Eurographics Conference on Visualization (EuroVis) 2025
W. Aigner, N. Andrienko, and B. Wang
(Guest Editors)

Volume 44 (2025), Number 3

Ask Humans or AI? Exploring Their Roles in Visualization Troubleshooting

Shuyu Shen[1], Sirong Lu[1], Leixian Shen[2], Zhonghua Sheng[2], Nan Tang[1,2], Yuyu Luo[1,2]

[1]The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China
[2]The Hong Kong University of Science and Technology, Hong Kong SAR, China

Supplementary Materials: https://github.com/HKUSTDial/vistroubleshooting.github.io/

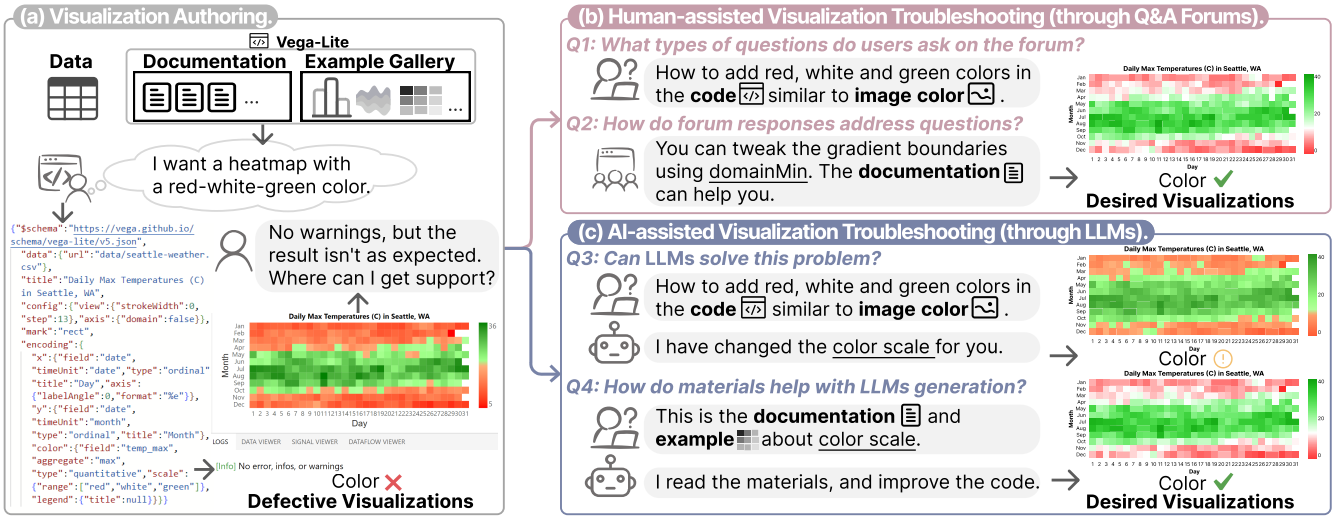

Figure 1: *Comparison of human-assisted and AI-assisted troubleshooting in visualization authoring. Given a user-identified issue in the defective visualization, experienced peers in forums can provide personalized, context-aware advice but may have varying response quality and delays. LLMs can offer immediate feedback but may introduce inaccuracies and require additional contextual guidance for reliability.*

Abstract

Visualization authoring is an iterative process requiring users to modify parameters like color schemes and data transformations to achieve desired aesthetics and effectively convey insights. Due to the complexity of these adjustments, users often create defective visualizations and require troubleshooting support. In this paper, we examine two primary approaches for visualization troubleshooting: (1) Human-assisted support via forums, where users receive advice from other individuals, and (2) AI-assisted support using large language models (LLMs). Our goal is to understand the strengths and limitations of each approach in supporting visualization troubleshooting tasks. To this end, we collected 889 Vega-Lite cases from Stack Overflow. We then conducted a comprehensive analysis to understand the types of questions users ask, the effectiveness of human and AI guidance, and the impact of supplementary resources, such as documentation and examples, on troubleshooting outcomes. Our findings reveal a striking contrast between human- and AI-assisted troubleshooting: Human-assisted troubleshooting provides tailored, context-sensitive advice but often varies in response quality, while AI-assisted troubleshooting offers rapid feedback but often requires additional contextual resources to achieve desired results.

1. Introduction

Visualization authoring is an iterative and challenging process, as users often need to modify intermediate visualization results to achieve their desired aesthetics and functionality [LLF*24]. This process requires adjusting parameters such as color schemes, scales, and layouts to convey data insights effectively [LQTL18, XLLT24]. In programming-based authoring tools, such as Vega-Lite [SMWH17], users define visualizations through declarative

grammar. For example, as shown in Figure 1(a), a user employs the Vega-Lite to create a heatmap but struggles to adjust the color scheme to a specific red-white-green gradient. Despite thoroughly consulting the Vega-Lite documentation and example gallery, the user may still not be able to resolve this issue independently. Given the complexity of these adjustments, users often seek external help to troubleshoot issues and refine their visualizations [SLR*19].

Human-assisted Visualization Troubleshooting. Traditionally, when users encounter challenging issues in visualization authoring, they often turn to online Q&A forums, such as Stack Overflow, to seek help [BFW22]. These forums allow users to receive personalized, context-aware advice from experienced peers. This approach often provides valuable insights and practical solutions tailored to the specific question. However, the quality of responses can vary widely depending on the expertise of the responders, and users may experience delays before receiving effective feedback. Given this context, we explore the role of human-assisted visualization troubleshooting by addressing two key research questions:

- ***Q1****: What questions do users commonly ask in the Q&A forum?* Specifically, we aim to categorize the common challenges users face, such as syntax errors or design-related issues, to better understand the gaps between user-created defective visualizations and their desired outcomes.
- ***Q2****: How effectively do human responses in forums address users' questions?* Specifically, we investigate the accuracy and practicality of human responses, evaluating whether the solutions adequately resolve users' issues.

AI-assisted Visualization Troubleshooting. The advent of Large Language Models (LLMs) provides users with a new option for troubleshooting support [ZDL*24, YHH*24]. By prompting LLMs, users can receive immediate feedback to address defects in visualizations. However, LLMs are prone to generating incorrect or irrelevant solutions due to hallucination issues [ZXY*24]. In addition, LLMs often require additional contextual guidance, such as external documentation or examples, to better interpret the user's intent and provide reliable answers [CZX*24, WYS*24]. To understand the potential of LLMs in this task, we ask the following questions:

- ***Q3****: To what extent can LLMs provide comparable or improved troubleshooting support for the same questions?* We evaluate whether LLMs can match or surpass the quality of human responses in Q&A forums for visualization troubleshooting tasks.
- ***Q4****: How do supplementary resources, such as documentation and example galleries, impact the effectiveness of LLMs assistance?* We explore how the integration of external resources influences the quality of LLM-based solutions.

Contributions. Our contributions are summarized as follows:

- **Dataset Curation.** We collected 889 Vega-Lite cases from Stack Overflow, including 288 troubleshooting cases, with 47 cases featuring supplementary resources and gold-standard solutions.
- **Empirical Study.** We conducted a detailed analysis of Q&A forums, focusing on question classification, the role of supplementary resources, code complexity, and operation classification. Our study systematically compared human-assisted and LLM-assisted troubleshooting using human-defined score metrics.
- **Key Findings.** We identified key practices and limitations in both human- and AI-assisted solutions, emphasizing the importance of supplementary materials in enhancing troubleshooting effectiveness. These findings offer insights to improve user manual designs and optimize LLM performance in this scenario.
- **Open Problem.** Our research raises key questions for future exploration, including how to better structure tutorials to meet the diverse needs of users. In addition, we highlight the potential of advanced LLM-driven solutions, such as Retrieval-Augmented Generation (RAG), to bridge the gap between user-specific troubleshooting issues and task-specific resources.

2. Related Work

2.1. Programming-based Visualization Authoring

Programming-based visualization authoring methods can generally be categorized into two types: imperative languages and declarative grammars. The former provides a wide range of customizable design primitives for creating expressive visualizations but tend to be cumbersome and require users to be proficient in programming. Processing [RF06], ProtoVis [BH09], and D3 [BOH11] are prominent examples of this paradigm. Declarative grammar provides varying-level expressiveness while closing programming concepts to users. Low-level grammar (*e.g.,* Vega [SRHH16]) enables highly flexible and expressive visualization authoring but requires users to specify all mapping elements. High-level grammar (*e.g.,* Vega-Lite [SMWH17], ggplot2 [Wic10], VizQL [STH02], ZQL [SKL*16], Atlas [LCMZ21], and Atom [PDFE18]) provides concise frameworks to support the rapid construction of interactive charts in part by omitting details of visualization construction.

Compared with imperative languages, declarative approaches trade fine-grained control over concise and expressivity. Besides, many declarative languages are JSON-specified [SRHH16, SMWH17, PDFE18, McN23], making them intuitive to understand. Therefore, in this paper, we choose declarative programming approaches, Vega-Lite [SMWH17], as our specification language, due to its comprehensive ecosystem and widespread adoption in visualization authoring [SST*22, CSX*22, MC21, YSL*24, SLW*24].

