To appear in an IEEE VGTC sponsored conference.

Are LLMs ready for Visualization?

Pere-Pau Vázquez*

ViRVIG Group - UPC, Barcelona

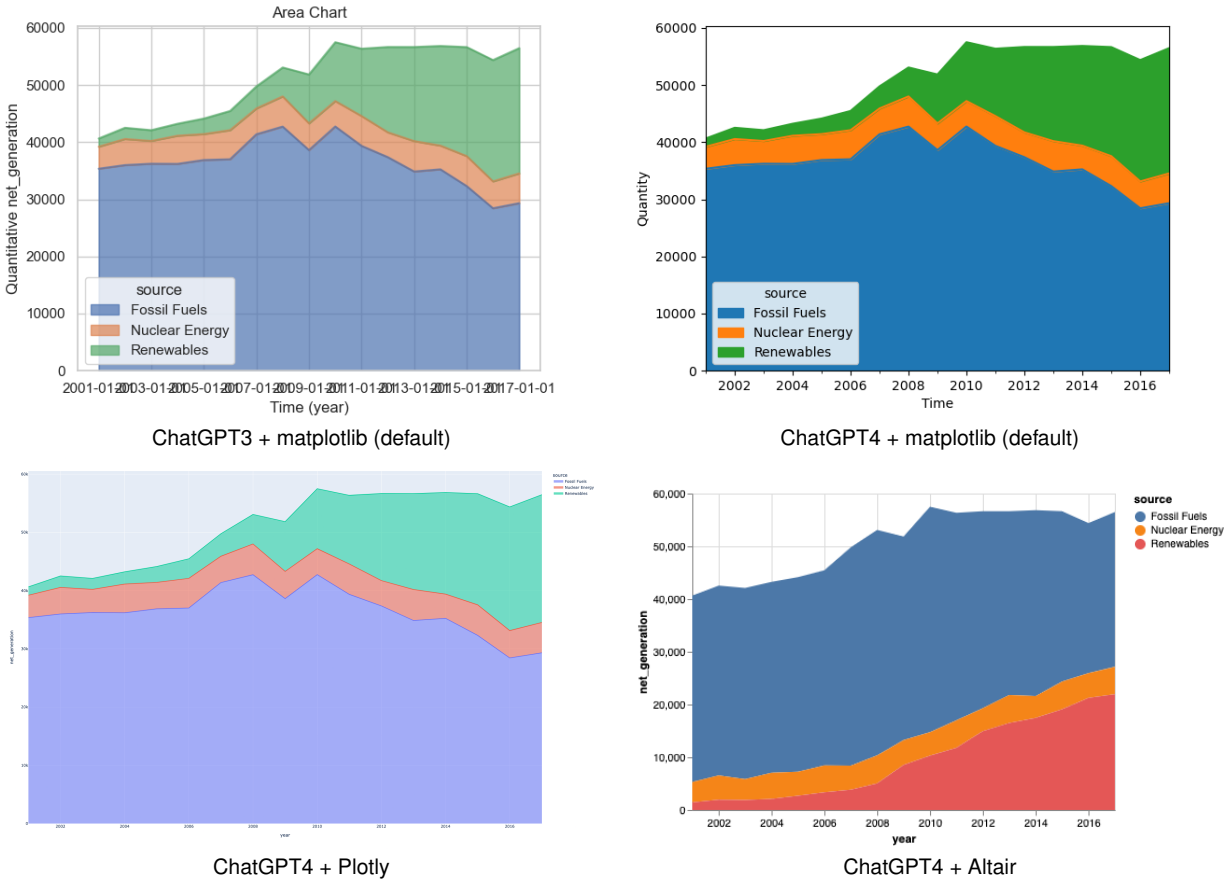


Figure 1: Generation of an area chart using ChatGPT3 and ChatGPT4. The top cases, the prompt did not ask for a concrete library. The bottom examples show the result when asking for a Plotly (left) or Altair (right). All the examples show the area chart, but ChatGPT3 generates labels that overlap. The solutions by ChatGPT4 seem all acceptable. The different configurations (legend placement and font sizes) are left to the libraries, which yield different looks. Also Altair (bottom right) sorts the areas in a different order.

Abstract

Generative models have received a lot of attention in many areas of academia and the industry. Their capabilities span many areas, from the invention of images given a prompt to the generation of concrete code to solve a certain programming issue. These two paradigmatic cases fall within two distinct categories of requirements, ranging from "creativity" to "precision", as characterized by Bing Chat, which employs ChatGPT-4 as its backbone. Visualization practitioners and researchers have wondered to what end one of such systems could accomplish our work in a more efficient way. Several works in the literature have utilized them for the creation of visualizations. And some tools such as Lida, incorporate them as part of their pipeline. Nevertheless, to the authors' knowledge, no systematic approach for testing their capabilities has been published, which includes both extensive and in-depth evaluation. Our goal is to fill that gap with a systematic approach that analyzes three elements: whether Large Language Models are capable of correctly generating a large variety of charts, what libraries they can deal with effectively, and how far we can go to configure individual charts. To achieve this objective, we initially selected a diverse set of charts, which are commonly utilized in data visualization. We then developed a set of generic prompts that could be used to generate them, and analyzed the performance of different LLMs and libraries. The results include both the set of prompts and the data sources, as well as an analysis of the performance with different configurations.

Index Terms: Human-centered computing—Visualization—Visualization techniques; Human-centered computing—Visualization—Empirical studies in visualization

*e-mail: pere.pau.vazquez@upc.edu

1 INTRODUCTION

The generation of visualizations from natural language is not a new approach. For example, in 2001 there was already a first attempt by Cox et al. [3] to create visualizations from given commands. However, the progress, as in many other research fields, has rapidly evolved thanks to the public availability of Large Language Models. Previous research has focused on the creation of rules based on natural language input, and the inputs required a high degree of structure. This has undergone a transformation, particularly during the past decade, owing to the advancements in Natural Language Processing techniques [20]. The typical process for most systems consists of the description of the visualization creation commands in terms of data attributes and task [13]. This is especially useful when tackling data analysis tasks. Such systems are, therefore, primarily developed for visualization professionals. However, users may require a certain visualization technique for the available data. This could be of interest to not only visualization researchers and practitioners, but also to any type of user. It is an especially interesting situation given the increasing number of freely available open datasets. We envision the following scenario: a dataset is discovered on an open data platform, and we aim to generate a variety of visualizations of it, with the user having the ability to select the visualization technique employed. Currently, the obvious choice seems to be to use an LLM to generate such visualization, or the code needed to render it. However, it is unclear to what extent the outcome will be correct, both in terms of syntax and fulfilling the user requirements. This is the topic we intend to explore in this paper.

We began the research by asking ourselves how good LLMs are at creating visualizations from a given dataset. This requires analyzing both the extent of their capabilities, such as the different techniques they can generate, and the depth of configurations that users can add to the desired charts. In this paper, we have conducted a systematic analysis of the capabilities of different LLMs by testing different, somewhat orthogonal features: a) Variety of charts, b) Libraries that can be used, and c) Level of configuration of visual variables.

