
3.1 Problem Statement

Let q denote a natural-language query, s denote the schema of the table from a database to analyze. The task of NL2Vis is to generate the visualization query $\hat{y}$, as follows:

$$\hat{y} = \mathrm{LLM}(q, s)\,. \tag{2}$$

It should be noted that the grounded tables are not restricted to one domain. We refer to scenarios where the databases in the test dataset have appeared in the training set as *in-domain*, while databases in the test dataset are unseen as *cross-domain*.

Figure 3 illustrates the pipeline of NL2Vis. The input to the models comprises a natural-language description and the grounded table. The model will then generate the visualization query in Vega-Zero. Subsequently, visualization specifications in various visual languages (e.g., Vega-Lite) can be parsed from the visualization query. Subsequently, the visualization specification will be rendered into an actual visualization chart, enabling users to observe and analyze the data effectively.



Schema serialization
```
table_name,column_name,value
```

Table (Column)
```
###Postgres SQL tables, with their properties:
#
#table_name(column_name,...)
```

Column=[]
```
table_name,
Columns = [column_name,…]
Foreign_keys = ['table_name. column_name =
table_name. column_name',…]
Values=[column_name(value),…]
```

A

Table2NL
```
NL generated by ChatGPT
```

Chat2Vis*
```
Use table table_name with columns column_name,….
The column column_name is type int64 and contains
numeric values. The column column_name has
categorical values:…
```

B

Table2XML
```
<database>
    <table_name>
        <row>
            <column_name>
                value
            </column_name>
        </row>
    </table_name>
</database>
```

Table2JSON
```
[
    {
        TableName:
            table_name,
        ColumnName:
            [column_name, ...],
        Data:
            [ [value,...], ...]
    },
    ...
]
```

Table2MD
```
# table_name
| column_name |...
| :---: |...
| value |
```

Table2CSV
```
table_name.column_name, ...
value,...
```

C

Table2SQL
```
CREATE TABLE table_name (
column_name data_type, column_name data_type,
PRIMARY KEY (column_name),
FOREIGN KEY (column_name)
REFERENCES table_name(column_name))
```

Table2Code
```
class Constraint:
    pass

class Column:
    def __init__(self, name, attribute_pk: Constraint = None,
attribute_fk: Constraint  = None):
        self.name = name
        self.PK_attribute = attribute_pk
        self.FK_attribute = attribute_fk

class Record:
    def __init__(self, content: List):
        self.content = content

class Table:
    def __init__(self, name, column_list: List[Column], data_list:
List[Record]):
        self.name = name
        self.column_list = column_list
        self.data_list = data_list

class Primary_Key(Constraint):
    pass

class Foreign_Key(Constraint):
    def __init__(self, referenced_table: Table,
referenced_column: "the name of the column with primary
key constraint of the referenced_table"):
        super(Foreign_Key, self).__init__()
        self.referenced_table = referenced_table
        self.referenced_column = referenced_column

people_column_list = [Column("people_id", Primary_Key()),
Column("name"), Column("country")]
people_data_list = [Record([1, "mike weir", "canada",])]
people = Table("people", people_column_list,
people_data_list)

# this Foreign key mean the field refer to the primary key
"people_id" of table people
FK_people_id = Foreign_Key(people, "people_id")

# both Column "male_id" and "female_id" refer to
"people_id".
wedding_column_list = [Column("church_id"),
Column("male_id", FK_people_id), Column("year")]
wedding_data_list = [Record([1, 1, 2014 ]),]
wedding = Table("wedding", wedding_column_list,
wedding_data_list)
```

D

Fig. 4. A summary of approaches explored to transforming the table into textual prompts. (A) Table Serialization flattens database tables into linear schemas, (B) Table Summarization distills table content into concise descriptions, (C) Table Markup Formatting converts data into XML, Markdown, and CSV formats, and (D) Table Programming translates table structures into code representations.

3.2 Feeding Data into LLMs via Prompting

Given that most contemporary LLMs are primarily designed to process textual prompts due to their pre-training on sequential textual datasets, it becomes imperative to efficiently integrate structured tabular data into LLMs through effective prompting. In this study, we investigate various strategies, i.e., table serialization, table summarization, table markup format, and table programming, to transform structured tabular data into sequential texts while preserving semantics to the greatest extent, as summarized in Figure 4.

A. Table Serialization (A in Figure 4). Generally, previous research has proposed to feed the serialized schema into models [9]. For example, **Table (Column)** [22] lists each table along with its columns inside parentheses to represent the table schemas. **Column=[]** represents each table along with a list of its columns. On top of **Column=[]**, **+FK** further adds foreign keys to indicate the relationships between tables [37], and **+Value** adds rows of tables.

B. Table Summarization (B in Figure 4). It is intuitive to simply describe the tables by generating a natural-language summary. Based on this intuition, Chat2Vis [31] uses a description built from a template to transform the table into a text prompt. The description is comprised of individual entries, each signifying the data type of a corresponding column. In this paper, we generate the table summaries by invoking the API of ChatGPT. Specifically, we accomplish this by flatting the table column names and values into a prompt, delineated as "`[TABLE]`, *Describe the tabular data in text, including all metadata such as its name and type.*".

C. Table Markup Formating (C in Figure 4). In this strategy, we explore describing the tabular data using markup languages, including CSV, JSON, Markdown, and XML. **Table2CSV** converts tabular data into CSV format, where each line represents a data record containing one or more comma-separated fields. **Table2JSON** aims to convert tabular data into a JSON object, where data is represented in <name, value> pairs, enclosed by curly braces "{}" for objects and square brackets "[]" for arrays. **Table2MD** aims to convert the table into a Markdown file, which uses simple and intuitive syntax to denote structure and style. **Table2XML** aims to convert the table into an XML file, which is an eXtensible Markup Language used to represent structured data in a human-readable and machine-readable way.

D. Table Programming (D in Figure 4). In order to feed the structured tabular data into LLMs, we propose to represent the structured tabular data in programming languages. **Table2SQL** uses `Create` statements to describe database schema [38], which emphasizes the relationship among tables. For showcasing the content of a database, **+Select** [38] employs the "`SELECT * FROM Table LIMIT R`" query to display the first *R* rows of each table. **Table2Code** proposes to transform the tabular data into general-purpose programming languages. In particular, we resort to the Python programming language, which effectively employs an inherent object-oriented paradigm and is well-suited for encoding structured data [50]. Moreover, we utilize Python's type hinting feature and represent tabular data using a class-based representation in Python. We define classes of `Table`, `Column`, `Constraint`, `Primary_Key`, `Foreign_Key`, and `Record`, followed by the instantiation of these classes to create a table object.

3.3 Visualization Query Generation

We utilize APIs provided by OpenAI in the GPT-3.5 and GPT-4 series to engage with LLMs. Specifically, we initially represent the tabular data with the format as outlined in Figure 4. Then, we construct the tables and queries into prompts as delineated in Figure 2. Finally, these prepared prompts are subsequently fed into LLMs via API calls. Our research methodology can be applied to any LLMs that support prompting.

3.4 Language-Aware Rending

According to nvBench [26], the visualization query can be presented as the tree format as introduced in [27] for fair evaluation of grammar. These *Abstract Syntax Trees (ASTs)* can then be translated to target visualization specification syntax, so as to render the visualization charts. The translation from a visualization query to a target visualization specification is hard-coded based on the grammar of ASTs. Currently, the nvBench provides a module of Python 3 code [27] for converting visualization query to Vega-Lite. Thus, the pipeline is fully automated and enables an effortless evaluation of the visualization query.

4 EVALUATION SETUP

We investigate the capability of LLMs in automating data visualization from natural-language descriptions, by answering the following three *Research Questions (RQs).*

- **RQ1 [Prompt Engineering]:** How to feed the natural-language query as well as the structural table into LLMs via prompting?
- **RQ2 [Overall Performance]:** How do the LLMs perform when compared with several existing models in NL2Vis?
- **RQ3 [Iterative Updating]:** Can the results be iteratively updated via some optimization strategies?

To address RQ1, we first explore the conversion of structured tables into sequential prompts, followed by an examination of how the content of these tables influences the results. To answer RQ2, we conduct a comparative analysis of the performance exhibited by both conventional neural networks and LLMs concerning NL2Vis. Furthermore, we analyze the impact of the number of in-context demonstrations on the performance of LLMs. To answer RQ3, we delve into instances of failure within the LLMs and subsequently propose strategies for enhancing model performance through optimization techniques.

