VisPath: Automated Visualization Code Synthesis via Multi-Path Reasoning and Feedback-Driven Optimization

Anonymous ACL submission

Abstract

Unprecedented breakthroughs in Large Language Models (LLMs) has amplified its penetration into application of automated visualization code generation. Few-shot prompting and query expansion techniques have notably enhanced data visualization performance, however, still fail to overcome ambiguity and complexity of natural language queries - imposing an inherent burden for manual human intervention. To mitigate such limitations, we propose a holistic framework ***VisPath : A Multi-Path Reasoning and Feedback-Driven Optimization Framework for Visualization Code Generation***, which systematically enhances code quality through structured reasoning and refinement. *VisPath* is a multi-stage framework, specially designed to handle underspecified queries. To generate a robust final visualization code, it first utilizes initial query to generate diverse reformulated queries via Chain-of-Thought (CoT) prompting, each representing a distinct reasoning path. Refined queries are used to produce candidate visualization scripts, consequently executed to generate multiple images. Comprehensively assessing correctness and quality of outputs, *VisPath* generates feedback for each image, which are then fed to aggregation module to generate optimal result. Extensive experiments on benchmarks including *MatPlotBench* and the *Qwen-Agent Code Interpreter Benchmark* show that *VisPath* significantly outperforms state-of-the-art (SOTA) methods, increased up to average 17%, offering a more reliable solution for AI-driven visualization code generation.

1 Introduction

Data visualization has long been an essential tool in data analysis and scientific research, enabling users to uncover patterns and relationships in complex datasets (Vondrick et al., 2013; Demiralp et al., 2017; Unwin, 2020; Li et al., 2024a). Traditionally, creating visualizations requires manually writing code using libraries such as Matplotlib, Seaborn, or D3.js (Barrett et al., 2005; Bisong and Bisong, 2019; Zhu, 2013). This approach demands programming expertise and significant effort to craft effective visual representations, which can be a barrier for many users (Bresciani and Eppler, 2015; Saket et al., 2018; Sharif et al., 2024). As datasets continue to grow in size and complexity, researchers have explored ways to automate visualization generation, aiming to make the process more efficient and accessible (Wang et al., 2015; Dibia and Demiralp, 2019; Qian et al., 2021).

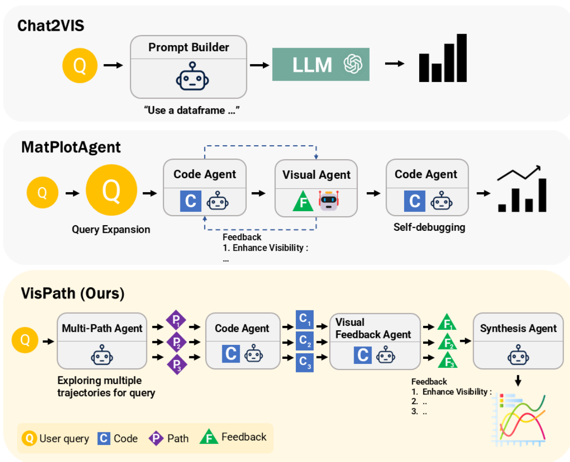

Figure 1: **Overview of different approaches for visualization code generation**. Comparing two baseline methods, namely *Chat2VIS* (Maddigan and Susnjak, 2023) and *MatPlotAgent* (Yang et al., 2024), with our proposed *VisPath* framework.

In response to this challenge, Large Language Models (LLMs) have emerged as a promising solution for simplifying visualization creation (Wang et al., 2023a; Han et al., 2023; Xie et al., 2024). By translating natural language instructions into executable code, LLM-based systems eliminate the need for extensive programming knowledge, allowing users to generate visualizations more intuitively (Xiao et al., 2023; Ge et al., 2023; Zhang et al., 2024b). Recent visualization methods such as ChartGPT (Tian et al., 2024) and NL4DV (Sah et al., 2024) demonstrate the potential of LLMs to provide interactive, conversational interfaces for visualization. These systems enable users to create complex charts with minimal effort, bridging the gap between technical expertise and effective data exploration (Dibia, 2023; Kim et al., 2024).

More recently, LLM-based visualization frameworks such as Chat2VIS (Maddigan and Susnjak, 2023) and MatPlotAgent (Yang et al., 2024) have been introduced to improve automated visualization code generation. Specifically, Chat2VIS follows a prefix-based approach, guiding LLMs to generate visualization code consistently; and MatPlotAgent expands the query before code generation. However, these methods face several limitations: ① both generate code in a single-path manner, limiting exploration of alternative solutions and unable to fix-out when caught in misleading bugs; ② both rely on predefined structures or examples which restrict adaptability to ambiguous or unconventional user queries. ③ both approaches encapsulate limitation in their inability to aggregate and synthesize multi-dimensional feedback. Without a mechanism to retrieve outputs that reflect diverse possibilities, they struggle to capture intricate details, ultimately limiting the precision and adaptability of the generated visualizations.