We contribute to the field in different ways:

- We have developed a prompt test set that can be used with current and future evolutions of LLMs.
- We have analyzed how good are ChatGPT3 [6] and ChatGPT4 [14] in chart generation.
- We have analyzed the performance of ChatGPT4 with different libraries: matplotlib, Plotly, and Altair.
- We have analyzed the capabilities of ChatGPT4 for configuring a limited set of visual variables in the most common chart techniques.

The results are quite positive, with ChatGPT4 being able to create almost 80% of the proposed charts. Furthermore, we have gained valuable insights into the capabilities of the LLMs to work with different libraries and data types. All the prompts, datasets, and resulting code snippets analyzed are publicly available in GitHub[1].

2 RELATED WORK

The problem of using natural language to generate visualizations has been studied for more than 20 years now. The interested reader can refer to the recent surveys by Shen et al. [20] or Wu et al. [27], that provide a comprehensive review of the most relevant developments throughout the last two decades. Cox et al. [3] created one of the first systems intended to generate visualizations from natural language. The field has undergone rapid advancements since the NLP researchers developed word embeddings, [11], which have proven to be a significant advancement in language comprehension. Although rule-based systems can be highly robust, they do not allow a significant amount of flexibility [24]. Other probabilistic grammar systems combine grammar rules and probability distributions to increase flexibility. Nevertheless, a high amount of expertise is necessary to generate such grammars [13].

Many papers or systems focus on the use of natural language for data analysis through visualization. That is, visualization is a means to interpret the data. These systems must understand the user's intention [9] in order to craft the visualizations suitable to help them. As a result, significant efforts have been made to comprehend the analysis question. For example, Gao et al. focus on handling ambiguity in our language [7]. Often, this issue cannot be automatically resolved and may require user intervention. In Eviza [19], the authors address the problem in a task-based fashion, and deal with questions related to the data, such as "where is Texas" in a map. More recently, Narechania et al. adopted a different approach, named NL4DV [13], a system that combines direct questions for visualization generation (e.g., "show me an area chart of how comedy movies evolved over the years") with follow-up questions (e.g., "As a boxplot now"). The system provides the user with the option to select the visualization, which is generated using Vega Lite [18]. However, the number of different available charts is also small. Other modern systems also provide a relatively small number of different visualization techniques [20]. Wang et al. [25] have created a NL framework to interpret the user's intentions in chart creation and edition. They demonstrate their architecture using a tool called VisTalk, which is an extended version of Excel that allows for verbal communication. The authors consider the system as a proof-of-concept, since it was created based on only a few samples. Furthermore, no indication was provided regarding the quantity and variety of charts that are accessible for manipulation.

In fact, the evolution of many of such NL2Vis systems has gone hand in hand with the evolution of Natural Language Processing technologies, as described by Shen et al. in their survey [20]. For example, Chen et al. [2] use BERT [4] to help interpret user inputs and create a domain-specific language for visualizations by fine-tuning a BERT model. Other systems have also taken advantage of LLMs to create visualizations. Two recent examples are Chat2Vis [10] and Lida [5]. In both cases, the user requires an API key of some LLM (ChatGPT3.5, ChatGPT4, LlamaCode ...) to use the systems. Subsequently, these architectures utilize the LLM to interpret the input prompt and generate visualizations. This may pose a problem, since most people do not have a subscription to OpenAI LLMs and smaller LLMs do not produce high-quality results [5]. Kim et al. conducted a very interesting experiment that analyzed the capabilities of ChatGPT of giving answers to visualization questions [8]. Despite the results from the language model are not on par with humans, the authors expressed their optimism in the utility of LLM-guided advice. LLMs have also been used with a different perspective, such as chart explanations, or data exploration. Sultanum and Srinivasan have explored their use to support the authors of data-driven articles in providing explanations [22]. Furthermore, Lida also provides a first data summarizer stage in its pipeline [5], and a code interpreter, as well as visualization explanation, which are powered by LLMs.

In contrast to the systems that respond to user queries, other approaches have also focused on the fully automatic generation and recommendation of charts, based on data, such as [15, 16]. Our goal is to complement all of this research, since we are interested in the user directing the generation of visualizations. To this end, we evaluate how LLMs can be leveraged for data visualization generation, without API keys requirements, and all based on publicly available software.

[1] https://github.com/pere-pau/LLMsForDataVis

3 Overview of the process

To evaluate the capabilities of LLMs, we had to determine the boundaries of our approach. Initially, and as previously mentioned, our objective is to enable the user to specify and delineate the desired outcome. This output could be either an image of a chart, or code. We opted for the second approach due to two factors, namely its utility and its evaluation. Firstly, generating code presents the possibility of modifying the output or using it as part of a more complicated framework. Second, it is easier for us to better understand how the prompt has been understood by the LLM if we further analyze the resulting code. Whether the system generated the appropriate code for reading or manipulating the data can only be answered if we fully inspect the output. We chose Python since it is a language that is expected to be highly represented in LLM training corpuses and has a wide range of visualization libraries.

To create the prompt dataset and subsequent analysis, we performed the following steps:

- Selection of visualization techniques.
- Creation or acquisition of datasets suitable for the techniques.
- Selection of LLMs to analyze.
- Design and fine-tuning of the prompts.
- Exhaustive testing.

3.1 Selection of visualization techniques

The objective of the project was to evaluate the capabilities of LLMs in generating different visualization techniques. Therefore, a task-based approach (asking questions about the data) was inadequate. Instead, we needed to guide the LLMs to generate the techniques of interest. However, first, a set of techniques needed to be defined.

We approached this by using a progressive methodology. We first consulted Google and Google Images for charts. However, this results in mostly repeated charts, such as bar charts, pie charts, and line charts, and very few maps. We checked the Financial Times Visual Vocabulary[2] and Tableau Visual Vocabulary[3]. These were useful to ensure we had a consistent vocabulary, but also showed several, very specific charts such as arc charts (typically shown only for election results), or waterfall charts. We also visited certain specialized websites by conducting a search for "most common charts" (e.g., the now Atlassian-owned chart.io website [4], or Piktochart[5]) and the results were quite limited, since they contained not-so common charts, such as funnel charts, or very specific charts such as Gantt charts, but were lacking in charts such as choropleths, very common in news articles. We also analyzed webpages that depicted relevant charts to analyze from New York Times from 2020[6] and 2023[7].