All experiments are conducted on a machine with 252 GB memory and 4 Tesla V100 32GB GPUs. We use the default hyperparameter settings provided by each method. Besides, we split the dataset into a training dataset, a valid dataset, and a test dataset with 7:2:1 based on in-domain and cross-domain settings, as explained in Sec. 4.1. We use the test dataset for evaluation and the training dataset for demonstrations.

4.1 Dataset

We employ the widely-recognized benchmark nvBench [26] to assess performance in the NL2Vis task. The nvBench dataset encompasses 780 relational tables sourced from 153 databases across 105 diverse domains such as sports, colleges, hospitals, and more. It features 7,247 visualizations, resulting in 25,750 pairs of natural-language descriptions and corresponding visualizations.

Domain Setting. In prior studies [27, 28], datasets are predominantly partitioned randomly based on visualization queries, without considering database divisions. Upon re-implementing the ncNet[1], we notice that databases from the test set are exposed during the training process, constituting an in-domain setting. We propose a cross-domain setting by partitioning the dataset such that there is no overlap between databases in the training and test datasets. In this setup, models are required to predict queries on the unseen databases. To ensure a fair and comprehensive evaluation, we conduct experiments in both in-domain and cross-domain settings.

Multi-Table Setting. In our experiments, we categorize scenarios based on the number of tables involved in the input. Instances with multiple tables are designated as "*join*" scenarios, while those centered on a single table are categorized as "*non-join*" scenarios. In "*join*" scenarios, our objective entails generating visualizations by integrating information from multiple tables. This process involves merging data based on shared columns, presenting challenges to both data integration and the subsequent visualization efforts. Conversely, "*non-join*" cases involve generating visualizations from individual tables, acting as a benchmark for evaluating the model's ability to manage less complex data structures.

[1]https://github.com/Thanksyy/ncNet

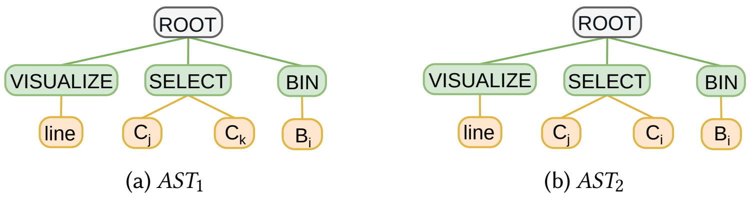


Fig. 5. An illustration of ASTs of VQL queries. Even though they may not be exactly matched, their execution results are identical.

4.2 Evaluation Metrics

To ensure a thorough and equitable assessment of the generated VQL queries, we employ three widely recognized metrics: exact accuracy, execution accuracy, and component accuracy, as established in the NL2VIS task [27]. Before introducing these metrics, we first introduce two visualization queries represented as ASTs, as shown in Figure 5. Each of them has three subtrees based on the grammar of VQL described in Sec. 2.1. Specifically, the node of `VISUALIZE` denotes a line chart type. The `SELECT` node combines the attributes of the columns C_j and C_k. The `BIN` node with B_i sets a bucket of values in the temporal column.

Exact Accuracy [27]. This metric is designed to assess the exact match between the predicted AST and the ground-truth AST of VQL queries. It can be formulated as $Acc_{AST} = N_{AST}/N$, where N_{AST} represents the count of generated ASTs that are exactly equivalent to the ground truth ASTs in each node, and N signifies the total number of ASTs under consideration.

Execution Accuracy [27]. This metric measures the accuracy of the visualization results by determining whether the predicted visualization aligns with the ground truth. It can be calculated using the formula $Acc_{exe} = N_{exe}/N$, where N_{exe} denotes the number of VQL whose results match the ground truth in the execution, and N denotes the total number of visualizations. From Figure 5, it is apparent that the subtrees rooted at "`VISUALIZE`" and "`BIN`" retain consistent structures between AST_1 and AST_2. While variations within the "`SELECT`" subtrees suggest they are not exact matches. Even though, the VQL execution results could be matched if $x/y/z$-axis data (executed by VQL) are correct, as exemplified by queries such as "`VISUALIZE line SELECT date, COUNT(date) from payments BIN date by month`" and "`VISUALIZE line SELECT date, date_count from payments BIN date by month`".

4.3 Comparison Models for NL2VIS

Traditional Neural Models. SEQ2VIS [27] is a sequence-to-sequence model that initially encodes the natural-language query into a hidden embedding using an LSTM and subsequently decodes it into a visualization through another LSTM. Transformer [46] is another encoder-decoder network that has been considered a foundational component in LLMs. We also apply the Transformer into NL2VIS. Based on Transformer, ncNet [28] designs several novel visualization-aware optimizations, such as using attention-forcing to optimize the learning process and visualization-aware rendering to produce better visualization results. RGVisNet [43] is a hybrid framework for NL2VIS to retrieve, refine, and generate visualizations from a large codebase using a graph neural network.

LLMs (Finetuned Models). T5 [42] is an encoder-decoder model pre-trained on a multi-task mixture of unsupervised and supervised tasks for which each task is converted into a text-to-text format. T5-Small is a released T5 model with 60 million parameters. T5-Base is a released T5 model with 220 million parameters.

LLMs (Inference-only Models). Chat2Vis [31] leverages the power of `code-davinci-002` to enable users to generate data visualizations using natural-language queries in Python plots and measures results against nvBench. Our evaluation focuses on the models accessible via the OpenAI API: GPT-3.5 (`text-davinci-002`, `text-davinci-003`, `gpt-3.5-turbo-16k`), and GPT-4 (`gpt-4`). GPT-3.5 is a set of models built on the InstructGPT [35], trained on a large corpus of programming languages and natural languages. `text-davinci-003`[2] is fine-tuned by reinforcement learning from human feedback [6], which improves its ability to generate better quality and longer output. `text-davinci-002`[2] is trained with supervised fine-tuning instead of reinforcement learning. `gpt-3.5-turbo-16k` is optimized for chat applications and increased input length. GPT-4 is a large multimodal model that is capable of solving complex problems with greater accuracy than any of the previous models, due to its excellent general knowledge and reasoning capabilities [11]. `gpt-4` revolutionizes loss function computation by incorporating a scaling law and an irreducible loss term [34], enabling precise prediction of final loss within the internal codebase.

5 RESULTS AND ANALYSIS

In this section, we present the experimental results for each RQ with comprehensive analysis.

5.1 RQ1: Prompt Engineering

In this RQ, we investigate various strategies for inputting structured tabular data into LLMs through prompting and analyze which aspect of the table content has the greatest influence on NL2Vis.

5.1.1 RQ1-1: How to transform the structured tabular data into sequential prompts? We evaluate the performance of the model `text-davinci-003` while varying the methods for transforming tabular data into textual prompts. We carry out this assessment in both cross-domain and in-domain settings. Additionally, we investigate configurations for both non-join and join scenarios, where single-table and multi-table contexts are considered, respectively. Note that when transforming tabular data into a markup format, we only consider a single row of content, specifically the one most relevant to the input question, as determined by Jaccard similarity.

Table 2 presents the model performance for `text-davinci-003` across various methods of transforming tabular data into textual prompts, considering both cross-domain and in-domain scenarios. All results are derived from a single-shot example provided in the ICL. From this table, it is clear that feeding LLMs with tables by encoding them in programming languages is the most effective method, both in the in-domain and cross-domain settings. Specifically, under the in-domain setting, prompting table via *Table2SQL*, *Table2XML*, and *Table2JSON* achieves the best performance, reaching up to 61% in terms of the Exact Accuracy. While under the cross-domain setting, prompting table via *Table2Code*, *Table2SQL*, and *Chat2Vis** achieves the best Execution Accuracy of 56%, 55%, and 55%, respectively. It suggests that converting structured tabular data into machine-readable markup formats and using general-purpose programming languages can effectively design prompts to interact with LLMs for NL2Vis. When comparing the prompts of *Table2Code* to *Chat2Vis**, we can see that *Table2Code* obtains 6% and 1% improvement in terms of the Execution Accuracy in the in-domain and cross-domain settings, respectively. Interestingly, we observe that the LLM-based model exhibits slight fluctuations in the performance of several settings. In cross-domain scenarios, the optimal performance is achieved through the application of table programming to transform input tables; however, this trend does not persist in in-domain settings. One plausible explanation for this observation is that, in the in-domain scenario, where the

[2]Note that this study was conducted between April and October 2023. Subsequently, as of January 2024, the `text-davinci-003` and `text-davinci-002` have been upgraded to `gpt-3.5-turbo-instruct`. More details are referred to OpenAI documentation: https://platform.openai.com/docs/deprecations.

Table 2. Performance of the model `text-davinci-003` while varying the methods for transforming tabular data into textual prompts, under both the cross-domain and in-domain settings.