To address these limitations, we introduce *VisPath: A Branch Exploration Framework for Visualization Code Synthesis via Multi-Path Reasoning and Feedback-Driven Optimization*, a transformative approach that redefines how visualization code is generated. Traditional single-pass methods may seem efficient, but they often fall short of delivering the depth and precision users truly need. They generate code quickly, but struggle to capture the intricate details that make a visualization not just functional but meaningful. *VisPath* challenges this limitation by incorporating *Multi-Path Reasoning* and *Feedback-Driven Optimization*, systematically exploring multiple interpretative pathways to construct a more accurate, context-aware, and fully executable visualization.

Rather than simply translating user input into code, *VisPath* ensures that every critical aspect—both explicitly stated and implicitly necessary—is carefully considered, creating a visualization that is not only correct but insightful. At its core, it generates multiple reasoning paths that analyze the user's intent from different perspectives, producing structured blueprints that are then transformed into visualization scripts through Chain-of-Thought (CoT) prompting. These multiple candidates are evaluated using a Vision-Language Model (VLM) to assess accuracy, clarity, and alignment with the intended message. The results are then refined through an Aggregation Module, optimizing the final output for both reliability and impact. By shifting from a single-pass, one-size-fits-all approach to a dynamic, multi-layered process, *VisPath* sets a new standard for visualization generation—one that is not just about creating code, but about ensuring data is represented in its most clear, meaningful, and compelling form.

Extensive experiments on benchmark datasets, including *MatPlotBench* (Yang et al., 2024) and *Qwen-Agent Code Interpreter* Benchmark[1], demonstrate *VisPath*'s superiority over the state-of-the-art (SOTA) visualization generation methods. By systematically generating and evaluating multiple reasoning paths and leveraging iterative feedback aggregation, *VisPath* significantly enhances the accuracy, robustness against underspecified queries, and adaptability to diverse user intents. Our investigations demonstrate its ability to capture nuanced user intents, improve execution reliability, and minimize errors, making visualization code generation more accessible and effective for domains such as business intelligence, scientific research, and automated reporting.

2 Related Work

Numerous methods have been applied for Text-to-Visualization (Text2Vis) generation, which has significantly evolved over the years, adapting to new paradigms in data visualization and natural language processing (Dibia and Demiralp, 2019; Wu et al., 2022; Chen et al., 2022a,b; Rashid et al., 2022; Zhang et al., 2024a). Early approaches such as Voyager (Wongsuphasawat et al., 2015) and Eviza (Setlur et al., 2016) largely relied on rule-based systems, which mapped textual commands to predefined chart templates or specifications through handcrafted heuristics (de Araújo Lima and Diniz Junqueira Barbosa, 2020). While these methods demonstrated the feasibility of automatically converting text into visualizations (Moritz et al., 2018; Cui et al., 2019), they often required extensive do-

[1]https://github.com/QwenLM/Qwen-Agent/blob/main/benchmark/code_interpreter/README.md

main knowledge and struggled with more nuanced or ambiguous user requirements (Li et al., 2021; Wang et al., 2023b). Inspired by developments in deep learning, researchers began to incorporate neural networks to handle free-form natural language and broaden the range of supported visualization types (Liu et al., 2021; Luo et al., 2021).

Building on these machine learning strategies, numerous studies have utilized Large Language Models (LLMs) to further enhance system flexibility. Recent frameworks such as Chat2VIS (Maddigan and Susnjak, 2023) and Prompt4Vis (Li et al., 2024b) utilize few-shot learning or query expansion to refine user queries, subsequently generating Python visualization scripts through instruction-based prompting. More recent approaches, such as MatPlotAgent (Yang et al., 2024) and PlotGen (Goswami et al., 2025), extend these frameworks by integrating a vision-language feedback model to iteratively optimize the final code based on evaluations of the rendered visualizations. The aforementioned approaches often struggle to effectively capture user intent in complex visualization tasks. By committing to a single reasoning trajectory, they may produce code that is syntactically correct yet semantically misaligned with user expectations, requiring extensive manual adjustments. This challenge is particularly pronounced when user input is ambiguous or underspecified, leading to an iterative cycle of prompt refinement and code modification—ironically undermining the intended efficiency of automation. To address these limitations, we introduce *VisPath*, a novel framework that integrates multi-path reasoning with feedback from Vision-Language Models (VLMs) to enhance visualization code generation. Compared to conventional methods constrained by a single reasoning path, *VisPath* dynamically explores multiple solution pathways, improving interpretability, minimizing manual intervention, and adapting more effectively to diverse visualization requirements.

3 Methodology

We introduce *VisPath*, a framework for robust visualization code generation that leverages diverse reasoning and visual feedback. VisPath is built on three core components: (1) *Multi-Path Query Expansion*, which generates multiple reasoning paths informed by the dataset description; (2) *Code Generation from Expanded Queries*, which synthesizes candidate visualization scripts via Chain-of-Thought (CoT) prompting while grounding them in the actual data context; and (3) *Feedback-Driven Code Optimization*, where a Vision-Language Model (VLM) evaluates and refines the outputs to ensure generation robust visualization code. An overview of this process is shown in Figure 2.