Finally, we evaluated the available charts in a popular tool, Datawrapper. The set of charts that are available in that location bears resemblance to the previous sets, shares the same vocabulary, and is sufficiently extensive. We discarded multiple charts' configurations and very specific charts, such as the arrow plot or the election donut, and added some widely known charts, such as violin plots. As a result, we ended up with the following list of charts (the names are in alphabetical order, and they also represent the text used to prompt the LLM):

- Area chart: It uses a temporal, a categorical, and a quantitative dimension.
- Bar chart: Common bar charts, although they are assumed to be displayed horizontally in some sites (to distinguish them from vertical bars, that are called column charts). The data consists of a categorical variable and a quantitative one.
- Box plot: The well-known chart to analyze data distributions. It requires a categorical variable and a quantitative one.
- Bubble chart: Classical scatterplot with two quantitative dimensions, and a third one modulating the size of the dots.
- Bullet chart: A bar chart with one or more reference values, typically depicted as a thinner and more opaque bar to indicate the value and a background bar to indicate the reference. It uses a categorical variable, plus two quantitative ones.
- Choropleth: Common map with colors encoding quantitative values for regions. It requires a geojson/json file with the geometry, and another file that contains a categorical data indicating the regions, and a quantitative value.
- Column chart: Typical bar chart. This name is used in contrast to bar to indicate that bars are vertical. Like in the previous case, both a categorical and a quantitative variable are required.
- Donut chart: Similar to a pie chart. It represents data with one category and one quantitative dimension.
- Dot plot: Scatterplots, though the name is often used to indicate that the variable encoded in Y may be discrete. It requires two quantitative values.
- Graduated symbol map: Map that encodes a quantity with a symbol (commonly a circle) that changes its size according to a quantitative variable. Like a choropleth, it uses a geojson/json indicating the regions and another CSV file with a categorical variable and one quantitative value.
- Grouped bar chart: Same as column chart, but assumed that bars are horizontal. In this case, two categories and one quantity are represented.
- Grouped column chart: Same as bar chart, but emphasizing the vertical direction of the bars. Two categories and one quantity.
- Line chart: With temporal, category, and quantitative variables.
- Locator map: Map where data encodes only a position given in latitude and magnitude. The symbol to denote the position can be a circle or a pin. Together with a geojson/json representing the regions, another map indicating the locations in GPS coordinates with a categorical variable (name of the object), together with longitude and latitude are required.
- Pictogram chart: Also known as an isotype chart. It is similar to a quantized bar chart, where the length of the bar is formed by a series of isotype elements. It is encoded through a categorical and a quantitative dimension.
- Pie chart: One category and one value are enough.
- Pyramid chart: Used commonly for population data. Three variables are required, one categorical that indicates the ranges, and two quantitative variables for both the components.
- Radar chart: It can be encoded with an ordered variable, a categorical one, and a quantitative one.

[2] https://www.ft.com/content/c7bb24c9-964d-479f-ba24-03a2b2df6e85

[3] https://www.tableau.com/solutions/gallery/visual-vocabulary

[4] https://chartio.com/learn/charts/essential-chart-types-for-data-visualization/

[5] https://piktochart.com/blog/types-of-graphs/

[6] https://www.nytimes.com/2020/06/10/learning/over-60-new-york-times-graphs-for-students-to-analyze.html

[7] https://www.nytimes.com/2023/07/26/learning/over-75-new-york-times-graphs-for-students-to-analyze.html

- Range plot: A categorical and a quantitative dimension are used. In this case, more than one quantity per category is required.
- Scatterplot: Two quantitative values.
- Stacked bar chart: Like grouped bar charts, but different layout. Two categorical attributes and a quantitative one are necessary.
- Stacked column chart: Identical to the previous one.
- Violin plot: A more complex representation of distributions. It uses one category and one quantitative dimension.
- XY Heatmap chart: Also known as heatmap matrix, or reorderable matrix, as named by Bertin [1]. It uses two categorical (or ordered) dimensions in addition to a quantitative value that is encoded using a color.

These charts include commonly known categories, except for hierarchical or network representations, which are difficult to see in most libraries. Moreover, those charts are very common in scientific literature, despite using sometimes slightly different names. As noted, there are some redundant charts, such as the bar vs column techniques. In data visualization, bar charts are assumed to be vertical (e.g., see [12]), although some Visual Vocabulary guides assume they are horizontal, or that bars refer to both (e.g., [26]). In general, unless otherwise specified, visualization libraries will generate vertical bars for both names.

3.2 Dataset creation

The techniques depicted above have at least three different variables, which can be categorical, quantitative, temporal, or geometrical. In some cases, GPS location is needed, such as in the Locator maps. Hence, we needed specific files for testing the data. We either borrowed or created several files. The following is a description of their contents:

- myfile.csv: Contains five entries of a categorical variable and a quantitative variable. Suitable for simple charts.
- heatmapdataOrig.csv: Stores random values from 1 to 100 in a 50×50 matrix for the heatmap plots.
- iowa-electricity.csv: Sample file from the Vega Datasets, used for line data such as line charts or area charts. A shorter version has been derived for the radar chart.
- mycarsUnique.csv: A synthetic dataset with 53 entries that have two categories (trademark and type of vehicle) and a quantitative column indicating the price. Suitable for grouped and stacked bar charts.
- carsMod.csv: Based on a cars dataset, with multiple quantitative variables for bubble charts.
- myfileDotPlot.csv: A synthetic dataset with a categorical column and two quantitative values, for dot plots.
- population2021.csv: A derived dataset with the population of countries in 2021, with the name, abbreviation, year, and population. It has 274 entries, and it is used for the choropleth and graduated symbol map plots.
- pyramiddata.csv: Synthetic data with a first column indicating age ranges (17 different), and invented values of male and female populations in the next two columns.

In addition, we used two geojson files for the cartography of the world and the US.

3.3 Selection of LLMs

Before starting the analysis, we performed some tests with different LLMs, including ChatGPT3 [6], ChatGPT4 [14], CodeLlama [17], Mixtral, etc. We tested different versions of Llama2 and Code Llama2 (from Meta) and Mixtral (one of the open-source models from Mistral AI [8]), were downloaded from Meta and HuggingFace, respectively. Locally, we tested the 7B parameter models by utilizing both the command line with proper libraries, LM Studio (a tool for easy deployment and testing of LLMs), and Chat2Vis (chat2vis.streamlit.io). Based on our initial experiments, neither Llama2, Code Llama2, nor Mixtral were exhibiting satisfactory performance. On numerous occasions, the generated code displayed errors. It is likely that larger models would perform better, but a high-end machine is required. Instead, ChatGPT3 (running from openai.com) and ChatGPT4 (through bing.com/chat) performed significantly well. As a result, we focused on those.

3.4 Prompts design

To design the prompts, we defined the following requirements:

- Prompts should be simple to reproduce. They should all have a uniform structure: requesting the creation of a certain chart, employing the columns indicated from a CSV file.
- Vocabulary should be similar to the one used by commercial tools (e.g., Tableau), since many practitioners have their own public blogs where they discuss visualization. And that data is likely to be used to train LLM models.
- We would not use the concrete names of columns in the files, but just simple names such as A, B, C and data types would be described (e.g., "A (categorical), and B (quantitative)").

3.5 Testing procedure

For each experiment, we performed the following way:

- Create a fresh new session.
- Input all the prompts in the same session and the same day.
- Copy the output code and edit the names of files and columns accordingly.
- Execute the Python code and analyze the results.