	Cross-domain						**In-domain**					
	Non-join		**Join**		**Overall**		**Non-join**		**Join**		**Overall**	
	Exa.	**Exe.**	**Exa.**	**Exe.**	**Exa.**	**Exe.**	**Exa.**	**Exe.**	**Exa.**	**Exe.**	**Exa.**	**Exe.**
Table Serialization												
Schema	0.33	0.50	0.06	0.09	0.32	0.47	0.30	0.31	0.27	0.23	0.29	0.28
Table (Column)	0.37	0.56	0.09	0.11	0.36	0.54	0.60	0.59	0.52	0.44	0.57	0.53
Column=[]	0.37	0.54	0.11	0.12	0.36	0.52	0.61	0.61	0.52	0.45	0.58	0.55
Table Summarization												
Table2NL	0.35	0.54	0.13	0.17	0.34	0.52	0.63	0.63	0.53	0.46	0.60	0.57
Chat2Vis*	0.37	0.57	0.14	0.20	**0.36**	**0.55**	0.58	0.61	0.32	0.28	0.49	0.49
Table Markup Format												
Table2JSON	0.36	0.54	0.08	0.15	0.35	0.52	0.63	0.63	**0.56**	**0.48**	**0.61**	0.57
Table2CSV	0.35	0.52	0.07	0.12	0.34	0.50	0.59	0.59	0.54	0.46	0.57	0.54
Table2MD	0.36	0.53	0.07	0.09	0.34	0.50	0.59	0.61	0.53	0.46	0.57	0.56
Table2XML	0.36	0.54	0.08	0.16	0.35	0.52	0.63	0.64	0.56	0.47	**0.61**	**0.58**
Table Programming												
Table2SQL	**0.39**	**0.58**	**0.17**	**0.25**	**0.38**	**0.55**	**0.64**	**0.64**	0.53	0.47	**0.61**	**0.58**
Table2Code	0.36	**0.59**	**0.18**	0.08	0.34	**0.56**	0.53	0.58	0.47	0.43	0.55	0.55

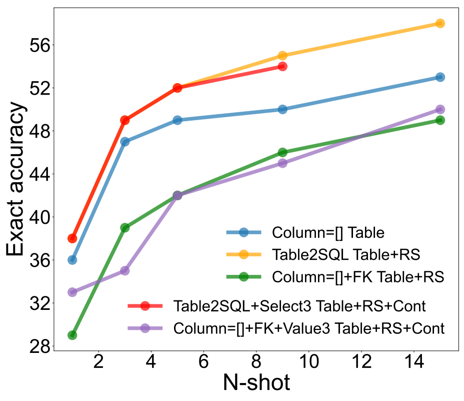
data adheres to a similar distribution, the model has the opportunity to discern the schema of the test database from demonstration examples during in-context learning. Consequently, the impact of exploring table transformations becomes diminished within this specific in-domain context.

Furthermore, when comes to the non-join and join scenarios under the cross-domain setting, we can see that our proposed methods of converting tabular data into code can still maintain the best performance. For example, *Table2Code* surpasses the state-of-the-art *Chat2Vis** by 2% and 4% in terms of the Execution Accuracy and Exact Accuracy when handling the non-join cases and join cases, respectively. We attribute the improvements to the effective preservation of inner structural information within tables when concerting structured tabular data into source code. This benefit arises from the inherent interplay between the structural aspects of source code and tabular data.

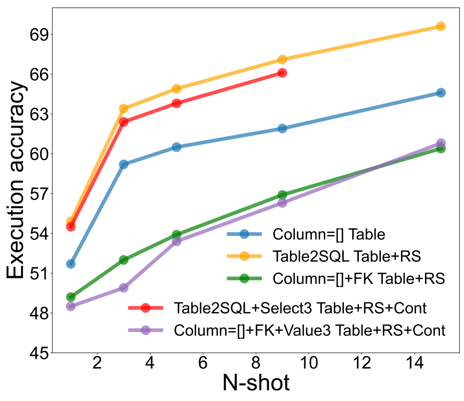
Finding 1-1. Converting structured tabular data into machine-readable markup formats (e.g., JSON, XML) or utilizing general-purpose programming languages (e.g., SQL, Python) yields superior performance compared to relying solely on natural-language summaries or straightforward table serialization.

5.1.2 *RQ1-2: What table content should be considered in NL2Vis?*

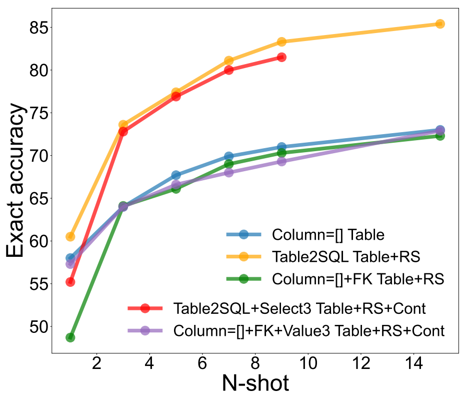
We further dive into analyzing the importance of table components in prompt construction and pinpoint the most reliant table component in different domain and few-shot settings. We categorize table contents into three levels: (i) *Table Schema*, which encompasses table names and column names, (ii) *+RS*, denoting relationships within the table, such as foreign keys, and (iii) *+Cont*, encompassing the actual content of the table, including the data in table row values. To simplify our experiment, we have chosen table serialization as the primary prompting approach. More specifically, our exploration encompasses



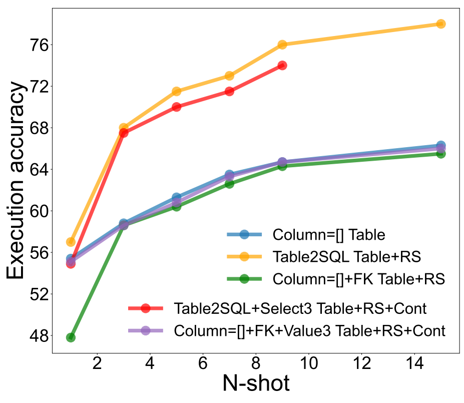
(a) Exa. Acc. in cross-domain



(b) Exe. Acc. in cross-domain



(c) Exa. Acc. in in-domain



(d) Exe. Acc. in in-domain

Fig. 6. Exact Accuracy and Execution Accuracy of `text-davinci-003` under the cross-domain and in-domain settings, with 1, 3, 5, 7, and 15 demonstrations provided in in-context learning. RS and Cont correspond to the table relationship and table content, respectively.

three distinct variants: (i) *Column=[]*, (ii) *Column=[]+FK*, *Table2SQL*, and (iii) *Column=[]+FK+Value3*, *Table2SQL+Select3*.

Figure 6 shows the Exact Accuracy and Execution Accuracy of `text-davinci-003` under the cross-domain and in-domain settings, with few demonstrations provided in in-context learning. From this figure, it is evident that the performance of the *Table Column=[]* configuration surpasses that of the *Table+RS Column=[]+FK* and *Table+RS+Cont Column=[]+FK+Value* configurations in both in-domain and cross-domain settings. This observation underscores the significance of the table schema as a comprehensive element within the prompt for *Column=[]*.

Moreover, when comparing the disparity between *Table Column=[]* and *Table+RS Column=[]+FK*, we observe that in the in-domain setting, the gap slightly narrows as the number of examples increases. This suggests that table knowledge in prompts has minimal influence on input, likely due to the fact that, in the in-domain setting, LLMs have seen test tables during demonstrations from the same database domain. However, a significant gap emerges in the cross-domain setting (i.e., increasing from 2.5% to 7.2% in terms of Execution Accuracy as the number of examples increases from 1 to 3), indicating that the additional table knowledge, specifically table relationships and

Table 3. Results of comparing LLMs with several baselines.

	Cross-domain		In-domain	
	Exa. Acc.	Exe. Acc.	Exa. Acc.	Exe. Acc.
Baseline				
Seq2Vis	0.02	-	0.66	-
Transformer	0.03	-	0.73	-
ncNet	0.26	-	**0.77**	-
RGVisNet	**0.45**	-	-	-
Chat2Vis	-	0.43	-	-
Finetuned				
T5-Small	0.60	0.61	0.92	0.81
T5-Base	0.71	0.72	0.93	0.82
Inference-only				
`text-davinci-002`	0.57	0.68	0.84	0.75
`text-davinci-003`	0.58	0.70	0.87	0.77
`gpt-3.5-turbo-16k`	0.56	0.63	0.59	0.59
`gpt-4`	0.61	0.72	0.83	0.74

content, in the *Column=[]* prompt is superfluous. Moreover, this redundancy could potentially introduce complexity and adversely impact the performance of LLMs in the NL2Vis task.