2.2. LLM-based Visualization Authoring

The advancement of LLMs has introduced a new paradigm in visualization authoring, allowing users to describe visualizations in natural language and automatically generate them [SSL*23, WYS*24, WWZ*24, WHT*24, CH24, BS24, LLC*24, LSL*24]. For example, LIDA [Dib23] leverages LLMs to create grammar-agnostic visualizations and infographics, ChartGPT [TCD*24] generates visual charts from abstract natural language inputs, and LLM4Vis [WZW*23] further utilizes LLMs to produce interpretable visualizations. DracoGPT [WGBH24] has explored visualization design preferences of LLMs, using these preferences to guide and constrain recommended visualizations. The emergence of LLMs has significantly reduced the barrier for users without specialized visualization knowledge. Compared to prior work [LTL*21c, WHS*23, LQT*18], LLM-based approaches demonstrate stronger capabilities in understanding user intent.

However, LLMs often produce defective visualizations that do

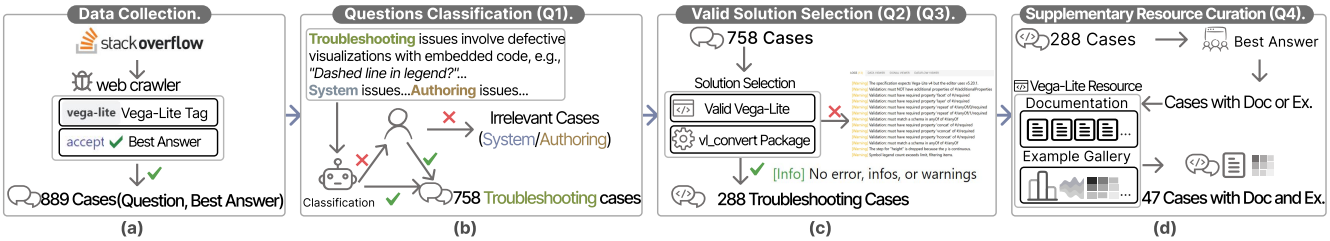

Figure 2: *The data curation process, designed to address the four research questions (Q1-Q4), involves four main steps: (a) Data collection, (b) Question classification (for Q1), (c) Valid solution selection (for Q2 and Q3), and (d) Supplementary resource curation (for Q4).*

not align with user expectations, prompting research into designing evaluation frameworks to assess these visualizations [CH24, CZX*24, LTL21a, SNL*21, PK24, KJP*24, LTL*21b]. Yet, there has been no systematic study on the performance of LLMs in visualization troubleshooting, especially compared to humans (*e.g.,* on forums). In this paper, we aim to evaluate LLMs' ability to address real-world instances found in forums.

2.3. Visualization Troubleshooting

Visualization troubleshooting refers to resolving user-identified issues in defective visualizations to achieve desired outputs. Existing studies have explored what defective visualizations are generated both by humans and AI. From the human perspective, Leo et al. [LCYQ24,LQ25] reviewed over 1,000 misleading visualizations and developed a taxonomy of common misleading elements. Lan et al. [LL25] analyzed 2,227 flawed visualizations from an online gallery, classifying 76 design flaws into three types: misinformation, uninformativeness, and unsociability. Battle et al. [BFW22] identified debugging challenges faced by D3 users through forum analysis. From the AI perspective, Choi et al. [CLJ24] analyzed design issues in LLM-generated visualizations. Leo et al. [LQ25] explored prompt engineering to improve LLMs' detection of misleading visualizations. VisEval [CZX*24] introduced a dataset and evaluation framework to assess LLM-generated visualizations, focusing on validity, legality, and readability. However, most studies focus on what are defective visualizations, lacking a systematic investigation into how such visualizations can be addressed.

Some tools have been developed to fix incorrect charts. For instance, VizLinter [CSX*22] uses constraint programming to detect and rectify flaws in existing non-compliant charts. Similarly, Bavisitter [CLJ24] addresses non-compliant charts generated by LLMs. MobileVisFixer [WTD*20] and GeoLinter [LFMM23] are tailored for specific scenarios, which optimize layouts for mobile and geographic visualizations. While these tools assist in linting incorrect visualizations into reasonable ones, they may not satisfy users' more specific, fine-grained expectations. This paper focuses on resolving defective visualizations (without compilation errors and with user-identified issues) into desired outputs.

3. Dataset Curation

In this section, we collect and curate real-world datasets from *Stack Overflow* to investigate the four questions outlined in Figure 1, ranging from understanding common troubleshooting issues faced by users to evaluating the effectiveness of LLMs in providing solutions. Figure 2 provides an overview of the dataset curation process.

Data Collection. Stack Overflow contains numerous Q&A cases related to visualization troubleshooting, which typically feature a range of issues such as coding errors, design challenges, and specific visualization operations. For an example, refer to this case[†]. These cases often involve multimodal question descriptions, combining text-based issue descriptions, defective visualizations, code snippets, and datasets. As shown in Figure 2(a), we crawled 889 high-quality cases tagged with "*Vega-Lite*" from Stack Overflow. Specifically, each case consists of a *question description* and its *corresponding best answer*, which is selected by the question owner based on relevance and effectiveness, often indicated with a note like "*the question owner accepted this as the best answer*".

Dataset Curation for Q1. To analyze how users formulate their troubleshooting questions, we first selected non-troubleshooting cases from the 889 collected examples. Based on the expert evaluation, the questions were classified into three main categories: ***troubleshooting*** *issues*, ***authoring*** *issues*, and ***system*** *issues*. Authoring and system issues were considered outside the scope of our study, as they primarily concern platform-specific problems or basic authoring queries. In contrast, troubleshooting issues, such as code errors or incorrect visual outputs (*e.g.,* "Dashed line in legend?"), focus on defects in visualizations and formed the core of our analysis. For the classification process, we employed GPT-4 with detailed prompts, providing clear definitions and representative examples for each category (see Figure 2(b)). Each case was assigned a single primary category, with the possibility of multiple subcategories to capture the complexity of the issues. After filtering, we manually reviewed the cases to ensure all troubleshooting issues were included, resulting in a final set of 758 troubleshooting-related cases. The statistics of the classification results are shown in Figure 3. We will discuss the details in Section 4.1.

Dataset Curation for Q2. To investigate human responses to visualization troubleshooting questions, we used the 758 troubleshooting-related cases identified in **Q1**. Our goal was to analyze the structure of the *best answers* provided in these cases, with a focus on two key aspects: the use of supplementary reference materials and the characteristics of solution codes. To this end, we first categorized the best answers based on their use of supplementary resources, which we summarized in Table 1. Next, we examined the

[†] https://stackoverflow.com/questions/56425430

Table 1: *Distribution of external links in our curated real-world datasets across troubleshooting issues.*

Source	External Link Category									
	Visualization Assets			Official Resources			Additional Resources	Community Resources		
	Vega-Lite Code	Dataset File	Visualization	Documentation	Example	Usage	Unofficial Documentation	Stack Overflow	Github	Other Forum
#Question	254	18	398	55	46	1	30	57	52	17
#Best Answer	234	1	457	140	12	1	40	25	88	14

two levels of code-based solutions: solution-level and operation-level codes, which are essential for understanding the complexity of the responses. Figure 4 provides an example illustrating the distinction between these two levels.

As shown in Figure 2(c), we validated the answer codes using the `vl_convert` package, which revealed that only 288 cases contained valid codes that met the established criteria. For these valid cases, we classified code complexity to determine whether solution-level code contributed to increased complexity (see Table 2). Additionally, we employed prompt-based classification to identify and categorize key operation-level codes, with the results visualized in Figure 6.

Dataset Curation for Q3. To evaluate LLMs' performance in addressing troubleshooting issues, we used 288 forum cases that contained valid answer codes (see Figure 2(c)). Since users often embed external links within their questions, as detailed in Table 1, we ensured that this information was included as part of the LLM's input. For each case, we collected all available information from the forum, including text descriptions, external links, and visualizations. This comprehensive data collection allowed the LLM to access the same resources available to the forum responder, ensuring a fair and accurate evaluation of troubleshooting performance.

Dataset Curation for Q4. To evaluate the impact of supplementary resources on LLM performance, we reviewed 288 cases, identifying 43 cases that included documentation, 2 with examples, and 2 containing both. For cases lacking supplementary resources, we manually enriched the dataset by adding relevant guidance from the official Vega-Lite documentation or examples from the Vega-Lite gallery. When suitable examples were unavailable, we extended our search to GitHub repositories. As a result, a total of 47 cases were supplemented with both documentation and examples. Figure 2(d) provides an overview of this process.

Furthermore, we reviewed forum solutions to ensure that incomplete visualizations met the required standards. 7 forum responses were refined for clarity and completeness, while the rest were considered gold-standard solutions. This curated dataset facilitated a systematic assessment of how additional resources impact LLM performance in troubleshooting tasks.

4. Human-assisted Visualization Troubleshooting

In this section, we explore human-assisted visualization troubleshooting by analyzing two key aspects, as shown in Figure 1: **(Q1)** What questions do users ask in the Q&A forum? and **(Q2)** How do responders in forums address troubleshooting issues?