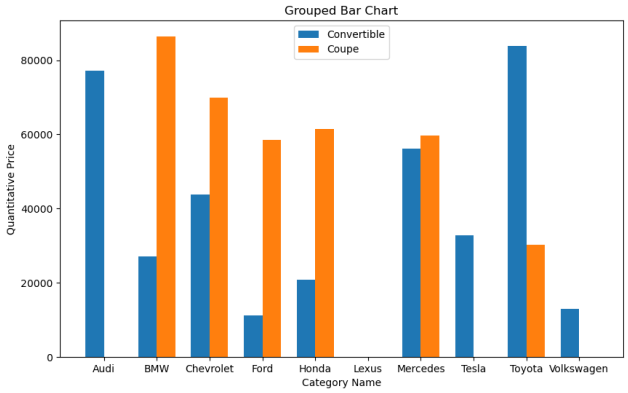
Since LLM chat sessions use the previous prompts as input, it was important to ensure that the session was created as a new one for each experiment. Hence, the vanilla tests, which do not require a specific library, and each library test are executed in distinct sessions. If many queries are generated asking for a chart in Altair and then perform a query without a library requirement, the output will be in Altair, instead of using matplotlib, which is the default library both ChatGPT3 and ChatGPT4 choose when asked for a chart in Python.

Furthermore, we wanted to keep the prompts as similar as possible. That is why we did not use real column names from the files. This ensures that no previous knowledge is used by the LLM. In addition, we specify the types of columns in every prompt to ensure that every query is self-contained.

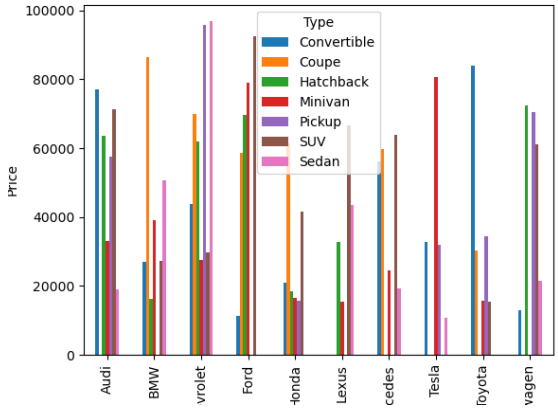
[8] https://mistral.ai/news/announcing-mistral-7b/

Table 1: Analysis of charts that can be generated using ChatGPT3 and ChatGPT4. For this experiment, only the chart was prompted, and no library was imposed. Even though the same libraries are used most of the time, ChatGPT4 generates better outputs. It is noteworthy that ChatGPT4 utilizes seaborn in a lesser number of instances compared to ChatGPT3.

Technique	GPT3	Selected library	GPT4	Selected library
Area chart	✓	matplotlib and seaborn	✓	matplotlib
Bar chart	✓	matplotlib	✓	matplotlib
Box plot	✓	matplotlib and seaborn	✓	matplotlib and seaborn
Bubble chart	✓	matplotlib	✓	matplotlib
Bullet chart	✗	matplotlib	✗	matplotlib
Choropleth	✓	matplotlib	✓	matplotlib
Column chart	✓	matplotlib	✓	matplotlib
Donut chart	✓	matplotlib	✓	matplotlib
Dot plot	✗	matplotlib	✓	matplotlib
Graduated symbol map	✗	matplotlib	✗	matplotlib
Grouped bar chart	✗	matplotlib	✓	matplotlib
Grouped column chart	✗	matplotlib	✓	matplotlib
Line chart	✓	matplotlib and seaborn	✓	matplotlib
Locator map	✓	matplotlib	✓	matplotlib
Pictogram chart	✗	matplotlib	✗	matplotlib
Pie chart	✓	matplotlib	✓	matplotlib
Pyramid chart	✓	matplotlib	✓	matplotlib
Radar chart	✗	matplotlib	✗	matplotlib
Range plot	✗	matplotlib and seaborn	✗	matplotlib
Scatterplot	✓	matplotlib	✓	matplotlib
Stacked bar chart	✓	matplotlib	✓	matplotlib
Stacked column chart	✓	matplotlib	✓	matplotlib
Violin plot	✓	matplotlib	✓	matplotlib and seaborn
XY Heatmap chart	✓	matplotlib and seaborn	✓	matplotlib and seaborn
Total	16 (66.7%)		19 (79.17%)	



ChatGPT3 + matplotlib



ChatGPT4 + matplotlib

Figure 2: Generation of a grouped bar chart using ChatGPT3 and ChatGPT4 with the default configuration. In the first case, the system fails because it incorrectly assumes that only two values will be present per type category. ChatGPT, although it also uses matplotlib by default, properly generates as many bars as values per category. Nonetheless, the code additionally utilizes default values for the majority of the chart, resulting in unintended outcomes such as a legend that overlaps the data and labels that do not fit within the window.

4 Generation of Charts using LLMs

4.1 Default configuration

The first test we carried out was to evaluate how both ChatGPT3 and ChatGPT4 performed in free form, without the specification of the library. The outcomes were quite promising. They are shown in Table 1, together with the libraries the LLM system has selected for the output. The symbol ✓ indicates that the chart was generated correctly, and the symbol ✗ indicates that the chart is either depicting a different technique, or the code has some errors. A more thorough analysis of the issues that may appear even in correct charts is performed in 4.2.

ChatGPT3 was able to generate code for 16 out of the 24 charts, exceeding the 66% of the charts. Some of the problematic cases are relatively surprising. The failure cases for ChatGPT3 are: *a)* bullet charts overlay a point and a bar chart, *b)* for dot plots, it gerenates a connected scatterplot, *c)* graduated symbol maps are drawn as choropleths, *d)* grouped bar charts (both types) do not show all the elements in one category, as shown in Figure 2 (but ChatGPT4 does, using the same library), *e)* the pictogram chart prompt generates a line of circles of different sizes, *f)* radar charts generate wrong

Table 2: Results of the analyzed charts in ChatGPT4 using different libraries. In order to facilitate comparison, the results of the default configuration are provided as the second column. According to the experiment, no significant differences were found between libraries. The number of visualization techniques that are supported remains constant, although some of the techniques that are accurately represented may differ.

Technique	Matplotlib [+seaborn]	Plotly (observations)	Altair (observations)
Area chart	✓	✓	✓
Bar chart	✓ (categories not sorted)	✓ (categories not sorted)	✓
Box plot	✓ (categories not sorted)	✓ (categories not sorted)	✓
Bubble chart	✓ (no legend for size)	✓ (no legend for size, but interactive)	✓
Bullet chart	✗	✗	✓ (with ticks)
Choropleth	✓	✓ (but legend not properly scaled)	✓
Column chart	✓ (categories not sorted)	✓ (categories not sorted)	✓
Donut chart	✓ (categories not sorted)	✓ (categories not sorted)	✓ (quantities on hover)
Dot plot	✓ (Y axis not sorted)	✓ (Y axis not sorted)	✓
Graduated symbol map	✗ (generates choropleth)	✓	✗ (expects centroids in data)
Grouped bar chart	✓ (clipped labels, legend overlaps)	✓	✓
Grouped column chart	✓ (clipped labels, legend overlaps)	✓	✓ (horizontal array of horizontal bar charts)
Line chart	✓	✓	✓
Locator map	✓	✓	✓ (map inverted in Y)
Pictogram chart	✗ (generates scatterplot)	✗ (not supported)	✗ (not supported)
Pie chart	✓ (labels not sorted)	✓ (labels not sorted)	✓ (labels sorted)
Pyramid chart	✓ (legend overlaps)	✗ (two pairs of bars)	✗ (two bar charts side to side)
Radar chart	✗ (can be fixed manually)	✗ (expects angles)	✗ (not supported)
Range plot	✗ (expects different data)	✗ (generates connected scatterplot)	✗ (expects different data)
Scatterplot	✓	✓	✓
Stacked bar chart	✓ (clipped labels, legend overlaps)	✓	✓
Stacked column chart	✓ (clipped labels, legend overlaps)	✓	✓ (generates horizontal stacked bars)
Violin plot	✓ (categories not sorted)	✓ (categories not sorted, points outside)	✓
XY Heatmap chart	✓ (Y axis inverted)	✓ (Y axis inverted)	✓ (Y axis inverted)
Total	19 (79.17%)	19 (79.17%)	19 (79.17%)