Additionally, when comparing the performance of *Table+RS Table2SQL* and *Table+RS+Cont Table2SQL+Select3*, we observe that there is no reduction in the gap between these two prompts. This observation suggests that *Table2SQL* remains effective even without the inclusion of row values. This can be attributed to the nature of the NL2Vis task, where the importance of table content often takes a secondary role. Users typically only need to reference table and column names to convey their visualization requirements. Essentially, the model's primary task is to establish a seamless connection between the natural-language query and the database schema, subsequently generating the visualization query successfully. This underscores that table content may not play a significant role in this context.

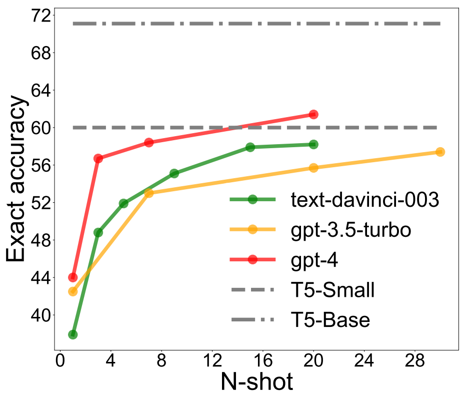
Finally, when comparing the performance of *Table+RS Column=[] +FK* and *Table+RS Table2SQL*, we can observe that while both prompts incorporate table schema and table relationship knowledge, they yield markedly distinct results. This observation underscores the substantial impact of well-crafted formats for representing structured tabular data.

Finding 1-2. Table schema plays an important role in the task of NL2Vis, both under the cross-domain and in-domain settings.

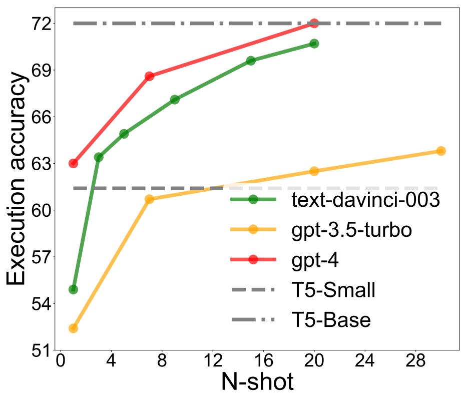
5.2 RQ2: Overall Performance

In this RQ, we investigate the overall performance of LLMs, and compare them with several traditional neural models for NL2Vis, in both in-domain and cross-domain settings.

5.2.1 RQ2-1: How do the LLMs perform when compared with existing works? To answer this RQ, we conduct a comparative analysis to evaluate LLMs, including the fine-tuned T5 models together with the inference-only GPT-3.5 and GPT-4 series models, against several state-of-the-art baseline models. We fine-tune T5-Small and T5-Base for the NL2Vis task with a maximum of 11.6*k* steps



(a) Exa. Acc.



(b) Exe. Acc.

Fig. 7. Exact Accuracy and Execution Accuracy with varying number of support examples. The x-axis indicates the number of few-shot examples used.

and 100*k* steps, respectively. Our configurations specify a maximum learning rate of 1×10^{-3}, a batch size of 4, and a dropout rate of 0.1. To set up a fair comparison, we explore inference-only models using the same *Table2SQL* prompt with 20 examples provided in the ICL (20-shot).

Table 3 shows the overall performance of LLMs as well as several baselines, under both the cross-domain and in-domain settings for the NL2Vis task. From this table, it is clear that LLMs (both the finetuned models and inference-only models) significantly outperform the traditional baselines, both in the in-domain and cross-domain settings. A closer examination of in-domain performance reveals that the predominant baseline models, namely, Seq2Vis, Transformer, and ncNet, achieve their peak scores at 66%, 73%, and 77%, respectively. This underscores the role of ncNet in enhancing the seq2seq models through attention forcing. Furthermore, the fine-tuned language models, T5-Small and T5-Base, significantly outperform state-of-the-art methods with scores of 92% and 93%, respectively. For in-context learning, inference-only models also achieve commendable scores. Notably, `text-davinci-003` stands out with an 87% score, surpassing existing baseline methods by 10%. This underscores the idea that LLMs, trained on natural-language text or code corpora, represent the preeminent models for the task. The enhancement is attributed to the robust capabilities of LLMs in comprehending the inherent knowledge, including table schema and structure, within the same domain, leading to superior responses to new queries.

Furthermore, when comparing performance in cross-domain scenarios, we are genuinely surprised by the substantial decline observed across all baseline models (e.g., ncNet, plummeting from 77% to 26%). It is noteworthy that RGVisNet is the first to address this phenomenon and succeeds in boosting cross-domain performance by 45%. This underscores the vast potential for enhancement in this context. The underperformance of traditional neural models, designed for random-splitting data settings, becomes evident as they struggle to generalize to previously unseen databases and grapple with the task of linking natural language to table schema. However, the fine-tuned models, specifically T5-Small (60%) and T5-Base (71%), significantly outshine the baseline models. Among the inference-only models, `gpt-4` shows the most impressive improvement, surging by 61%, while `gpt-3.5-turbo-16k` lags behind with a 56% increase. This serves as a testament to the exceptional generalization capabilities of LLMs, empowering them to effectively assimilate new knowledge and establish connections with queries when applied to previously uncharted databases. The result is an unequivocal improvement in visualization results.

Table 4. Statistics of model parameters and cost time.

	Parameters	**Cost Time**	**Model Size**
T5-Small	60M	3 days	200MB
T5-Base	220M	5 days	500MB
`text-davinci-003`	1.5B	15 hours	1GB
`gpt-3.5-turbo-16k`	4B	4 hours	2GB
`gpt-4`	-	4 hours	-

Lastly, in comparing the performance of fine-tuned models with that of inference-only models, we observe that while fine-tuned models exhibit the most desirable performance, their advantages in fine-tuning can be equaled by inference-only models with in-context learning. As shown in Figure 7, both fine-tuned T5-Small and T5-Base models achieve execution accuracies of 61% and 72%, respectively. The models `text-davinci-003` and `gpt-3.5-turbo-16k` demonstrate superior performance compared to the T5-Small model, particularly with 3-shot and 13-shot scenarios. This suggests that, after exposure to an ample number of examples in contextual learning, LLMs acquire the capability to utilize demonstrations for precise inference and the generation of visualizations in tables not encountered before. Additionally, we investigate the comparative costs of fine-tuned models and inference-only models with 20-shot in-context learning. From Table 4, we can observe that even though the inference-only models typically incur significant computational costs due to their huge size, they offer considerable time savings through in-context learning.

Finding 2-1. The LLMs demonstrate a remarkable ability to surpass state-of-the-art models in both cross-domain and in-domain scenarios, achieving an improvement of 26% and 16% in terms of Exact Accuracy, respectively. This underscores the LLMs' exceptional generalization capabilities. Furthermore, with the provision of more few-shot samples, the performance of inference-only models consistently improves, eventually surpassing that of fine-tuned models.

5.2.2 *RQ2-2: How does the number of in-context demonstrations affect the performance of LLMs?*

In this RQ, we explore the impact of demonstration selection methods on in-context learning for NL2Vis task, by selecting different databases and varying quantities of examples in demonstration. To investigate the effect of the number of databases and examples per database in the demonstration, we conduct experiments encompassing various combinations. Specifically, our demonstration examples for in-context learning are drawn from A distinct databases to simulate the cross-domain scenario. From each of these databases, we extract B paired instances consisting of a natural-language query and its corresponding VQL query. Collectively, this amounts to $C = A \times B$ examples. In our configuration, we permit both variables A and B to attain a maximum value of 4, ensuring that their cumulative total does not surpass the length limit specified by the prompt. We conduct this experiment using the *Table2SQL* prompt in a cross-domain scenario across all samples within the test dataset.

In Figure 8, it is evident that there is a notable improvement (45%-47%) in performance when all examples are sourced from the same database. This improvement is observed as the number of examples increases from 1 to 4. When the total number of examples is fixed (e.g., at 4), sourcing them from entirely different databases reveals superior performance (49%) compared to sourcing them from the same database (47%). It indicates that, in in-context learning, selecting a greater number of examples from diverse databases is beneficial.