4.1. (Q1) What Questions Do Users Ask in Q&A Forum?

To understand how users formulate their questions in Q&A forum, we analyzed three key aspects: the types of questions asked, the non-textual elements included in the questions, and the tools and resources users reference to resolve their issues. We conducted this analysis using the dataset curated in Figures 2(a) and (b).

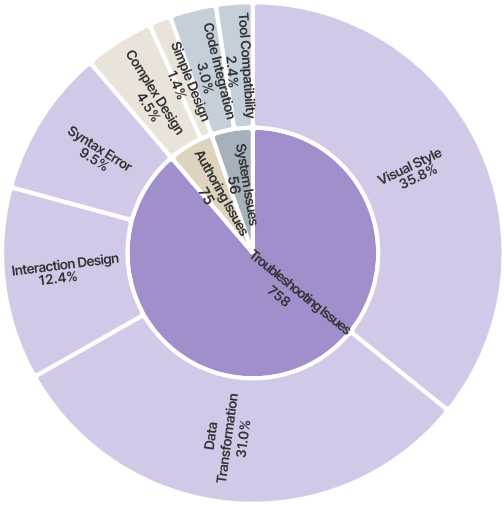

Figure 3: *Classification of questions in forums. The inner circle represents the count of cases in each main category, while the outer circle displays the percentage distribution of subcategories within those main categories. Each question is assigned to a main category, but may include multiple subcategories.*

4.1.1. Analysis of Question Types

As shown in Figure 2(a), we classified the 889 collected examples into three categories: ***troubleshooting*** *issues*, ***authoring*** *issues*, and ***system*** *issues*. Figure 3 shows the distribution of question types.

Troubleshooting issues (758 out of 889 cases) are the most common questions in the forum, representing a significant demand for real-case visualization. These questions often involve defective visualizations, typically accompanied by embedded code snippets. The primary concern is on improving or correcting an existing visualization rather than creating a new one from scratch. Troubleshooting issues can be divided into four subcategories.

- **Visual Style (35.8%)** emerged as the most prevalent subcategory. These inquiries adopt a visual problem-solving approach, focusing on adjusting or enhancing visual elements. For example, *"How to get a dashed line in the legend?"* The user seeks guidance on altering the visual style of chart elements, specifically the legend.
- **Data Transformation (31.0%)** is the second most common subcategory, involving questions originating from data requirements, including restructuring data for visualization. For exam-

ple, *"How to encode table-based data?"* The user struggles with converting data structures to match visualization requirements.

- **Interaction Design (12.4%)** represents the third common subcategory. These questions involve adding or refining interactive visual features where users describe dynamic behaviors rather than static elements. Such inquiries are often accompanied by code that functions partially, with interaction defects. For example, *"Is there a way to have a dynamic tooltip in Deneb?"* The user has already completed the tooltips but wants them to dynamically display updated values for each line in the chart.
- **Syntax Error (9.5%)** is another common subcategory, with users frequently including fragments of Vega-Lite compiler error logs. These issues can be challenging for users, who often struggle to derive actionable insights from the error messages. For example, *"Vega-Lite editor table is empty"*. The user encounters an error caused by incorrect code or configurations.

Authoring issues (75 out of 889 cases), rather than explicitly involving defective visualizations, typically contain code snippets related to functionality. Question owners usually provide data and describe their requirements, requesting the creation of visualizations from scratch or addressing specific needs without relying on complete code. **System issues (56 out of 889 cases)** arise when users tag multiple environments or language-related labels with their questions, commonly found in cross-categorized forum discussions. These inquiries relate to challenges beyond the Vega-Lite compiler. A more detailed discussion of these categories can be found in Section A in our supplementary materials.

4.1.2. What Example References Were Used in the Question?

By analyzing the 758 cases, as shown in Figure 2(b), we found that questions often include embedded elements such as *code examples*, *visualizations*, and *input datasets*, all integrated with *textual descriptions* to make the issue more comprehensible.

Textual descriptions provide the primary context for the issue, where users express their needs, including observations, difficulties, and visualization targets. To improve clarity, many questions are supplemented with **visualizations**, which can include defective, human-annotated, or desired visualizations. In addition, users frequently provide existing **code examples** as a basis for modification to address the issue. These examples may be embedded within the text or linked to external code execution platforms. Furthermore, the **input datasets** serve to validate the generated visualizations and provide the data structure necessary for code implementation. The dataset may be provided as an embedded table within the code or description, or shared via external file links (*e.g.,* CSV files). In some cases, the dataset might be omitted entirely, requiring responders to infer it independently and resolve the issue.

4.1.3. What Tools and Resources Are Used to Solve Problems?

In the forum, askers often attempt to solve their problems independently before posting their questions. They leverage a variety of tools and resources, including documentation, example galleries, previous Q&A threads, and their own exploration results. Despite these efforts, they may fail to resolve their issues, leading them to seek help on the forum. This section investigates the strategies and resources askers typically use prior to asking for assistance.

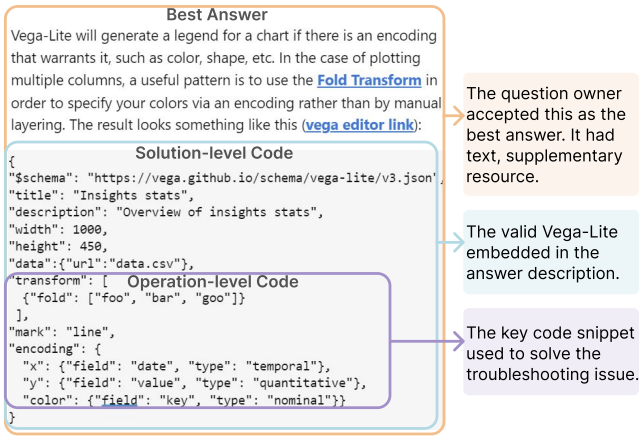

Figure 4: *Structure of forum responses to Vega-Lite troubleshooting issues, showing the different levels of solutions.*

In 71.4% of cases, no supplementary materials were provided, while the remaining 28.6% were analyzed in detail (see Table 1). A single case may involve the use of multiple supplementary resources. The main tools and resources used by askers include:

- **Documentation (11.1%)** is one of the most commonly used supplementary resources. It includes official Vega-Lite documentation, which serves as a primary reference for understanding syntax, along with other external documentation such as Altair guides, Wiki pages, and community-generated resources. These materials help users adapt Vega-Lite concepts to broader contexts and provide an in-depth understanding of syntax and features.
- **Examples (6.1%)** sourced from the official Vega-Lite website serve as templates for implementation. These examples showcase a wide range of use cases, offering users ready-made structures that they can customize to meet specific requirements.
- **Existing Q&A Resources (15.2%)** refer to the supplementary problem-related materials users draw upon, often originating from previous forum discussions. These include answers from Stack Overflow, GitHub discussions (particularly those related to Altair), and posts on other forums. These materials provide historical context and tested solutions, helping users troubleshoot visualization issues based on the experiences of others.
- **Self-Solved Attempts with Tools** refer to users' previous efforts to address the issue independently, such as experimenting with different implementation methods, using tools like LLMs (*e.g.,* ChatGPT), or trying other visualization libraries (*e.g.,* Matplotlib, Plotly, Seaborn) to explore potential solutions. For example, *"One possible solution is to just use native Python and if the X-axis is datetime data, then write the code."* The user uses Python to tackle the problem.

4.2. (Q2) How Do Responders Solve Troubleshooting Issues?

In this section, we evaluate how responders address troubleshooting issues by analyzing the structure of the "best" answers. We examine three key aspects: the characteristics of the best answers, the impact of solution-level code on code complexity, and the operation-level code involved in these solutions. As shown in Figure 4, the struc-

Table 2: *Code Complexity Classification. Code complexity is classified into four levels: Simple (16 or fewer keys), Medium (17 to 24 keys), Complex (25 to 41 keys), and Extra Complex (more than 41 keys).*

Type	Evaluation metric / criteria	Forum	
		Question Code	Solution
Quantity	# of specs	288	288
Complexity	Total # of keys across specs	12368	13438
	Average # of keys in a spec	42.94	46.66
	Simple (#-keys $\leq$ 16)	36	23
	Medium (#-keys $\leq$ 24)	57	50
	Complex (#-keys $\leq$ 41)	84	87
	Extra complex (#-keys $>$ 41)	111	128
Key Differences	Key decreased	32	
	Key unchanged	64	
	Key increased	192	

ture of the best answer typically consists of two main components: **solution-level** code and **operation-level** code. The solution-level code provides the complete Vega-Lite code that addresses the question, while the operation-level code focuses on the specific operations or modifications used to resolve the issue. This breakdown allows for a more detailed analysis of how responders craft their answers and the tools they use to enhance clarity and effectiveness.