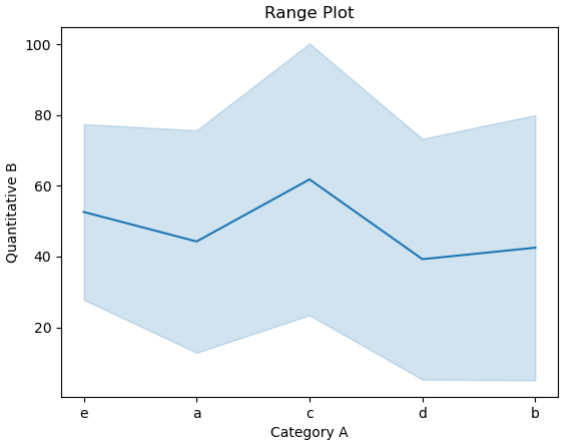


Figure 3: Incorrect generation of a range plot by ChatGPT3. In this case, the chart is displayed, but it does not correspond to a range plot.

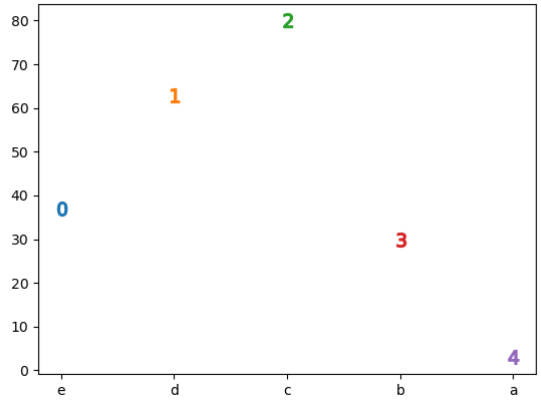


Figure 4: Incorrect generation of a dot plot by ChatGPT4. The code executes, but the output resembles a scatterplot with numbers.

code, and *g)* range plots generate a chart that resembles a line chart with a confidence interval (see Figure 3). ChatGPT4 works properly, although both use the same library.

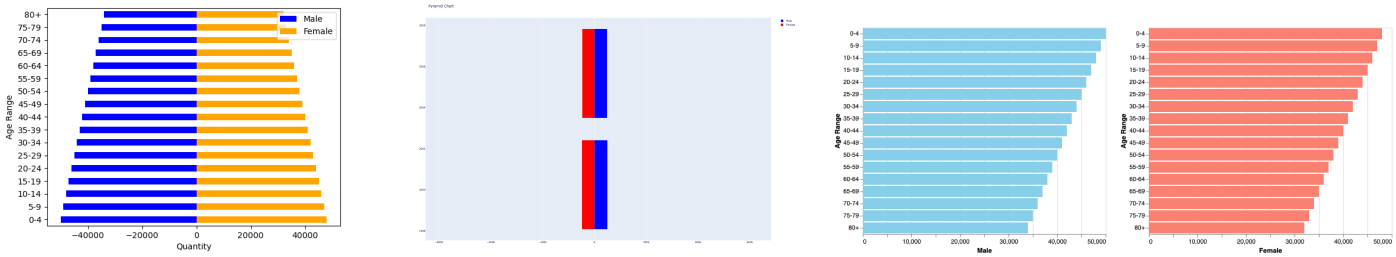
ChatGPT4, on the other hand, exhibits excellent performance in 19 out of the 24 distinct charts. The majority of issues arise from charts that necessitate combining with other charts, such as the range plot, which can be constructed by overlaying a line chart with a scatterplot or a line chart with circular marks at the data points. More concretely, the failure cases are: *a)* bullet chart, where the output is a small multiples of pairs of horizontal bars one on top of the other, without width or opacity changes, *b)* the graduated symbol map generates a choropleth, *c)* the pictogram chart, that outputs a dot plot with labels (as shown in Figure 4), *d)* the radar and range plot charts exhibit errors in the code. In the first case, the issue can be resolved manually by modifying two references to columns. Nonetheless, in the second case, the data is expected in a more distinct format.

4.2 Library tests

The initial test indicated that ChatGPT generates charts, preferably using matplotlib. Nevertheless, there are many other popular Python

ChatGPT4 + matplotlib ChatGPT4 + Plotly ChatGPT4 + Altair

Figure 5: Generation of a bullet chart using the ages' dataset. The prompt specifies the column B as the value, and the column C as the second value. From the three prompts, only the one using Altair seems to succeed. The default configuration (left) renders two bars, but with the same size and opacity, thus making it difficult to understand whether it is a stack bar or two similar values are represented. Plotly generates something not very similar to a bullet chart, while Altair (right) uses a tick to mark the reference value.



ChatGPT4 + matplotlib ChatGPT4 + Plotly ChatGPT4 + Altair

Figure 6: Generation of a pyramid chart using the ages' dataset. From the three prompts, only the one using the default ChatGPT4 configuration (left) works properly. Plotly generates something really awkward, while Altair (right) displays to side-by-side bar charts.

libraries that can be used, such as bokeh, Plotlib, ggplot, Altair, and echarts. For the next test, we chose a well established library, Plotly [21], and a relatively new one, Altair [23]. Altair is a Python library that generates Vega Lite code [18]. Both libraries offer functionalities for interactive chart exploration; however, we shall solely examine the capabilities of generating static charts.

To generate charts using a specific library, we added "with Plotly" or "with Altair" to the prompt. In all cases, the generated Python code used the desired libraries correctly. When imposing a library through the prompt, the results obtained with ChatGPT4 are similar to the previous ones. The number of charts that are now made correctly is the same for both libraries, 19 out of 24. However, there are some changes to the charts that are generated correctly and others that are coded incorrectly. As this experiment is more closely aligned with the requirements of both researchers and practitioners, we conduct a thorough analysis of the outcomes. Table 2 shows the results of the analysis. A check ✓ symbol indicates that the chart is correctly generated, despite it may have some presentation issues such as labels not fitting the window, or the legend overlapping the data. Most of these issues arise from the automation part of the library. Consequently, they are not considered to be failure cases. However, we provide a detailed account of the pertinent aspects that were discovered. The cross ✗ symbol indicates that the LLM did not produce the desired chart successfully. Either it produced a different chart, or the code breaks. Once again, the respective observation columns provide descriptions of the issues discovered.

By analyzing Table 2, we can see that the behavior of different libraries is quite similar. There are some cases where they do not yield the same results, as in the bullet charts, illustrated in Figure 5. In this case, Altair seems to be the only library that performs as expected, since matplotlib and Plotly generate some output that does not correspond to the required chart. Another complex case is the one of pyramid charts. We expected better results, since population charts are quite common. However, only matplotlib performs well, as shown in Figure 6.