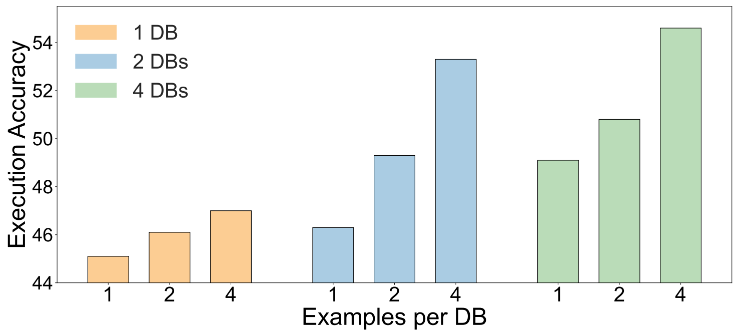


Fig. 8. The average Execution Accuracy across test dataset using *Table2SQL* by `text-davinci-003` with respect to different numbers of databases (DBs) and examples per database (Exp/DB) in the demonstration for in-context learning.

Finding 2-2. Multiple cross-domain demonstrations are generally more beneficial than examples derived from the same database domain.

5.2.3 User Study. We conduct a user study on LLMs to evaluate whether LLMs work well in the real world with users from different backgrounds. First, we design two tasks as follows: (1) Given tables and a target visualization with description, users express a natural-language query for creating such a visualization. (2) If the query can not generate the target visualization correctly, the users could choose to revise it three times. Next, we invite 3 graduate students as experts and 3 undergraduate students as non-experts, all majoring in computer science, to participate in the user study. Each expert possesses over six years of proficiency in software development, demonstrating advanced skills in data analysis and visualization. On the other hand, non-experts, with approximately two years of programming experience, are capable of executing basic visualization operations in Excel. Then, we select 5 databases at random to ensure a diverse range of data for our study. From each database, we pick 3 visualizations for each of the 4 difficulty levels (i.e., easy, medium, hard, and extra hard), resulting in a total of 60 visualizations. We design a command-line interface that enables users to engage in data visualization over the selected databases using LLMs by crafting natural-language queries. Subsequently, these crafted natural-language queries, along with the serialized table and a set of 20 demonstration examples, are input into the LLM (i.e., `text-davinci-003`) through in-context prompting. The generated VQL queries will be transformed into visualizations for users, allowing them to iteratively revise the natural-language queries.

Figure 10 shows the success rates of users querying for visualizations across four levels of difficulty. We observe that the experts are excellent in formulating queries for complex visualizations, successfully obtaining 95.6% hard charts. While the non-experts are proficient in queries for 84.4% easy charts. Figure 9 reports that non-experts take approximately 16 seconds more for the initial composition of queries and 15 seconds more for revision than experts. The system consistently maintains an average response time of 3 seconds for generating prompt examples and 2 seconds for VQL generation, applicable to both user groups. Finally, we collect feedback on the ability of LLMs for the NL2Vɪs task. All users acknowledge the LLM's proficient understanding of natural-language query for NL2Vɪs task, and appreciate its substantial facilitation of the visualization process.

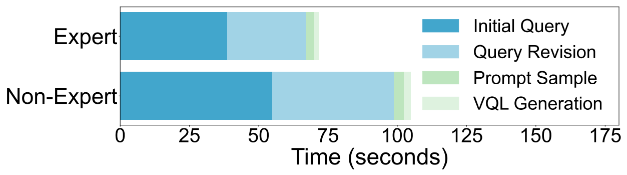


Fig. 9. Quantitative analysis of the composition of user time on average.

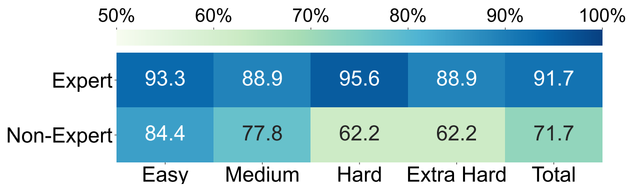


Fig. 10. The average success rates of experts and non-experts queried for 4 difficult levels of visualization.

5.3 RQ3: Iterative Updating

In this RQ, we commence by analyzing the conditions under which LLMs encounter failures in NL2Vis. Subsequently, we propose a strategy to mitigate these failures by iteratively updating the results through in-context learning with chain-of-thought.

5.3.1 RQ3-1: When do the LLMs fail in NL2Vis? To gain a deeper insight into the instances where LLMs encounter challenges, we conduct a comprehensive analysis of erroneous outputs. Specifically, we scrutinize all outcomes generated by the `text-davinci-003` model using the *Table2SQL* prompt with a 20-shot approach in a cross-domain scenario. We classify the cases of failure into distinct categories, depending on the component (e.g., `wrong tables` and `wrong columns`) of the VQL it struggles to predict accurately. Based on the AST of visualization query [26], a visualization consists of two components: the visual part (visualization types, axis) and the data part (transformation from a database) [27]. Specifically, the evaluation of visualization types involves measuring type tokens such as bar, scatter, line, and pie. With regard to the axis component, the assessment focuses on the "`SELECT`" component of the visualization query. Data part includes "`BIN`", "`GROUP`", "`JOIN`", "`COND (ORDER, WHERE, and AND/OR)`", and nested components.

Figure 11 presents a breakdown of these failure statistics. It is evident from this figure that data-related errors constitute the majority at 73.8%, which is approximately three times more prevalent than visual-related errors at 26.2%. This observation underscores the challenge of precisely identifying and visualizing the relevant data. In terms of the data-related errors, the highest proportion of errors is associated with the "cond" (condition) attribute (35.6%). This indicates a need to enhance the ability of LLMs to effectively handle queries related to filtering operations. This error analysis motivates us to iteratively update the results in a conversational manner, mirroring the functioning of LLMs.

Finding 3-1. In summary, the analysis reveals that errors in the data part of the visualization query are more frequent compared to those in the visual part. Respectively, errors are mainly related to data filtering and the y-axis of visualization.

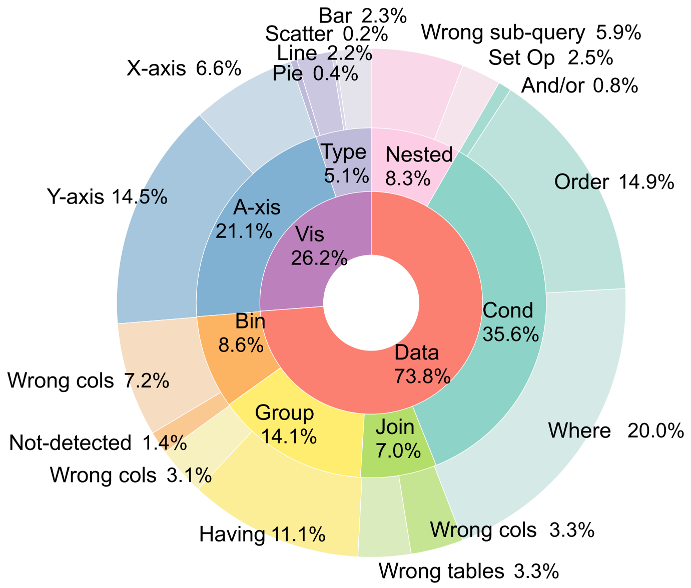


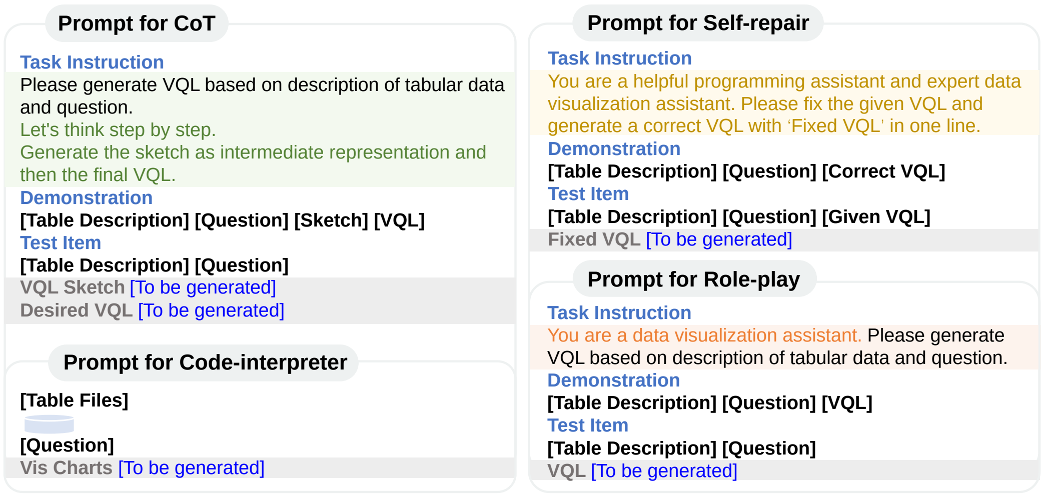
Fig. 11. Statistics of failures by `text-davinci-003` in 20-shot using *Table2SQL*.