4.2.1. The Key Characteristics of the Best Answer?

To identify the key characteristics of the best answers, we analyzed 758 best-answer structures (see Table 1). A single answer may involve the use of multiple supplementary resources. Our findings reveal several patterns:

- **Complete Vega-Lite code** is one of the most commonly included supplementary resources. It encompasses fully functional Vega-Lite code that directly addresses the user's question. Some answers also provided external links to code execution platforms (30.9%), offering a more interactive solution.
- **Visual aids (60.3%)** are regularly used in the best answers, providing important clarifications to the textual explanation. Visual aids help responders demonstrate the output of the code and further clarify the troubleshooting process.
- **Supplementary resources** are frequently referenced in the best answers. A significant portion of the best answers (24.0%) referenced official or unofficial manuals, while 16.0% cited prior Stack Overflow questions or external forums. These resources enriched the answers, offering additional context and ensuring a comprehensive solution.

4.2.2. Do Solution-Level Codes Increase Code Complexity?

When analyzing the solution-level code in the best answers, our goal is to understand whether providing a complete solution typically increases the complexity of the code. This is important for determining how responders balance code complexity with effectiveness when offering solutions to troubleshooting issues.

Measuring Code Complexity. Inspired by the ChartLLM framework [KJP*24], we classify code complexity based on the number of keys in Vega-Lite specifications. Specifically, we utilized the Vega-Lite example gallery dataset to establish a criterion for code complexity by calculating the quartiles of the number of keys in

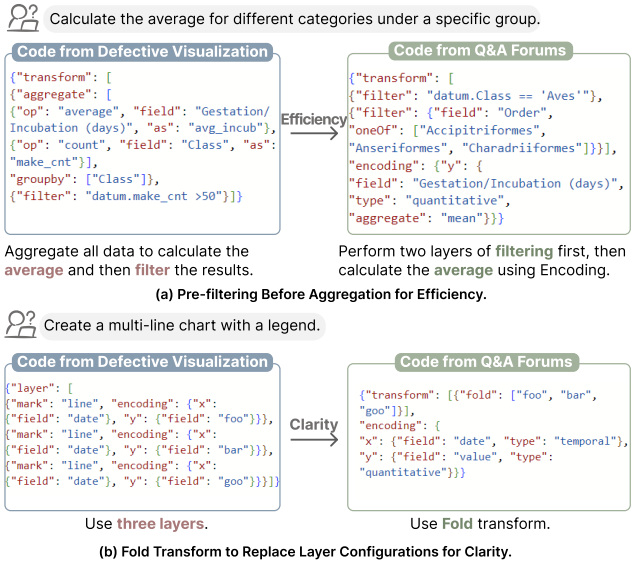

Figure 5: *Examples of reduced code complexity. Figure(a) illustrates an example of enhancing efficiency. The code from the user's initial defective visualization first aggregates and then filters. The solution from forums is pre-filtering before aggregation. Figure(b) demonstrates an example of enhancing clarity. The code from the user's initial defective visualization employs three layers. The solution from forums is fold transform to replace layer configurations.*

the specifications: $Q_1 = 16$, $Q_2 = 24$, and $Q_3 = 41$. These values serve as conditions for classifying code complexity into different levels (*e.g.,* simple, complex, extra-complex). Table 2 outlines this classification in detail. Next, we analyze their characteristics.

General Trend Toward Increased Complexity. The analysis of 288 forum cases reveals a general trend where the solution-level code tends to be more complex than the original question code. Specifically: (1) 29.2% of question codes were classified as complex, and 38.5% were extra-complex; (2) In contrast, 30.2% of solution-level codes were classified as complex, while 44.4% were extra-complex. On average, the complexity increased from 42.94 (in the question code) to 46.66 (in the solution-level code).

The above indicates that, in many cases, responders add advanced configurations or more detailed code structures to solve the issues effectively. The increase in complexity suggests that responders often need to incorporate more intricate solutions, especially when addressing more complex visualization problems.

Cases of Complexity Reduction. However, not all solutions result in increased complexity. In fact, 32 cases were identified where the solution-level code reduced complexity while maintaining or even improving the clarity and efficiency of the code. These cases highlight the responders' ability to simplify the problem while still providing an effective solution. Specifically:

- **Filtering before encoding averages**: As shown in Figure 5(a), this approach reduces redundancy in the code, making it more efficient while achieving the same functionality.
- **Using the fold transform**: As shown in Figure 5(b), this case

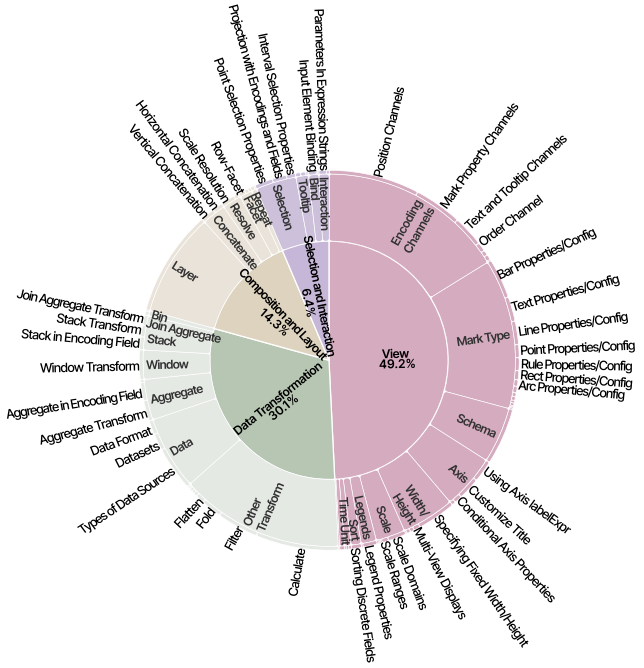

Figure 6: *Statistics of Operation-level Codes. The inner circle represents main categories, the middle circle represents subcategories, and the outer circle shows individual operations. Only frequently occurring operations are included. Multiple labels may be assigned to each operation due to the use of prompt-based methods.*

reduces the number of layers needed, simplifying the code and improving its clarity.

These examples demonstrate that, while many solutions may increase complexity to address more sophisticated visualization problems, others strategically reduce complexity to make the code more efficient and easier to understand.

4.2.3. The Characteristics of Operation-Level Codes

To better understand the operation-level code used in solutions (an example is shown in Figure 4), we classified the types of operations included in the best answers. This classification was based on the *Table of Contents* from the *Vega-Lite User Manual*, which helped us organize the operations into four main categories, 28 subcategories, and 135 distinct operations. The detailed classification is visually represented in Figure 6. For more details on how the operation classification was constructed, please refer to Section B in our supplementary materials.

Effective solutions in visualization troubleshooting often depend on the coordination of individual operations. Here, we break down the most commonly used operation-level codes based on their category and subcategory usage patterns:

- **Operations in View (49.2%)**: This category is the most frequently employed in solutions, indicating that visual elements and their layout are central to solving troubleshooting issues. Among the subcategories, *encoding channels* (16%) play a crucial role, with particular emphasis on *position channels* (7.7%). These channels often work in combination with *layer* for composition and layout adjustments, or with *calculate* for data transformation. This highlights that users frequently adjust positioning in multi-layer visualizations and often use complex expressions to fine-tune their visual layout. In addition, *mark types* (13.0%) are essential, with *bar* (3.4%) being the most common type. Stacked bar charts, in particular, are frequently encountered. *Line* (2.0%) and *text* (2.1%) are also used to address visualization issues, with more than half of these adjustments occurring in multi-layer visualizations. Another common operation in view is *specifying fixed width/height* (3.3%), particularly in multi-layer visualizations, where precise control over size and layout is often required.
- **Operations in Data Transformation (30.1%)**: This is the second most common type of operation used in the solutions. *Calculate* (7.0%) is the most frequently used operation, enabling additional calculations necessary for data transformation. It is often combined with other operations like *window*, *fold*, or *filter* to meet specific requirements. *Filter* (3.5%) is also prevalent, particularly when users need to refine data before visualization. Furthermore, many users encounter issues with the data format, such as arrays or multi-column structures, which differ from the standard input format expected by Vega-Lite. These issues often require transformations using operations like *flatten*, *fold*, and *window* to make the data compatible with the desired visualizations. While visual-style operations are common, users often focus on solving the visual issues rather than directly addressing the underlying data structure.
- **Operations in Composition and Layout (14.3%)**: This category focuses on the arrangement and alignment of visual elements, particularly in multi-layer visualizations. Issues in composition and layout are often resolved through a combination of operations from the *view* and *data transformation* categories. A key operation within this category is *scale resolution* (1.2%), where users need to unify scales or share axes between different charts. This highlights the need for consistency in layout when dealing with composed charts.
- **Operations in Selection and Interaction (6.4%)**: Selection and interaction operations are used to define parameters and bind them to specific actions. *Point selection properties* (1.5%) is one of the most common selection operations, particularly when working with multi-layer visualizations. These operations enable users to interact with the visualized data in meaningful ways, such as selecting specific points for further analysis or action.

5. AI-assisted Visualization Troubleshooting

This section examines the role of AI-assisted support in visualization troubleshooting, comparing LLMs (specifically GPT-4V) with human assistance by evaluating their performance in addressing forum issues **(Q3)** and investigating how supplementary resources influence the quality of LLM-generated answers **(Q4)**.