By examining the code, we saw that most of the chart configuration (legend position, sizes of fonts, etc.) were left to the library. As a result, on numerous occasions, this generates issues with the result, such as legends overlapping the data, or too tiny fonts. We succinctly described those issues in the respective observations columns.

4.3 Visual variables configuration

To complete the experiments, we decided to include some prompts that, along with the chart creation, implied some modifications of the visual variables. For this experiment, we selected a subset of plots: bar charts, line charts, scatterplots/bubble charts and pie charts.

Table 3: Visual variables configurations using ChatGPT4 and the default library. The experiments indicate that certain visual variables appear to be more straightforward to configure than others. It is somewhat surprising that numerous attempts to modify the default configuration of scatterplots and bubble charts were unsuccessful. The proposed changes are quite common. However, the LLM system might have issues when those changes involve numerous columns, as it happens in those cases. The other techniques performed very well, and the only change that did not work was the definition of the sector colors in a pie chart.

Technique	Prompt	Result	Observations
Bar chart	with horizontal bars	✓	
Bar chart	with bars in light green	✓	
Bar chart	with bars of width 120 pixels	✓	
Bar chart	with the bars of different colors	✓	Continuous palette
Bar chart	with title "This is a bar chart generated by an LLM"	✓	
Bar chart	with labels indicating the quantities on top of the bars	✓	
Line chart	with all lines in purple	✓	
Line chart	with lines of width 20 pixels	✓	
Line chart	with the opacity of the lines as 0.5	✓	
Line chart	marking the data points with circles	✓	
Line chart	with green lines and the data points encoded with purple squares	✓	
Line chart	with dashed lines	✓	
Pie chart	sorting the values from larger to smaller	✓	
Pie chart	sorting the values from smaller to larger	✓	
Pie chart	with the labels in boldface	✓	
Pie chart	with the title "Pie"	✓	
Pie chart	with a sequential color palette that depends on the B column	✗	
Pie chart	sorting the values from larger to smaller, clockwise, and starting at 90 degrees	✓	
Pie chart	sorting the values from larger to smaller, clockwise, and starting at 90 degrees, and not displaying the categorical labels, only the quantities, and outside the pie	✓ ✓	
Bubble chart	with points as triangles, whose angle depends on column D	✗	Error
Bubble chart	with the shape of points depending on column B	✗	No points, not enough categories
Scatterplot	with the shape of points depending on column B and their size dependent on column D with the opacity encoding the values in E column	✗	No points
Scatterplot	using columns C and D for X and Y axis and the size defined by the A column	✗	Big blue square
Scatterplot	with the shape of points depending on column B and their size dependent on column D, and a size of 20 pixels	✗	No points
Total		19 (76%)	

To generate the chart, we modified the prompt with instructions to change some default parameters. We tested both changing visual variables based on fixed values (e.g., a certain width or opacity) and making some variables dependent on other columns. These changes refer to visual aspects of the marks, including direction, size, width, color, shape, opacity, stroke, ordering of the marks. In addition, we also introduced changes to the annotations: extra labels, their position, the font (boldface), and title of the chart.

For each chart type, a different set of configurations was tested, based on common modifications we all made to our depictions. In Table 3, we show the modification prompt that was included in the chart creation, as well as the results. As it can be seen, most of the changes were successful: the code generated without errors, and the results were satisfactory. A notable exception was in the case of scatterplots and their variations. Most of the configurations generated either incorrect code or blank plots. We hypothesize that this might be due to the large number of variables included in the prompt. Since these plots were the ones that required a large number of visual variables' configuration changes dependent on input data.

Some examples of successful prompts are shown in Figure 7 for the lines chart. Those prompts indicated changes in the marks, adding points as circles (left), adding points as squares with a different color (center), and changing the lines to dashes (right). As said, scatterplots and bubble charts are likely to fail when configuring multiple variables, as shown in Figure 8.

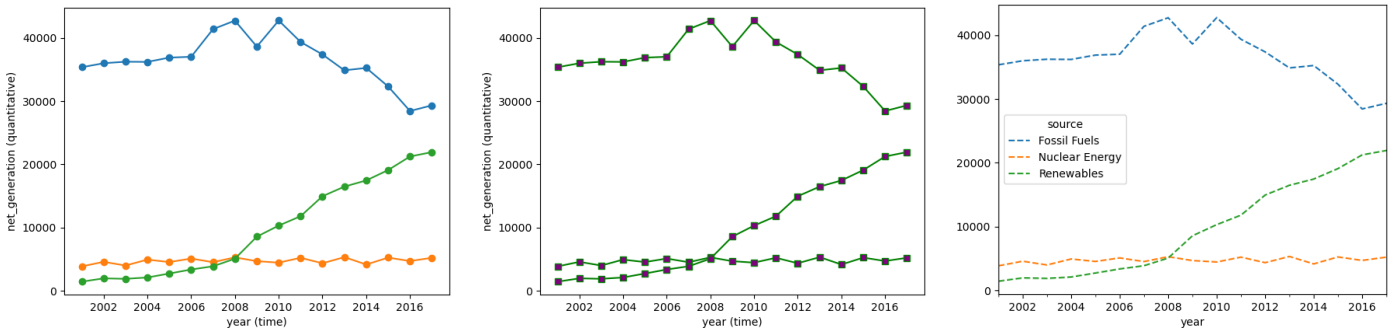
5 Analysis of results

The obtained results were very positive, and actually exceeded the author's expectations. We purposefully crafted single-prompt experiments, devoid of any subsequent inquiries. Even in the visual variables' configuration tests, the prompt was simple. This strategy is based on our belief that future LLMs will work well with a limited amount of input information. Furthermore, we believe that most users will expect a proper solution the first time. Hence, we sought to comprehend the extent of the capabilities of LLMs in those circumstances. Furthermore, we understand that follow-up prompts may improve the quality of the charts, as several papers have demonstrated. But we do not believe that the future for the users is to become prompt engineers. We are confident that advancements in LLM technologies will facilitate a more natural description of the output we aim to obtain from them.

We also conducted further testing to see how the stochastic nature of LLMs may affect the results. For all the charts, we ran a test involving five distinct executions of the prompt using ChatGPT4, utilizing newly generated sessions for each prompt. In the subsequent executions, the charts that give errors are still not correct, with one exception: the bullet chart, which is generated correctly on one occasion. The rest of incorrect charts, were in all instances. For the charts that were initially correct, they are generated properly the rest of the executions, with the exception of the grouped bar chart, that is drawn wrong (inverted Y axis) in one out of the 5 attempts. Despite the charts getting the geometry correct, in three instances of different charts, a legend with a color palette that was unrelated to the representation was also incorporated to the code.