5.3.2 RQ3-2: Can we iteratively update the results via optimization strategies? Here, we employ the CoT strategy with manual construction [52] to enhance the prompts for NL2Vis, infusing LLMs (i.e., `gpt-3.5-turbo` and `gpt-4`) with an intermediary cognitive process. Our approach utilizes a sketch as an intermediate expression, containing essential keywords from the visualization query. Additionally, we harness `gpt-3.5-turbo`'s innate self-instruct capability by introducing the phrase "*Let's think step by step.*" This combined strategy is denoted as CoT. Furthermore, we delve into role-playing, where `gpt-3.5-turbo` takes on the persona of a visualization expert. In consideration of `gpt-4`'s self-repair capabilities, we explore instructions for rectifying visualization queries. The optimization strategy is elucidated in Figure 12. Additionally, we investigate GPT-4's newly introduced code-interpreter feature in ChatGPT Plus[3], which enables GPT-4 to demonstrate programming proficiency within a conversational context, thus facilitating user learning and problem-solving.

We evaluate the total failure results of `text-davinci-003` with *Table2SQL* in 20 shots to explore the optimization strategy. For `gpt-3.5-turbo`, we employ the following strategies. (1) CoT guided by the prompt "*Let's think step by step. Generate the sketch as an intermediate representation and then the final VQL*". (2) Role-playing, where the prompt "*You are a data visualization assistant*" sets the model's persona. For GPT-4 (`gpt-4`), we explore the (3) self-repair by adopting the following prompt "*You are a helpful programming assistant and expert data visualization assistant. Please fix the given VQL and generate a correct VQL*". We also explore the (4) code-interpreter by uploading database files and entering natural-language queries on the ChatGPT Plus website. By pairing every same VQL with only the first natural-language query as a test example, we filter 176 different charts from the LLM's failed dataset. The generated visualizations are manually checked for accuracy.

Figure 13 depicts the comparative performance of diverse optimization strategies employed by both `gpt-3.5-turbo` and `gpt-4`. Our findings reveal significant improvements in Execution

[3]https://chat.openai.com/?model=gpt-4-code-interpreter


Prompt for CoT

Task Instruction
Please generate VQL based on description of tabular data and question.
Let's think step by step.
Generate the sketch as intermediate representation and then the final VQL.
Demonstration
[Table Description] [Question] [Sketch] [VQL]
Test Item
[Table Description] [Question]
VQL Sketch [To be generated]
Desired VQL [To be generated]

Prompt for Code-interpreter

[Table Files]
[Question]
Vis Charts [To be generated]

Prompt for Self-repair

Task Instruction
You are a helpful programming assistant and expert data visualization assistant. Please fix the given VQL and generate a correct VQL with 'Fixed VQL' in one line.
Demonstration
[Table Description] [Question] [Correct VQL]
Test Item
[Table Description] [Question] [Given VQL]
Fixed VQL [To be generated]

Prompt for Role-play

Task Instruction
You are a data visualization assistant. Please generate VQL based on description of tabular data and question.
Demonstration
[Table Description] [Question] [VQL]
Test Item
[Table Description] [Question]
VQL [To be generated]

Fig. 12. Optimization strategies employed for LLMs to iteratively update the results.

Accuracy: the CoT strategy enhances accuracy by 9.3%, while the role-playing strategy shows an impressive 12.8% boost. Notably, gpt-4's self-repair strategy leads to a remarkable 13.3% accuracy improvement. When subjected to the code-interpreter on ChatGPT Plus within the extract dataset, the Execution Accuracy of visualization improves to 50.3%.

Finding 3-2. The Self-repair strategy outperforms the CoT and role-playing strategies in updating the results within NL2Vis. Furthermore, the code-interpreter in gpt-4 excels, indicating a promising avenue for future research.

6 DISCUSSION

In this section, we discuss our findings during the experiments and hope researchers can address some of the issues in future work.

6.1 Findings and Implications

In this study, we have uncovered several significant findings that offer valuable insights.

Finding 1: Investigating the strategies of inputting tables into LLMs, we find that representing tables in a programming language format is most effective, showcasing remarkable performance in addressing NL2Vis tasks when working with structured tabular data. Future work should delve deeper into investigating the intricacies of table representation within programming languages.

Finding 2: In analyzing the impact of three table components (i.e., schema, relationship, and content) in prompts, we identify that table schema is the most crucial component in both cross-domain and in-domain settings. Table content is found to be inconsequential in NL2Vis tasks, emphasizing the importance of the format and relationships in future research, especially when feeding extra large databases into LLMs.

Finding 3: LLMs exhibit superior capabilities for automating data visualization based on natural-language descriptions, outperforming state-of-the-art models in both cross-domain and in-domain scenarios. Furthermore, as more few-shot samples are provided, inference-only models excel, even

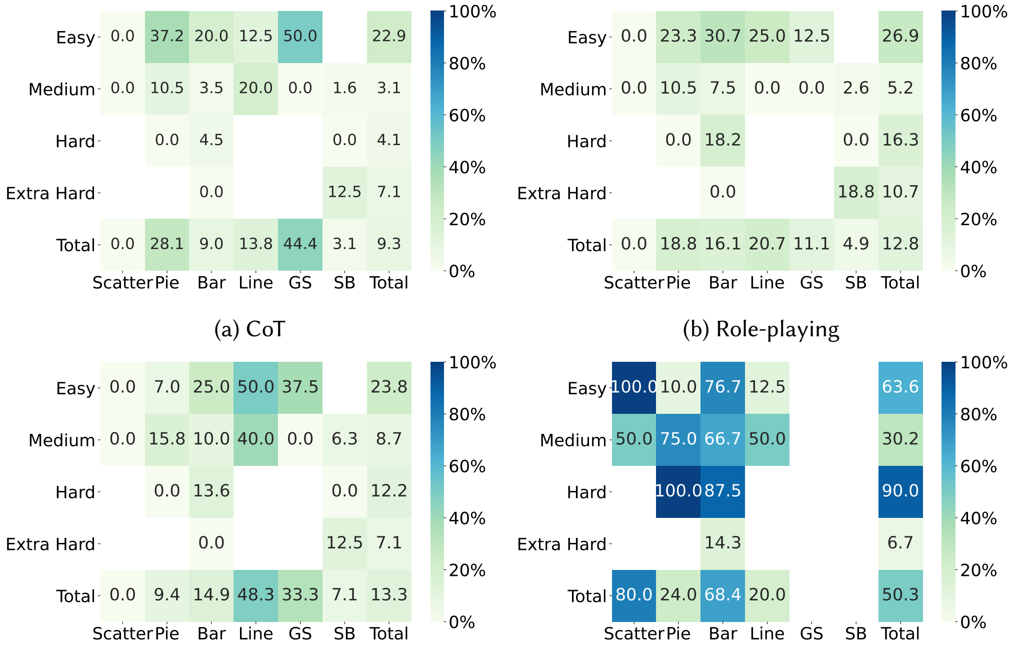


Fig. 13. The Execution Accuracy of optimization strategy in the failed results of `text-davinci-003` in one-shot with *Table2SQL*. GS and SB refer to grouping scatter and stacked bar chart types, respectively.

surpassing fine-tuned models. It is suggested to apply our framework to open-source LLMs for more general NL2Vis task, which involves implicit and multi-type table structures.

Finding 4: Investigating the selection of demonstrations in in-context learning, we determine that multiple out-of-domain demonstrations generally offer greater benefits than examples from the same database domain, which could guide the design of demonstration selection in the future.

Finding 5: The analysis of failed generation reveals that errors are more frequent in the data part of the visualization query, with particular issues related to data filtering and the y-axis of visualization. There is room for improvement in generating condition and group attributes, which inspires further exploration of iterative updating based on multi-turn dialogs of LLMs.

Finding 6: Optimization strategies such as the self-repair of `gpt-4` and ChatGPT Plus's code-interpreter demonstrate superior performance, although challenges persist in handling complex natural-language queries and join cases. Future work should focus on intuitive interfaces to enhance the usability and reliability of NL2Vis tasks in real-world applications.

6.2 Limitations

There are some potential limitations on the validity of our experimental results and conclusions.

Limited tasks and dataset. In this work, we explore the NL2Vis task with regular table structures and descriptive table context. However, in practice, tables could be in irregular structures, such as those featuring merged rows and columns in Excel, or containing mixed data content. On the other hand, table schema and column headers may not be descriptive. Future studies can extend the dataset to irregular tables for visualization, to enrich NL2Vis benchmarks in more applicable scenarios. Moreover, all experiments in this paper are assessed using a synthesized dataset derived from the NL2SQL dataset. We anticipate extending our study to real-world datasets in future work.