5.1. (Q3) Make LLMs Solve the Same Troubleshooting Tasks

Experimental Design. To explore whether LLMs can effectively solve visualization troubleshooting issues (Q3), we conducted a Zero-shot experiment (Case 0). We selected 288 forum cases containing valid answer codes. For each case, the LLM was tasked with providing a comprehensive response, including problem understanding, solution description, and the corresponding solution-level code. The solution-level code from the forum served as the reference for evaluating the correctness of the LLM's output. To ensure the evaluation's accuracy, two visualization experts reviewed the questions and the forum solutions, assessing whether the LLM's code aligned with the problem's requirements.

Experimental Results. The results showed that LLMs struggled to solve problems independently. Only 56 out of 288 cases (19.4%) were deemed correct by human evaluators, indicating that LLMs alone often fail to fully resolve complex visualization issues. This highlights the necessity of incorporating supplementary resources to support LLM-based troubleshooting, mirroring the human problem-solving process.

Interestingly, in some cases, the solution-level code generated by LLMs outperformed that of forum answers. For example, in the case of *"Plot multiple columns with Vega-Lite and include a legend"*, a forum response incorrectly used mismatched data, resulting in a visualization that did not meet the user's requirements. In contrast, the LLM applied the correct data, yielding a visualization that fulfilled the user's intent. Similarly, for the question *"Is it possible to apply the same condition as the color encoding for the legend?"*, the forum response offered only general guidance and reference documentation, without modifying the original code. The LLM, however, generated a complete solution based on the existing defective visualization.

However, comparing solutions based solely on the correctness of the visual output may overlook alternative valid approaches. In some cases, multiple correct solutions exist. For instance, in the question *"How to specify a Rule Line with a single value in Vega-Lite?"*, there are different ways to position the rule line, such as using a fixed numeric value or dynamically positioning it based on data. Our expert evaluators identified at least two valid approaches, suggesting that a more nuanced evaluation criterion is needed to assess the range of possible solutions accurately.

5.2. (Q4) The Impact of Supplementary Resources on LLMs

Experimental Design. To explore the impact of supplementary resources on LLM-based solutions (Q4), we designed experiments using single-type and combined supplementary materials. As shown in Figure 2(b), we collected 47 cases, with some including documentation (**+Doc**), examples (**+Ex.**), or both (**+Doc and Ex.**). Furthermore, we manually refined the quality of these materials, resulting in 9 cases generated from more detailed documentation (**+Doc***) and 7 from better examples (**+Ex.***).

To evaluate solution quality, we invited two visualization experts to score both the forum-based and LLM-generated solutions using a 5-point scale, designed based on insights from (Q3). In total, 47 scores were collected for forum solutions and 157 for LLM solutions. Next, we introduce the scoring criterion.

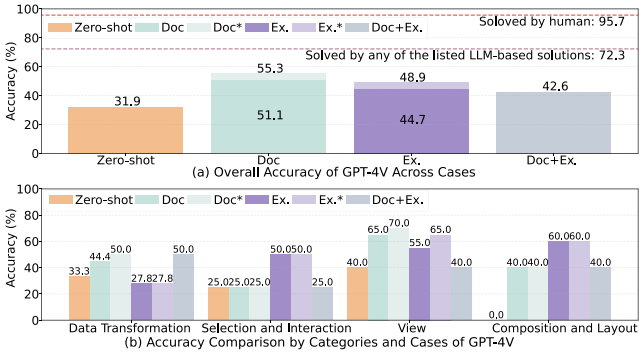

Figure 7: *Experimental Accuracy. Figure(a) A stacked bar chart showing overall accuracy across the three cases (Case 0: zero-shot, Case 1: documentation or example, and Case 2: documentation and example). The stacked portions are the result of the refined documentation or example. Two horizontal lines represent the forum solution score (red) and the score for cases solved by any LLM-based solution (purple). Figure(b) A grouped bar chart illustrating the accuracy across different categories (Data Transformation, Selection, Interaction, View, Composition, and Layout) for the three cases.*

Scoring Criterion. As found in Section 5.1, LLMs often simplify solution codes, which may not align fully with the specific requirements of the troubleshooting tasks. We annotated key operation-level codes in the gold-standard solutions for 47 cases and defined a 5-point scale for assessing correctness:

- **Score 0 (No Key)**: Missing key operations, indicating a lack of understanding of the core issue.
- **Score 1 (Partial Key)**: Only partial key operations included, offering an incomplete solution.
- **Score 2 (Key Operations but Incorrect)**: All key operations present, but with incorrect implementation or expression.
- **Score 3 (Correct but Not Displayed)**: Correct operations and expression are present, but the visualization does not render correctly, often due to code simplification or alternative data processing methods.
- **Score 4 (Correct Visualization)**: The solution meets the requirements and successfully renders the visualization, even if it differs from the forum's solution.

Experimental Results. Table 3 shows the overall impact of supplementary resources on LLMs.

Effectiveness of Single-Supplementary Resources. Compared to the zero-shot setting, The accuracy of LLM-based solutions improved to 51.1% when supplemented with documentation alone (**+Doc**). Providing examples alone resulted in a slightly lower accuracy of 44.7% (**+Ex.**).

Effectiveness of Combined Resources. When both documentation and examples were used together, the accuracy dropped to 42.6% (**+Doc and Ex.**). The increase in token length due to the combination led to a loss of focus on the core aspects of the problem, making it harder for LLMs to generate effective solutions.

Table 3: *Overall evaluation results showing the accuracy of forum-based solutions and LLM-based solutions across 47 cases.*

Count	Question Types (Operation Details)	Experimental Results (Accuracy)				
		Forum-based Solutions	GPT-4V Zero-Shot	GPT-4V (+Doc)	GPT-4V (+Ex.)	GPT-4V (+Doc and Ex.)
20	**View**	**90%**	**40%**	**65%**	**55%**	**40%**
6	Color chart with specific colors (Scale, Condition)	83.3%	50%	100%	66.7%	66.7%
4	Add line mark (Rule Properties/Config)	75%	75%	75%	75%	50%
1	Conditional grid format (Grid)	100%	0%	0%	0%	0%
1	Discontinuous issue (Discrete Scales)	100%	0%	0%	0%	0%
1	Hide legend and normalise size (Legends, Circle Properties/Config)	100%	100%	100%	100%	0%
3	Overlay text (Stack, Calculate, Join Aggregate)	100%	0%	33.3%	66.7%	33.3%
4	Text format (Format, Time Unit)	100%	25%	50%	25%	25%
18	**Data Transformation**	**100%**	**33.3%**	**44.4%**	**27.8%**	**50%**
6	Data sorting (Window, Calculate, Sort, Order Channel)	100%	33.3%	33.3%	16.7%	50%
6	Data structure (Flatten, Fold, Calculate)	100%	33.3%	50%	33.3%	66.7%
3	Data transformation (Aggregate, Window, Join aggregate)	100%	0%	33.3%	0%	0%
3	Topn (Window, Filter, Aggregate)	100%	66.7%	66.7%	66.7%	66.7%
5	**Composition and Layout**	**100%**	**0%**	**40%**	**60%**	**40%**
1	Create scatter matrix (Pivot, Repeat)	100%	0%	0%	0%	0%
1	Field appears in the wrong chart (Concatenate)	100%	0%	0%	0%	0%
1	Independent color in concatenated plot (Resolve)	100%	100%	100%	100%	0%
2	Share axes in concatenated plot (Resolve)	100%	0%	50%	100%	100%
4	**Selection and Interaction**	**100%**	**25%**	**25%**	**50%**	**25%**
2	Add tooltip (Tooltip Based on Encoding, Tooltip Based on Data Point)	100%	50%	50%	50%	50%
1	Clicking on the main bar chart (Using Parameters Data Extents)	100%	0%	0%	100%	0%
1	Select at a distance (Point Selection Properties)	100%	0%	0%	0%	0%
47	**Overall Accuracy**	**95.7%**	**31.9%**	**51.1%**	**44.7%**	**42.6%**

Figure 7(a) shows the effectiveness of LLMs-based solutions by improving the quality of supplementary materials. Overall, the best overall accuracy was achieved using a combination of different LLM-based solutions, with an accuracy rate of 72.3% (the purple line in Figure 7(a)). This surpassed the best individual solution (+Doc*) with 55.3% accuracy, indicating that using a variety of methods helped find better solutions for a broader set of problems.

Refined Resources for Improved Accuracy. (1) Improved Documentation (+Doc*): By refining the documentation, accuracy improved by 4.2%, leading to better solutions (9 cases). (2) Improved Examples (+Ex*): Similarly, improved examples resulted in a 4.2% increase in accuracy, with 7 cases showing improvement.

Contribution of Supplementary Resources by Problem Type: Figure 7(b) shows that incorporating documentation (+Doc*) was most effective for data transformation (50%) and view problems (70%), as these tasks benefit from detailed method explanations found in documentation. What's more, adding demonstration examples (+Ex*) proved more helpful for issues related to selection and interaction (50%) and composition and layout (60%), where actual code examples provide more intuitive guidance and help users understand complex interactions and layouts.