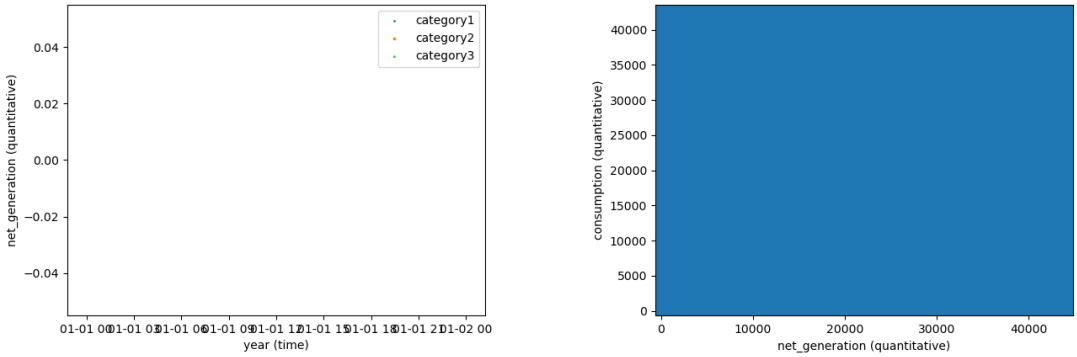
Besides, the experiments were performed on general LLMs. In the future, we expect that dedicated LLMs will also be broadly available, since fine-tuning for different tasks is a common step in neural model training. Thus, the results are only going to improve.

The tests that we conducted were limited to make them manageable. There were various areas where we could have extended the



Adding points | Adding points as colored squares | Dashed lines

Figure 7: Configuring visual variables in line charts.



Shape depending on column B | Using columns C and D for X and Y axes and A for the size

Figure 8: Failure cases for scatterplot (left) with shape defined as a variable and bubble chart (right) with multiple visual variables defined.

resulting dataset. Initially, we restricted ourselves to predetermined prompts for each chart. Subsequently, the final prompts emerged from diverse experiments. Although some LLMs may have slightly better versions of such prompts, we do not expect any significant changes. We also limited ourselves in the number of techniques: as explained above, after analyzing both Financial Times and Tableau Visual Vocabulary, we found that some techniques are highly uncommon in newspapers and articles, and several require special data manipulation that is not simple for a non-specialist. Therefore, we finally took the simple charts available on Datawrapper.io and added some additional ones that were not there. Consequently, more precise techniques such as Sankey diagrams are absent, and even though they can be properly generated by LLMs, we believed that the sample was sufficiently representative. We also did not include interaction capabilities in the analysis. Again, this is something we have experimented with, using LLMs, but, describing interaction verbally is difficult, and believe that non-professionals might find it difficult to express naturally.

6 Conclusions and Future Work

In this paper, we evaluated the capabilities of modern LLM systems to generate Python code that renders different visualization techniques. We have selected a total of 24 different techniques that are commonly used in research papers and infographics. We also designed a set of prompts to generate different techniques based on the vocabulary commonly used by professionals in the production or media industries. After conducting some tests, we decided to discard some publicly available 7B parameter models and opt for the public models of ChatGPT3 and ChatGPT4. We further analyzed the capabilities of generating Python code for different libraries by systematically evaluating these two different LLM models. We have decided to request the LLM systems to generate Python code instead of relying solely on visualization generation, as this approach enables us to gain a more profound understanding of the systems' output. Especially in the case of wrong code, we have had the possibility of correcting it or getting insights on the problems.

The results obtained were a positive surprise. Both the high number of charts that were properly built, and the fact that some of them are not straightforward, such as the maps, that need to combine spatial information with data. This makes us feel optimistic regarding the near future. However, there are still several shortcomings that cannot be solved easily by non-experts, such as the charts where labels are clipped, or the legends overlap the data. These require either prompt fine-tuning or code editing. Therefore, we believe that the present output may serve as a suitable starting point for users possessing programming or visualization expertise. However, end-to-end, from prompt to visualization flows, still do not achieve flawless results in numerous instances.

In the future, we would like to include more complex visualizations in the test dataset, such as using combinations of charts, interactions, etc. Additionally, we would like to include more publicly available LLMs. Nonetheless, the tests conducted with 7B parameter models such as LlamaCode or Mixtral yielded outcomes that were not comparable (in fact, significantly inferior) to those obtained with ChatGPT3 or ChatGPT4. We also considered utilizing a JavaScript library, such as D3, but ultimately decided that the present selection was more suitable, given the substantial number of programmers who utilize Python and its relatively smooth learning curve in comparison to JavaScript.

ACKNOWLEDGMENTS

The author would like to thank the anonymous reviewers for their fruitful comments. Supported by Ministerio de Ciencia e Innovación/AEI (PID2021-122136OB-C21 by 10.13039/501100011033/FEDER, UE).

REFERENCES

[1] J. Bertin. Sémiologie graphique: Les diagrammes-les réseaux-les cartes. Technical report, Gauthier-VillarsMouton & Cie, 1973.

[2] Q. Chen, S. Pailoor, C. Barnaby, A. Criswell, C. Wang, G. Durrett, and I. Dillig. Type-directed synthesis of visualizations from natural language queries. *Proceedings of the ACM on Programming Languages*, 6(OOPSLA2):532–559, 2022.

[3] K. Cox, R. E. Grinter, S. L. Hibino, L. J. Jagadeesan, and D. Mantilla. A multi-modal natural language interface to an information visualization environment. *International Journal of Speech Technology*, 4:297–314, 2001.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[5] V. C. Dibia. Lida: A tool for automatic generation of grammar-agnostic visualizations and infographics using large language models. *Annual Meeting of the Association for Computational Linguistics*, 2023. doi: 10.48550/arxiv.2303.02927

[6] L. Floridi and M. Chiriatti. Gpt-3: Its nature, scope, limits, and consequences. *Minds and Machines*, 30:681–694, 2020.

[7] T. Gao, M. Dontcheva, E. Adar, Z. Liu, and K. G. Karahalios. Datatone: Managing ambiguity in natural language interfaces for data visualization. In *Proceedings of the 28th annual acm symposium on user interface software & technology*, pp. 489–500, 2015.

[8] N. W. Kim, G. Myers, and B. Bach. How good is chatgpt in giving advice on your visualization design? *arXiv preprint arXiv:2310.09617*, 2023.

[9] B. Lee, P. Isenberg, N. H. Riche, and S. Carpendale. Beyond mouse and keyboard: Expanding design considerations for information visualization interactions. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2689–2698, 2012.

[10] P. Maddigan and T. Susnjak. Chat2vis: Generating data visualizations via natural language using chatgpt, codex and gpt-3 large language models. *IEEE Access*, 2023. doi: 10.1109/access.2023.3274199

[11] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[12] T. Munzner. *Visualization analysis and design*. CRC press, 2014.

[13] A. Narechania, A. Srinivasan, and J. Stasko. Nl4dv: A toolkit for generating analytic specifications for data visualization from natural language queries. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):369–379, 2020.