Limited visualization specification. In this paper, we choose VQL, an intermediate language that has been widely embraced in existing literature. While some may contend that visualizations can be directly generated through the generation of high-level specifications, such as Vega-Lite (in JSON format), we argue that this approach may encounter difficulties in accurately capturing specific details, such as chart type and rendering color. As part of our future work, we plan to explore the direct generation of diverse Vega-Lite specifications.

Support of conversational NL2Vis. Data analysts typically perform data visualization in a conversational manner. Take conversational visual analysis for example, a conversational natural-language inquiry can be made up of a number of separate but related natural-language inquiries. It is an intriguing and promising avenue to extend the NL2Vis benchmark to conversational visual data analytics.

Manual prompt design. Like other prompting strategies, the prompt design and optimizations are based on human comprehension and observations. The designer's knowledge may affect the effectiveness of the used prompts. Nonetheless, the purpose of our study is to investigate the viability of prompt design and related influential factors. Our experimental results demonstrate that the designed prompts are effective. In the future, we will investigate automated prompt construction techniques [24].

7 RELATED WORK

In this section, we review the related literature about NL2Vis task, LLMs for code generation, and LLMs for data engineering.

7.1 NL2Vis

NL2Vis is crucial in data analysis, aiming to generate visualization representations from brief succinct natural language explanations. While many approaches harnessing rule-based NLP techniques have been proposed [8, 13, 40] to address this challenge, their effectiveness is limited by the inflexibility of predefined rules in handling open-form natural-language inputs. Owing to the availability of large-scale corpora of natural-language descriptions paired with corresponding visualizations, such as nvBench [26], deep-learning-based techniques [25–28] have been employed to train a sequence-to-sequence model in an end-to-end manner for the NL2Vis task. For example, Seq2Vis [27] utilizes an LSTM to encode natural-language queries into hidden states, which are subsequently decoded into visualizations. Moreover, ncNet [28] considers another encoder-decoder architecture, the Transformer [46], to translate natural-language descriptions into visualizations. Leveraging the powerful capabilities of ChatGPT [5], Chat2Vis [31] invokes the API interfaces of `code-davinci-002` to enable users to create data visualizations using natural-language queries in Python plots. More recently, RGVisNet [43] has introduced a retrieval-based model, designed to retrieve the most pertinent visualization representation from an extensive visualization codebase for a given natural-language query.

7.2 LLMs for Code Generation

Code generation targets the automatic generation of program code from natural-language descriptions, which can assist developers in improving programming productivity and efficiency. In recent years, LLMs have exhibited remarkable capabilities in generating code [3], including Python programs, execution commands for Excel, and SQL queries for databases. These models are generally built upon the Transformer architecture and pre-trained on large-scale corpora using self-supervised objectives such as masked language modeling and next-sentence prediction [7]. Examples of these models include CodeGen [33], CodeT5+ [51], ChatGPT [5], StarCoder [21], and Code Llama [39]. To fully harness the zero-shot potential of LLMs, a range of techniques have emerged, including prompt tuning, in-context learning, chain-of-thought, and instruction tuning. In particular, in-context learning stands out as a method to fortify LLMs by providing contextual information or illustrative examples, as explored in [20] for code generation. Similarly, chain-of-thought is devised to ensure the logical coherence of LLM outputs, thereby enhancing the performance of code generation [19]. Moreover, instruction tuning has been conceived to enhance the generalization prowess of LLMs across various tasks, exemplified by the creation of Wizard-Coder [29], which augments the capabilities of StarCoder through the innovative *Evol-Instruct* approach to generate sophisticated code instructions.

7.3 LLMs for Data Engineering

In the field of data engineering and analysis, the integration of LLMs with data-centric tasks has opened avenues for transformative approaches to data interaction, processing, and visualization. Recently, numerous studies have been developed to weave natural language into tabular data analysis [14, 17, 18, 54, 55]. For instance, NL2SQL [14, 18] adeptly translates natural language into SQL commands to manipulate relational databases. Additionally, ChatExcel [2] and NL2Formula [56] have leveraged LLMs to generate Excel execution commands, thereby streamlining user interactions. SheetCopilot [17] has explored translating languages to VBA (*Visual Basic for Applications* - an embedded scripting language in Microsoft Excel), benefiting from a rich array of spreadsheet software functionalities. Data-Copilot [55], an LLM-based system, facilitates the automated management, invocation, and processing of a substantial volume of data from various sources, crafting sophisticated interface tools autonomously. Designed for table analysis, TableGPT [54], blends tables, natural language, and commands to manipulate data, visualize information, generate analysis reports, answer questions, and make predictions.

8 CONCLUSION AND FUTURE WORK

In this paper, we have investigated whether it is feasible to utilize LLMs for the NL2Vis task. Specifically, we compare LLMs including fine-tuned models (e.g., T5-Small, T5-Base) and inference-only models (e.g., `text-davinci-002`, `text-davinci-003`, `gpt-3.5 -turbo`, and `gpt-4`) against the state-of-the-art models. To start with, we investigate different approaches to transforming the structured tabular data into sequential prompts, so as to feed them into LLMs. Furthermore, we evaluate the LLMs on the NL2Vis benchmark under in-domain and cross-domain settings, against several traditional neural models for NL2Vis. At last, we analyze when the LLMs fail in NL2Vis, and propose to iteratively update the results using strategies such as chain-of-thought, role-playing, and code-interpreter.

In our future work, we plan to explore table representations by encoding tables with more well-designed programming languages. Moreover, we will extend the benchmarks to implicit and multi-type table structures for visualization in more applicable scenarios, to push forward the field of NL2Vis. While our current focus is on few-shot NL2Vis in GPT-3.5, our framework can also be applied to other LLMs, such as LLaMA [45]. Consequently, the application of our prompting methods to multiple semantic parsing tasks represents an intriguing avenue for future exploration.

ACKNOWLEDGMENTS

This work is supported by the Major Program (JD) of Hubei Province (Grant No. 2023BAA024). We would like to thank all the anonymous reviewers for their insightful comments.

REFERENCES

[1] [n. d.]. Amazon's QuickSight. https://aws.amazon.com/cn/blogs/aws/amazon-quicksight-q-to-answer-ad-hoc-business-questions.

[2] [n. d.]. ChatExcel. https://chatexcel.com.

[3] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258* (2021).

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Proceedings of the Advances in Neural Information Processing Systems*, Vol. 33. 1877–1901.

[5] ChatGPT. 2022. ChatGPT. https://openai.com/blog/chatgpt.

[6] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep Reinforcement Learning from Human Preferences. In *Advances in Neural Information Processing Systems*, Vol. 30.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186.

[8] Tong Gao, Mira Dontcheva, Eytan Adar, Zhicheng Liu, and Karrie G. Karahalios. 2015. DataTone: Managing Ambiguity in Natural Language Interfaces for Data Visualization. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (Charlotte, NC, USA) *(UIST '15)*. Association for Computing Machinery, New York, NY, USA, 489–500.

[9] Heng Gong, Yawei Sun, Xiaocheng Feng, Bing Qin, Wei Bi, Xiaojiang Liu, and Ting Liu. 2020. TableGPT: Few-shot Table-to-Text Generation with Table Structure Reconstruction and Content Matching. In *Proceedings of the 28th International Conference on Computational Linguistics*. International Committee on Computational Linguistics, Barcelona, Spain (Online), 1978–1988.

[10] GPT3.5. 2023. GPT3.5. https://platform.openai.com/docs/models/gpt-3-5.

[11] GPT4. 2023. GPT4. https://openai.com/research/gpt-4.

[12] Jiaxian Guo, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Boyang Li, Dacheng Tao, and Steven Hoi. 2023. From Images to Textual Prompts: Zero-shot Visual Question Answering with Frozen Large Language Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10867–10877.

[13] Enamul Hoque, Vidya Setlur, Melanie Tory, and Isaac Dykeman. 2017. Applying pragmatics principles for interaction with visual analytics. *IEEE Transactions on Visualization and Computer Graphics* 24, 1 (2017), 309–318.

[14] Chenxu Hu, Jie Fu, Chenzhuang Du, Simian Luo, Junbo Zhao, and Hang Zhao. 2023. ChatDB: Augmenting LLMs with Databases as Their Symbolic Memory. *arXiv preprint arXiv:2306.03901* (2023).