The above findings highlight the importance of choosing the right supplementary resource (documentation or examples) depending on the type of troubleshooting task and suggest that combining multiple resources may be more effective for addressing a wider variety of problems in visualization troubleshooting.

6. Discussion

This section summarizes the key insights derived from our empirical study, highlights potential research opportunities, and acknowledges the limitations of our work.

6.1. Key Findings

Finding-1: The Role of Human Assistance. Human-assisted (or expert-assisted) solutions are highly effective for resolving complex visualization issues due to their ability to interpret context, understand user intent, and integrate diverse resources like visualization examples, external materials, etc (see Sections 4.1 and 4.2). Human responders can fill in contextual gaps, infer problems using prior experience, and combine resources to address both symptoms and root causes. However, delays and variability in expertise can make it unsuitable for time-sensitive tasks, sometimes resulting in suboptimal solutions.

Finding-2: The Impact of Effective Questions on Response Quality. To improve response effectiveness, users should include structured data and a clear context, detailing the issue, prior attempts, and specific problem areas. (see Sections 4.1.2 and 4.1.3). Incorporating relevant examples, such as code snippets, tables, or current charts, helps clarify discrepancies and directs responders' efforts. Supplementary materials like documentation, prior exploration results, or related threads can further narrow the problem space, but they should be concise and directly relevant. Demonstrating iterative efforts, such as previous debugging attempts or applied solutions, signals engagement and encourages responders to build on existing work.

Finding-3: Characteristics of the Best Answers. The most effective human responses often incorporate supplementary resources, such as links to documentation or examples, which enhance the clarity and comprehensibility of the solution (see Section 4.2.1). When these resources are directly relevant to the problem, they help users address the issue more efficiently and accurately (see Figure 5).

Finding-4: The Role of LLMs Assistance. LLMs are effective for addressing simpler visualization troubleshooting issues by providing quick responses and actionable insights. However, they struggle with more complex problems that require deeper understanding or context-specific knowledge (see Table 3). Even when LLM outputs are not fully accurate, *they often highlight key aspects of the visualization troubleshooting issues.* Users should take these insights as starting points and refine them rather than expecting a perfect solution (see Section 5.1).

Finding-5: The Impact of Supplementary Materials on LLM Effectiveness Task-specific supplementary materials, such as relevant examples or documentation, can improve LLM performance in troubleshooting tasks. However, overloading LLMs with excessive resources may impede their ability to focus on the core issue (see Table 3). Users should prioritize concise, targeted materials that are directly aligned with the problem (see Figure 7(a)). Different resources serve distinct purposes: *documentation is ideal for data transformation and view-related issues*, while *examples are more effective for selection, interaction, composition, and layout challenges* (see Figure 7(b)). By carefully selecting and combining resources based on the problem type, users can enhance LLM performance. Despite this, users should still refine LLM-generated solutions to address potential oversimplifications, such as adjusting data paths or adding missing operations.

6.2. Design Implications for Future Practices and Research

How should programming language developers design the structure of tutorials? Users often combine prior exploration results with tutorials (*e.g.,* documentation) initially but still fail to resolve the issue, leading them to seek additional support (see Section 4.1). Existing programming language tutorials provide detailed descriptions of individual operations but often lack guidance on how to combine these operations effectively. This gap is a key reason users struggle to resolve issues independently. We found that users' codes are often partially incorrect rather than entirely wrong (see Section 5.2) and many troubleshooting problems involve specific patterns of operation combinations (see Section 4.2.3).

To make supplementary resources more effective, commonly observed operation combination patterns could be systematically summarized in the future to offer guidance to users. Developers could consider offering an alternative documentation version that aligns resources with user needs and prevalent issues. In addition, supplementary resources could offer intelligent suggestions, presenting relevant and practical examples beyond basic keyword matching. Incorporating interactive elements like guided tutorials could enhance the engagement and utility of these resources. Furthermore, user code exhibits potential for improvement in clarity and efficiency (see Section 4.2.2). Introducing optimization tools to streamline logic and enhance performance would not only improve the user experience but also elevate the quality of the code.

How should researchers enhance the effectiveness of AI-assisted support in troubleshooting? We found that well-designed supplementary resources can significantly enhance LLM-based solutions (see Section 5.2). For example, refined documentation and examples positively influence solution accuracy (see Figure 7(b)).

Moreover, the score for cases solved by any LLM-based solution consistently achieves the highest overall accuracy across all cases (see Figure 7(a)). By using diverse combinations, more effective LLM-based solutions can be found for a wider range of problem types, reducing accuracy fluctuations that may arise from relying on a single method.

Researchers can further improve LLM-based solutions by designing appropriate Retrieval-Augmented Generation (RAG) solutions [TYF*24]. By analyzing the user's query semantics, RAG technologies can retrieve and integrate refined documentation and examples that are relevant to the problem. Furthermore, the analysis from Table 3 indicates similar issues with similar solutions, suggesting that targeted RAG prompt strategies can effectively address various question types. These strategies could deduce potential operations, bridging the gap between operation-focused resources and user-specific inquiries.

6.3. Limitations

Our study also has some limitations. First, we relied solely on Stack Overflow for data curation. Future research could include other forums, such as Altair Issues or GitHub, to capture a broader range of troubleshooting scenarios. Second, our focus was limited to Vega-Lite due to its widespread use and mature ecosystem. Future work could explore other programming languages, such as D3 [BOH11] or Python, as well as interactive tools that may require different troubleshooting strategies. Third, our experiments used GPT-4V, representing only one AI framework. Evaluating additional models, such as GPT-4o, could offer a more comprehensive understanding of AI-assisted solutions in visualization troubleshooting.

7. Conclusions

This study presents a qualitative analysis of visualization troubleshooting cases in Q&A forums, comparing human-assisted and AI-assisted solutions. Through the systematic analysis of 889 Vega-Lite cases, we identified key insights into the formulation of troubleshooting questions and the effectiveness of various support strategies. Our findings highlight distinct differences between human and AI-driven solutions: while human assistance offers context-sensitive guidance, it varies in quality, AI-assisted solutions provide quick responses but often require additional resources to improve accuracy. Moreover, the study emphasizes the critical role of carefully selected supplementary materials in enhancing troubleshooting effectiveness. These insights offer valuable recommendations for forum users, tutorial designers, and the research community, contributing to better problem-solving practices in the field of visualization troubleshooting.

References

[BFW22] BATTLE L., FENG D., WEBBER K.: Exploring D3 Implementation Challenges on Stack Overflow. In *2022 IEEE Visualization and Visual Analytics (VIS)* (oct 2022), IEEE, pp. 1–5. doi:10.1109/VIS54862.2022.00009. 2, 3

[BH09] BOSTOCK M., HEER J.: Protovis: A Graphical Toolkit for Visualization. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (nov 2009), 1121–1128. doi:10.1109/TVCG.2009.174. 2

[BOH11] BOSTOCK M., OGIEVETSKY V., HEER J.: D3: Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (dec 2011), 2301–2309. doi:10.1109/TVCG.2011.185. 2, 10

[BS24] BENDECK A., STASKO J.: An Empirical Evaluation of the GPT-4 Multimodal Language Model on Visualization Literacy Tasks. *IEEE Transactions on Visualization and Computer Graphics* (2024), 1–11. doi:10.1109/TVCG.2024.3456155. 2

[CH24] CRISAN A., HOQUE E.: Towards a holistic evaluation of llm generated code for exploratory visual analysis. In *ACM CHI'24 Workshop on Human-Centered Evaluation and Auditing of Language Models* (2024). 2, 3

[CLJ24] CHOI J., LEE J., JO J.: Bavisitter: Integrating Design Guidelines into Large Language Models for Visualization Authoring. In *Proceedings of IEEE Visualization and Visual Analytics, IEEE VIS'24* (2024), IEEE, pp. 1–5. 3

[CSX*22] CHEN Q., SUN F., XU X., CHEN Z., WANG J., CAO N.: VizLinter: A Linter and Fixer Framework for Data Visualization. *IEEE Transactions on Visualization and Computer Graphics 28*, 1 (2022), 206–216. doi:10.1109/TVCG.2021.3114804. 2, 3

[CZX*24] CHEN N., ZHANG Y., XU J., REN K., YANG Y.: VisEval: A Benchmark for Data Visualization in the Era of Large Language Models. *IEEE Transactions on Visualization and Computer Graphics* (2024), 1–11. doi:10.1109/tvcg.2024.3456320. 2, 3

[Dib23] DIBIA V.: LIDA: A Tool for Automatic Generation of Grammar-Agnostic Visualizations and Infographics using Large Language Models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics, ACL'23* (Stroudsburg, PA, USA, 2023), ACL, pp. 113–126. doi:10.18653/v1/2023.acl-demo.11. 2

[KJP*24] KO H.-K., JEON H., PARK G., KIM D. H., KIM N. W., KIM J., SEO J.: Natural language dataset generation framework for visualizations powered by large language models. In *Proceedings of the CHI Conference on Human Factors in Computing Systems* (2024), pp. 1–22. 3, 6

[LCMZ21] LIU Z., CHEN C., MORALES F., ZHAO Y.: Atlas: Grammar-based Procedural Generation of Data Visualizations. *Proceedings - 2021 IEEE Visualization Conference - Short Papers, VIS 2021* (2021), 171–175. doi:10.1109/VIS49827.2021.9623315. 2