[14] OpenAI, :, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, R. Avila, I. Babuschkin, S. Balaji, V. Balcom, P. Baltescu, H. Bao, M. Bavarian, J. Belgum, I. Bello, J. Berdine, G. Bernadett-Shapiro, C. Berner, L. Bogdonoff, O. Boiko, M. Boyd, A.-L. Brakman, G. Brockman, T. Brooks, M. Brundage, K. Button, T. Cai, R. Campbell, A. Cann, B. Carey, C. Carlson, R. Carmichael, B. Chan, C. Chang, F. Chantzis, D. Chen, S. Chen, R. Chen, J. Chen, M. Chen, B. Chess, C. Cho, C. Chu, H. W. Chung, D. Cummings, J. Currier, Y. Dai, C. Decareaux, T. Degry, N. Deutsch, D. Deville, A. Dhar, D. Dohan, S. Dowling, S. Dunning, A. Ecoffet, A. Eleti, T. Eloundou, D. Farhi, L. Fedus, N. Felix, S. P. Fishman, J. Forte, I. Fulford, L. Gao, E. Georges, C. Gibson, V. Goel, T. Gogineni, G. Goh, R. Gontijo-Lopes, J. Gordon, M. Grafstein, S. Gray, R. Greene, J. Gross, S. S. Gu, Y. Guo, C. Hallacy, J. Han, J. Harris, Y. He, M. Heaton, J. Heidecke, C. Hesse, A. Hickey, W. Hickey, P. Hoeschele, B. Houghton, K. Hsu, S. Hu, X. Hu, J. Huizinga, S. Jain, S. Jain, J. Jang, A. Jiang, R. Jiang, H. Jin, D. Jin, S. Jomoto, B. Jonn, H. Jun, T. Kaftan, Łukasz Kaiser, A. Kamali, I. Kanitscheider, N. S. Keskar, T. Khan, L. Kilpatrick, J. W. Kim, C. Kim, Y. Kim, H. Kirchner, J. Kiros, M. Knight, D. Kokotajlo, Łukasz Kondraciuk, A. Kondrich, A. Konstantinidis, K. Kosic, G. Krueger, V. Kuo, M. Lampe, I. Lan, T. Lee, J. Leike, J. Leung, D. Levy, C. M. Li, R. Lim, M. Lin, S. Lin, M. Litwin, T. Lopez, R. Lowe, P. Lue, A. Makanju, K. Malfacini, S. Manning, T. Markov, Y. Markovski, B. Martin, K. Mayer, A. Mayne, B. McGrew, S. M. McKinney, C. McLeavey, P. McMillan, J. McNeil, D. Medina, A. Mehta, J. Menick, L. Metz, A. Mishchenko, P. Mishkin, V. Monaco, E. Morikawa, D. Mossing, T. Mu, M. Murati, O. Murk, D. Mély, A. Nair, R. Nakano, R. Nayak, A. Neelakantan, R. Ngo, H. Noh, L. Ouyang, C. O'Keefe, J. Pachocki, A. Paino, J. Palermo, A. Pantuliano, G. Parascandolo, J. Parish, E. Parparita, A. Passos, M. Pavlov, A. Peng, A. Perelman, F. de Avila Belbute Peres, M. Petrov, H. P. de Oliveira Pinto, Michael, Pokorny, M. Pokrass, V. Pong, T. Powell, A. Power, B. Power, E. Proehl, R. Puri, A. Radford, J. Rae, A. Ramesh, C. Raymond, F. Real, K. Rimbach, C. Ross, B. Rotsted, H. Roussez, N. Ryder, M. Saltarelli, T. Sanders, S. Santurkar, G. Sastry, H. Schmidt, D. Schnurr, J. Schulman, D. Selsam, K. Sheppard, T. Sherbakov, J. Shieh, S. Shoker, P. Shyam, S. Sidor, E. Sigler, M. Simens, J. Sitkin, K. Slama, I. Sohl, B. Sokolowsky, Y. Song, N. Staudacher, F. P. Such, N. Summers, I. Sutskever, J. Tang, N. Tezak, M. Thompson, P. Tillet, A. Tootoonchian, E. Tseng, P. Tuggle, N. Turley, J. Tworek, J. F. C. Uribe, A. Vallone, A. Vijayvergiya, C. Voss, C. Wainwright, J. J. Wang, A. Wang, B. Wang, J. Ward, J. Wei, C. Weinmann, A. Welihinda, P. Welinder, J. Weng, L. Weng, M. Wiethoff, D. Willner, C. Winter, S. Wolrich, H. Wong, L. Workman, S. Wu, J. Wu, M. Wu, K. Xiao, T. Xu, S. Yoo, K. Yu, Q. Yuan, W. Zaremba, R. Zellers, C. Zhang, M. Zhang, S. Zhao, T. Zheng, J. Zhuang, W. Zhuk, and B. Zoph. Gpt-4 technical report, 2023.

[15] X. Qin, Y. Luo, N. Tang, and G. Li. Deepeye: An automatic big data visualization framework. *null*, 2018. doi: 10.26599/bdma.2018.9020007

[16] P. Ren, Z. Mao, S. Li, Y. Xiao, Y. Ke, L. Yao, H. Lan, X. Li, M. Sheng, M. Sheng, Y. Zhang, and Y. Zhang. Intelligent visualization system for big multi-source medical data based on data lake. *null*, 2021. doi: 10.1007/978-3-030-87571-8_61

[17] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

[18] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics*, 23(1):341–350, 2016.

[19] V. Setlur, S. E. Battersby, M. Tory, R. Gossweiler, and A. X. Chang. Eviza: A natural language interface for visual analysis. In *Proceedings of the 29th annual symposium on user interface software and technology*, pp. 365–377, 2016.

[20] L. Shen, E. Shen, Y. Luo, X. Yang, X. Hu, X. Zhang, Z. Tai, and J. Wang. Towards natural language interfaces for data visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 2022. doi: 10.1109/tvcg.2022.3148007

[21] C. Sievert. *Interactive web-based data visualization with R, plotly, and shiny*. CRC Press, 2020.

[22] N. Sultanum and A. Srinivasan. Datatales: Investigating the use of large language models for authoring data-driven articles. *arXiv.org*, 2023. doi: 10.48550/arxiv.2308.04076

[23] J. VanderPlas, B. Granger, J. Heer, D. Moritz, K. Wongsuphasawat, A. Satyanarayan, E. Lees, I. Timofeev, B. Welsh, and S. Sievert. Altair: interactive statistical visualizations for python. *Journal of open source software*, 3(32):1057, 2018.

[24] H. Voigt, M. Meuschke, K. Lawonn, and S. Zarrieß. Challenges in designing natural language interfaces for complex visual models. In *Proceedings of the First Workshop on Bridging Human–Computer Interaction and Natural Language Processing*, pp. 66–73, 2021.

[25] Y. Wang, Z. Hou, Z. Hou, L. Shen, T. Wu, J. Wang, J. Wang, H. Huang, H. Zhang, and D. Zhang. Towards natural language-based visualization authoring. *IEEE Transactions on Visualization and Computer Graphics*, 2022. doi: 10.1109/tvcg.2022.3209357

[26] C. O. Wilke. *Fundamentals of data visualization: a primer on making informative and compelling figures*. O'Reilly Media, 2019.

[27] A. Wu, Y. Wang, X. Shu, D. Moritz, W. Cui, W. Cui, H. Zhang, D. Zhang, D. Zhang, and H. Qu. Ai4vis: Survey on artificial intelligence approaches for data visualization. *arXiv: Human-Computer Interaction*, 2021.