[15] Hyeonji Kim, Byeong-Hoon So, Wook-Shin Han, and Hongrae Lee. 2020. Natural language to SQL: Where are we today? *Proceedings of the VLDB Endowment* 13, 10 (2020), 1737–1750.

[16] Chuan Li. 2020. Demystifying gpt-3 language model: A technical overview. https://lambdalabs.com/blog/demystifying-gpt-3. [Online; accessed 1-Aug-2022].

[17] Hongxin Li, Jingran Su, Yuntao Chen, Qing Li, and Zhaoxiang Zhang. 2023. SheetCopilot: Bringing Software Productivity to the Next Level through Large Language Models. *arXiv preprint arXiv:2305.19308* (2023).

[18] Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. 2023. Graphix-T5: mixing pre-trained transformers with graph-aware layers for text-to-SQL parsing. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence (AAAI'23/IAAI'23/EAAI'23).* AAAI Press, Article 1467, 9 pages.

[19] Jia Li, Ge Li, Yongmin Li, and Zhi Jin. 2023. Enabling Programming Thinking in Large Language Models Toward Code Generation. *CoRR* abs/2305.06599 (2023).

[20] Jia Li, Ge Li, Chongyang Tao, Jia Li, Huangzhao Zhang, Fang Liu, and Zhi Jin. 2023. Large Language Model-Aware In-Context Learning for Code Generation. *CoRR* abs/2310.09748 (2023).

[21] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023. StarCoder: may the source be with you! *arXiv preprint arXiv:2305.06161* (2023).

[22] Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S Yu. 2023. A comprehensive evaluation of ChatGPT's zero-shot Text-to-SQL capability. *arXiv preprint arXiv:2303.13547* (2023).

[23] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. In *Advances in Neural Information Processing Systems*, Vol. 36. 21558–21572.

[24] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *ACM Comput. Surv.* 55, 9, Article 195 (jan 2023), 35 pages.

[25] Yuyu Luo, Xuedi Qin, Nan Tang, Guoliang Li, and Xinran Wang. 2018. DeepEye: Creating Good Data Visualizations by Keyword Search. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18).* Association for Computing Machinery, New York, NY, USA, 1733–1736.

[26] Yuyu Luo, Jiawei Tang, and Guoliang Li. 2021. nvBench: A large-scale synthesized dataset for cross-domain natural language to visualization task. *arXiv preprint arXiv:2112.12926* (2021).

[27] Yuyu Luo, Nan Tang, Guoliang Li, Chengliang Chai, Wenbo Li, and Xuedi Qin. 2021. Synthesizing Natural Language to Visualization (NL2VIS) Benchmarks from NL2SQL Benchmarks. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21).* Association for Computing Machinery, New York, NY, USA, 1235–1247.

[28] Yuyu Luo, Nan Tang, Guoliang Li, Jiawei Tang, Chengliang Chai, and Xuedi Qin. 2022. Natural Language to Visualization by Neural Machine Translation. *IEEE Transactions on Visualization and Computer Graphics* 28, 1 (2022), 217–226.

[29] Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023. WizardCoder: Empowering Code Large Language Models with Evol-Instruct. *CoRR* abs/2306.08568 (2023).

[30] Chenyang Lyu, Jitao Xu, and Longyue Wang. 2023. New trends in machine translation using large language models: Case examples with chatgpt. *arXiv preprint arXiv:2305.01181* (2023).

[31] Paula Maddigan and Teo Susnjak. 2023. Chat2vis: Fine-tuning data visualisations using multilingual natural language text and pre-trained large language models. *arXiv preprint arXiv:2303.14292* (2023).

[32] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations.* 55–60.

[33] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474* (2022).

[34] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson,

Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2023. Gpt-4 technical report.

[35] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, Vol. 35. 27730–27744.

[36] Shuyin Ouyang, Jie M. Zhang, Mark Harman, and Meng Wang. 2023. LLM is Like a Box of Chocolates: the Non-determinism of ChatGPT in Code Generation. *arXiv preprint arXiv:2308.02828* (2023).

[37] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. In *Proceedings of the Advances in Neural Information Processing Systems*, Vol. 36. 36339–36348.

[38] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research* 21, 140 (2020), 1–67.

[39] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950* (2023).

[40] Vidya Setlur, Sarah E. Battersby, Melanie Tory, Rich Gossweiler, and Angel X. Chang. 2016. Eviza: A natural language interface for visual analysis. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. 365–377.

[41] Vidya Setlur, Melanie Tory, and Alex Djalali. 2019. Inferencing underspecified natural language utterances in visual analysis. In *Proceedings of the 24th International Conference on Intelligent User Interfaces* (Marina del Ray, California) *(IUI '19)*. Association for Computing Machinery, New York, NY, USA, 40–51.

[42] Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2021. Compositional Generalization and Natural Language Variation: Can a Semantic Parsing Approach Handle Both?. 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). Association for Computational Linguistics, Online, 922–938.

[43] Yuanfeng Song, Xuefang Zhao, Raymond Chi-Wing Wong, and Di Jiang. 2022. RGVisNet: A Hybrid Retrieval-Generation Neural Framework Towards Automatic Data Visualization Generation. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Washington DC, USA) *(KDD '22).* Association for Computing Machinery, New York, NY, USA, 1646–1655.

[44] Yiming Tan, Dehai Min, Yu Li, Wenbo Li, Nan Hu, Yongrui Chen, and Guilin Qi. 2023. Evaluation of ChatGPT as a question answering system for answering complex questions. *arXiv preprint arXiv:2303.07992* (2023).

[45] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Ł. ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Proceedings of the Advances in Neural Information Processing Systems*, Vol. 30.

[47] Randle Aaron M. Villanueva and Zhuo Job Chen. 2019. ggplot2: Elegant Graphics for Data Analysis (2nd ed.). *Measurement: Interdisciplinary Research and Perspectives* 17, 3 (2019), 160–167.

[48] Yao Wan, Yang He, Zhangqian Bi, Jianguo Zhang, Yulei Sui, Hongyu Zhang, Kazuma Hashimoto, Hai Jin, Guandong Xu, Caiming Xiong, et al. 2022. NaturalCC: an open-source toolkit for code intelligence. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings.* 149–153.

[49] Longyue Wang, Chenyang Lyu, Tianbo Ji, Zhirui Zhang, Dian Yu, Shuming Shi, and Zhaopeng Tu. 2023. Document-Level Machine Translation with Large Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Singapore, 16646–16661.

[50] Xingyao Wang, Sha Li, and Heng Ji. 2023. Code4Struct: Code Generation for Few-Shot Event Structure Prediction. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers).* Association for Computational Linguistics, Toronto, Canada, 3640–3663.

[51] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 8696–8708.

[52] 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 *Proceedings of the Advances in Neural Information Processing Systems*, Vol. 35. 24824–24837.

[53] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887* (2018).

[54] Liangyu Zha, Junlin Zhou, Liyao Li, Rui Wang, Qingyi Huang, Saisai Yang, Jing Yuan, Changbao Su, Xiang Li, Aofeng Su, Tao Zhang, Chen Zhou, Kaizhe Shou, Miao Wang, Wufang Zhu, Guoshan Lu, Chao Ye, Yali Ye, Wentao Ye, Yiming Zhang, Xinglong Deng, Jie Xu, Haobo Wang, Gang Chen, and Junbo Zhao. 2023. TableGPT: Towards Unifying Tables, Nature Language and Commands into One GPT. *arXiv preprint arXiv:2307.08674* (2023).

[55] Wenqi Zhang, Yongliang Shen, Weiming Lu, and Yueting Zhuang. 2023. Data-Copilot: Bridging Billions of Data and Humans with Autonomous Workflow. *arXiv preprint arXiv:2306.07209* (2023).

[56] Wei Zhao, Zhitao Hou, Siyuan Wu, Yang Gao, Haoyu Dong, Yao Wan, Hongyu Zhang, Yulei Sui, and Haidong Zhang. 2024. NL2Formula: Generating Spreadsheet Formulas from Natural Language Queries. In *Findings of the Association for Computational Linguistics: EACL 2024, St. Julian's, Malta, March 17-22, 2024.* 2377–2388.

[57] Li Zhong and Zilong Wang. 2023. A Study on Robustness and Reliability of Large Language Model Code Generation. *arXiv preprint arXiv:2308.10335* (2023).

[58] Jonathan Zong, Josh Pollock, Dylan Wootton, and Arvind Satyanarayan. 2023. Animated Vega-Lite: Unifying Animation with a Grammar of Interactive Graphics. *IEEE Transactions on Visualization and Computer Graphics* 29, 1 (2023), 149–159.

Received October 2023; accepted January 2024