[LCYQ24] LO L. Y. H., CAO Y., YANG L., QU H.: Why Change My Design: Explaining Poorly Constructed Visualization Designs with Explorable Explanations. *IEEE Transactions on Visualization and Computer Graphics 30*, 1 (2024), 955–964. doi:10.1109/TVCG.2023.3327155. 3

[LFMM23] LEI F., FAN A., MACEACHREN A. M., MACIEJEWSKI R.: Geolinter: A linting framework for choropleth maps. *IEEE Transactions on Visualization and Computer Graphics* (2023). 3

[LL25] LAN X., LIU Y.: "I Came Across a Junk": Understanding Design Flaws of Data Visualization from the Public's Perspective. *IEEE Transactions on Visualization and Computer Graphics 31*, 1 (2025), 393–403. URL: https://ieeexplore.ieee.org/document/10670488/, doi:10.1109/TVCG.2024.3456341. 3

[LLC*24] LI B., LUO Y., CHAI C., LI G., TANG N.: The dawn of natural language to SQL: are we fully ready? [experiment, analysis \u0026 benchmark]. *Proc. VLDB Endow. 17*, 11 (2024), 3318–3331. 2

[LLF*24] LI G., LI R., FENG Y., ZHANG Y., LUO Y., LIU C. H.: Coinsight: Visual storytelling for hierarchical tables with connected insights. *IEEE Trans. Vis. Comput. Graph. 30*, 6 (2024), 3049–3061. 1

[LQ25] LO L. Y.-H., QU H.: How Good (Or Bad) Are LLMs at Detecting Misleading Visualizations? *IEEE Transactions on Visualization and Computer Graphics 31*, 1 (2025), 1116–1125. doi:10.1109/TVCG.2024.3456333. 3

[LQT*18] LUO Y., QIN X., TANG N., LI G., WANG X.: Deepeye: Creating good data visualizations by keyword search. In *SIGMOD Conference* (2018), ACM, pp. 1733–1736. 2

[LQTL18] LUO Y., QIN X., TANG N., LI G.: Deepeye: Towards automatic data visualization. In *ICDE* (2018), IEEE Computer Society, pp. 101–112. 1

[LSL*24] LIU X., SHEN S., LI B., MA P., JIANG R., LUO Y., ZHANG Y., FAN J., LI G., TANG N.: A survey of NL2SQL with large language models: Where are we, and where are we going? *CoRR abs/2408.05109* (2024). 2

[LTL21a] LUO Y., TANG J., LI G.: nvbench: A large-scale synthesized dataset for cross-domain natural language to visualization task. *arXiv preprint arXiv:2112.12926* (2021). 3

[LTL*21b] LUO Y., TANG N., LI G., CHAI C., LI W., QIN X.: Synthesizing natural language to visualization (NL2VIS) benchmarks from NL2SQL benchmarks. In *SIGMOD Conference* (2021), ACM, pp. 1235–1247. 3

[LTL*21c] LUO Y., TANG N., LI G., TANG J., CHAI C., QIN X.: Natural language to visualization by neural machine translation. *IEEE Transactions on Visualization and Computer Graphics 28*, 1 (2021), 217–226. 2

[MC21] MCNUTT A. M., CHUGH R.: Integrated Visualization Editing via Parameterized Declarative Templates. In *Proceedings of CHI Conference on Human Factors in Computing Systems, CHI'21* (Virtual Event, 2021), ACM, pp. 1–14. doi:10.1145/3411764.3445356. 2

[McN23] MCNUTT A. M.: No Grammar to Rule Them All: A Survey of JSON-style DSLs for Visualization. *IEEE Transactions on Visualization and Computer Graphics 29*, 1 (2023), 160 – 170. doi:10.1109/TVCG.2022.3209460. 2

[PDFE18] PARK D., DRUCKER S. M., FERNANDEZ R., ELMQVIST N.: Atom: A Grammar for Unit Visualizations. *IEEE Transactions on Visualization and Computer Graphics 24*, 12 (2018), 3032–3043. doi:10.1109/TVCG.2017.2785807. 2

[PK24] PONOCHEVNYI N., KUZMINYKH A.: Chart What I Say: Exploring Cross-Modality Prompt Alignment in AI-Assisted Chart Authoring. In *Proceedings of CHI Conference on Human Factors in Computing Systems, CHI'24 LBW* (2024), pp. 1–5. doi:10.1145/3613905.3650921. 3

[RF06] REAS C., FRY B.: Processing: programming for the media arts. *AI & SOCIETY 20*, 4 (sep 2006), 526–538. doi:10.1007/s00146-006-0050-9. 2

[SKL*16] SIDDIQUI T., KIM A., LEE J., KARAHALIOS K., PARAMESWARAN A.: Effortless data exploration with zenvisage: An expressive and interactive visual analytics system. *Proceedings of the VLDB Endowment 10*, 4 (2016), 457–468. arXiv:1604.03583. 2

[SLR*19] SATYANARAYAN A., LEE B., REN D., HEER J., STASKO J., THOMPSON J., BREHMER M., LIU Z.: Critical Reflections on Visualization Authoring Systems. *IEEE Transactions on Visualization and Computer Graphics 26*, 1 (2019), 1–11. doi:10.1109/TVCG.2019.2934281. 2

[SLW*24] SHEN L., LI H., WANG Y., LUO T., LUO Y., QU H.: Data playwright: Authoring data videos with annotated narration. *CoRR abs/2410.03093* (2024). 2

[SMWH17] SATYANARAYAN A., MORITZ D., WONGSUPHASAWAT K., HEER J.: Vega-Lite: A Grammar of Interactive Graphics. *IEEE Transactions on Visualization and Computer Graphics 23*, 1 (2017), 341–350. doi:10.1109/TVCG.2016.2599030. 1, 2

[SNL*21] SRINIVASAN A., NYAPATHY N., LEE B., DRUCKER S. M., STASKO J.: Collecting and characterizing natural language utterances for specifying data visualizations. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (2021), pp. 1–10. 3

[SRHH16] SATYANARAYAN A., RUSSELL R., HOFFSWELL J., HEER J.: Reactive Vega: A Streaming Dataflow Architecture for Declarative Interactive Visualization. *IEEE Transactions on Visualization and Computer Graphics 22*, 1 (jan 2016), 659–668. doi:10.1109/TVCG.2015.2467091. 2

[SSL*23] SHEN L., SHEN E., LUO Y., YANG X., HU X., ZHANG X., TAI Z., WANG J.: Towards Natural Language Interfaces for Data Visualization: A Survey. *IEEE Transactions on Visualization and Computer Graphics 29*, 6 (2023), 3121–3144. doi:10.1109/TVCG.2022.3148007. 2

[SST*22] SHEN L., SHEN E., TAI Z., WANG Y., LUO Y., WANG J.: GALVIS: Visualization Construction through Example-Powered Declarative Programming. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management, CIKM'22* (2022), ACM, pp. 4975–4979. doi:10.1145/3511808.3557159. 2

[STH02] STOLTE C., TANG D., HANRAHAN P.: Polaris: a system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics 8*, 1 (2002), 52–65. 2

[TCD*24] TIAN Y., CUI W., DENG D., YI X., YANG Y., ZHANG H., WU Y.: ChartGPT: Leveraging LLMs to Generate Charts from Abstract Natural Language. *IEEE Transactions on Visualization and Computer Graphics 14*, 8 (2024), 1–15. arXiv:2311.01920, doi:10.1109/TVCG.2024.3368621. 2

[TYF*24] TANG N., YANG C., FAN J., CAO L., LUO Y., HALEVY A. Y.: Verifai: Verified generative AI. In *CIDR* (2024), www.cidrdb.org. 10

[WGBH24] WANG H. W., GORDON M., BATTLE L., HEER J.: DracoGPT: Extracting Visualization Design Preferences from Large Language Models. *IEEE Transactions on Visualization and Computer Graphics* (2024), 1–11. doi:10.1109/TVCG.2024.3456350. 2

[WHS*23] WANG Y., HOU Z., SHEN L., WU T., WANG J., HUANG H., ZHANG H., ZHANG D.: Towards Natural Language-Based Visualization Authoring. *IEEE Transactions on Visualization and Computer Graphics 29*, 1 (2023), 1222 – 1232. doi:10.1109/TVCG.2022.3209357. 2

[WHT*24] WANG H. W., HOFFSWELL J., THANE S. M. T., BURSZTYN V. S., BEARFIELD C. X.: How Aligned are Human Chart Takeaways and LLM Predictions? A Case Study on Bar Charts with Varying Layouts. *IEEE Transactions on Visualization and Computer Graphics* (2024). doi:10.1109/TVCG.2024.3456378. 2

[Wic10] WICKHAM H.: A Layered Grammar of Graphics. *Journal of Computational and Graphical Statistics 19*, 1 (jan 2010), 3–28. doi:10.1198/jcgs.2009.07098. 2

[WTD*20] WU A., TONG W., DWYER T., LEE B., ISENBERG P., QU H.: Mobilevisfixer: Tailoring web visualizations for mobile phones leveraging an explainable reinforcement learning framework. *IEEE Transactions on Visualization and Computer Graphics 27*, 2 (2020), 464–474. 3

[WWZ*24] WU Y., WAN Y., ZHANG H., SUI Y., WEI W., ZHAO W., XU G., JIN H.: Automated Data Visualization from Natural Language via Large Language Models: An Exploratory Study. In *Proceedings of the ACM on Management of Data* (2024), ACM, pp. 1–28. doi:10.1145/3654992. 2

[WYS*24] WU Y., YAN L., SHEN L., WANG Y., TANG N., LUO Y.: Chartinsights: Evaluating multimodal large language models for low-level chart question answering. In *EMNLP (Findings)* (2024), Association for Computational Linguistics, pp. 12174–12200. 2

[WZW*23] WANG L., ZHANG S., WANG Y., LIM E.-P., WANG Y.: LLM4Vis: Explainable Visualization Recommendation using ChatGPT. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: Industry Track* (Stroudsburg, PA, USA, 2023), ACL, pp. 675–692. doi:10.18653/v1/2023.emnlp-industry.64. 2

[XLLT24] XIE Y., LUO Y., LI G., TANG N.: Haichart: Human and AI paired visualization system. *Proc. VLDB Endow. 17*, 11 (2024), 3178–3191. 1

[YHH*24] YE Y., HAO J., HOU Y., WANG Z., XIAO S., LUO Y., ZENG W.: Generative AI for visualization: State of the art and future directions. *CoRR abs/2404.18144* (2024). 2

[YSL*24] YU Y., SHEN L., LONG F., QU H., CHEN H.: PyGWalker: On-the-fly Assistant for Exploratory Visual Data Analysis. In *Proceedings of IEEE Visualization and Visual Analytics, IEEE VIS'24* (2024), IEEE, pp. 1–5. 2

[ZDL*24] ZHU Y., DU S., LI B., LUO Y., TANG N.: Are large language models good statisticians? In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track* (2024). URL: https://openreview.net/forum?id=j4CRWz418M. 2

[ZXY*24] ZHANG J., XIANG J., YU Z., TENG F., CHEN X., CHEN J., ZHUGE M., CHENG X., HONG S., WANG J., ZHENG B., LIU B., LUO Y., WU C.: Aflow: Automating agentic workflow generation. *CoRR abs/2410.10762* (2024). 2

Supplementary Materials

A. Detailed Analysis of Question Types

Based on expert evaluation, we have identified three main question types: troubleshooting issues, authoring issues, and system issues. The overall distribution of these question types is presented, with a detailed discussion of troubleshooting issues in Section 4.1.1.

In this section, we will provide an in-depth discussion of authoring issues and system issues.

Authoring issues (75 out of 889 cases) is the second most common category. These questions, rather than explicitly involving defective visualizations, typically contain code snippets related to functionality. Question owners usually provide data and describe their requirements, requesting the creation of visualizations from scratch or addressing specific needs without relying on complete code.

Authoring issues are divided into two subcategories.

- **Simple Design (1.4%)** involves straightforward visualization requirements and simple feature implementations. For example, *"How do I create a progress bar in Vega-Lite?"* The user is looking for a basic implementation of the bar using embedded data.
- **Complex Design (4.5%)** involves sophisticated visualization requirements or advanced feature implementations. For example, *"How can I have multiple levels in an axis in Vega-Lite?"* The user provides visual references (*e.g.,* images) to describe the desired output without including code. Questions about complex design are much more frequent than those about simple design. We hypothesize that most forum participants possess basic visualization coding skills and can implement simple designs using Vega-Lite gallery examples. However, for complex visualizations not covered in the gallery (such as double doughnut charts, Sankey diagrams, or box-and-whisker plots), users often struggle to implement these independently and typically provide desired visualizations in the questions.

System issues (56 out of 889 cases) is the least common category. Due to users potentially tagging multiple environments or language-related labels with their questions, these issues frequently appear in cross-categorized forum discussions. These questions pertain to challenges external to the Vega-Lite compiler.

System issues are divided into two subcategories.

- **Code Integration (3.0%)** involves challenges in embedding or integrating Vega-Lite into other platforms, such as Power BI or Deneb. For example, *"Vega-Lite API misbehaving."* The user encounters inconsistencies in output when using code and data across different environments.
- **Tool Compatibility (2.4%)** refers to issues related to the compatibility of tools, particularly when using Vega-Lite derivatives such as Altair or Vega. For example, *"How to zoom mark geoshape to a specific region in Altair?"* The user is seeking help in adapting geospatial visualizations for Altair.

B. Detailed Approach to Constructing Operation Classification

In Section 4.2.3, we will use classification to categorize key operational codes. To develop an operational classification for analyzing these codes, we reference Vega-Lite official website that has been curated and maintained over several years. We summarized the types of operations included in the solutions by using the *Table of Contents* from the *Vega-Lite Overview*. We collected Vega-Lite's 13 common main properties from the *Vega-Lite Overview*, excluded sections that are relatively independent or have content covered in other sections (*e.g., Overview, Config, Invalid Data, Property Types*), and organized the remaining content into four main categories: **View** (including *View Specification, Mark, Encoding, Projection*), **Data Transformation** (including *Data/Datasets, Transform*), **Composition and Layout** (including *View Composition*), and **Selection and Interaction** (including *Parameter, Tooltip*). We used documentation section titles from the *Vega-Lite Overview* as subcategories and first-level content from the documentation section as operation. For sections without first-level content, we grouped them under a specific name with section titles as operations. This classification resulted in 4 categories, 28 subcategories, and 135 operations. Table 4 summarizes the key characteristics.

Table 4: *Operation Classification from Vega-lite with categories describing visualization requirements, subcategories describing specific interfaces, and operations describing specific cases.*

Categories (4)	Sub-categories (28)	Operations (135)
Data Transformation	Data	Types of Data Sources, Data Format, Data Generators, Datasets
	Aggregate	Aggregate in Encoding Field Definition, Aggregate Transform, Supported Aggregation Operations, Argmin/Argmax
	Bin	Binning in Encoding Field Definition, Bin Transform, Bin Parameters, Ordinal Bin
	Join Aggregate	Join Aggregate Field Definition, Join Aggregate Transform Definition
	Stack	Stack in Encoding Field Definition, Stack Transform
	Window	Window Field Definition, Window Transform Definition, Window Only Operation Reference
	Other Transform	Calculate, Density, Extent, Filter, Flatten, Fold, Impute, Loess, Lookup, Pivot, Quantile, Regression, Sample
View	Schema	$schema
	Mark Type	Arc Properties/Config, Area Properties/Config, Bar Properties/Config, Boxplot Properties/Config, Circle Properties/Config, Errorband Properties/Config, Errorbar Properties/Config, Geoshape Properties/Config, Image Properties/Config, Line Properties/Config, Point Properties/Config, Rect Properties/Config, Rule Properties/Config, Square Properties/Config, Text Properties/Config, Tick Properties/Config, Trail Properties/Config
	Encoding Channels	Position Channels, Position Offset Channels, Polar Position Channels, Geographic Position Channels, Mark Property Channels, Text and Tooltip Channels, Hyperlink Channel, Description Channel, Level of Detail Channel, Key Channel, Order Channel, Facet Channels
	Other Encoding Functions	Band Position, Condition, Datum, Field, Format, Header, Impute, Type, Value
	Time Unit	Time Unit in Encoding Field Definition, Time Unit Transform, UTC Time, Time Unit Parameters
	Sort	Sorting Continuous Fields, Sorting Discrete Fields
	Scale	Scale Types, Scale Domains, Scale Ranges, Continuous Scales, Discrete Scales, Discretizing Scales, Disabling Scale
	Legends	Legend Types, Combined Legend, Legend Properties, Gradient, Labels, Symbols, Symbol Layout, Title
	Title	Alignment, Color, Font size
	Axis	Using Axis minExtent, Using Axis labelExpr, Using Axis tickBand, Customize Title, Grid, Conditional Axis Properties
	Width / Height	Specifying Fixed Width and Height, Specifying Responsive Width and Height, Step for Offset Channel, Specifying Width and Height per Discrete Step, Autosize, Width and Height of Multi-View Displays
Selection and Interaction	Bind	Input Element Binding, Legend Binding, Scale Binding
	Selection	Selection Projection with Encodings and Fields, Point Selection Properties, Interval Selection Properties
	Interaction	Using Parameters in Expression Strings,Using Parameters as Predicates,Using Parameters Data Extents
	Tooltip	Tooltip Based on Encoding, Tooltip Based on Data Point, Tooltip channel, Tooltip Image, Disable Tooltips
Composition and Layout	Layer	Layer
	Facet	Row-Facet, Wrapped Facet, Grid Facet
	Concatenate	Horizontal Concatenation, Vertical Concatenation, Wrappable Concatenation
	Repeat	Repeated Line Charts, Multi-series Line Chart, Repeated Histogram, Scatterplot Matrix
	Resolve	Scale Resolution, Axis Resolution, Legend Resolution