3.1 Multi-Path Generation

One of the biggest pitfalls in visualization code generation is rigid interpretation, a single query can have multiple valid visual representations depending on its dataset structure. *VisPath* mitigates this by generating multiple extended queries within a single interaction. Given a user query Q and a corresponding dataset description D, a *Multi-Path Agent* is employed to expand the query into K distinct reasoning pathways:

$$\{R_1, R_2, \ldots, R_K\} = f_{\text{mpa}}(Q, D), \tag{1}$$

where f_{mpa} denotes the function of the Multi-Path Agent implemented via an LLM. The dataset description D plays a crucial core in shaping these reasoning paths by providing contextual information about variable types, inherent relationships, and the suitability of different chart types for a more grounded interpretation of the query. Each R_i serves as a detailed logical blueprint outlining one possible approach to fulfill the visualization request. This design ensures that our framework effectively considers a broad range of potential interpretations, thereby increasing the quality of reasoning and the likelihood of capturing the true user intent—even when queries are ambiguous or underspecified.

3.2 Code Generation from Reasoning Paths

Once diverse reasoning paths are established, the next stage involves translating each path into executable Python scripts. For every reasoning path R_i generated in Equation (1), a dedicated Code Generation LLM produces the corresponding visualization code using chain-of-thought (CoT) prompting:

$$C_i = f_{\text{code}}(D, R_i), \tag{2}$$

where f_{code} represents the code generation function. Unlike naive code generation approaches, here, the dataset description D is explicitly provided to ground the generated code in the actual data context, ensuring that variable names, data types, and visualization parameters align correctly

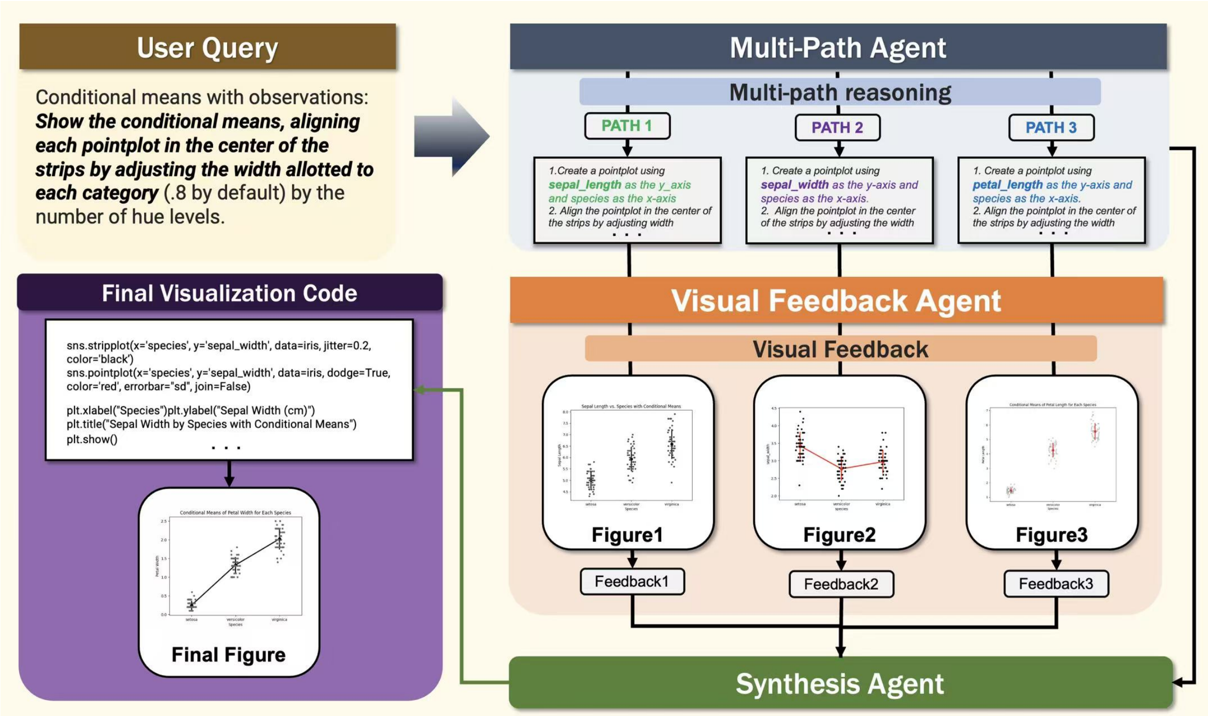

Figure 2: **Overview of the proposed *VisPath* framework for creating robust visualization code generation.** The framework consists of combination of Multi-Path Agent, Visual Feedback Agent, and Synthesis Agent.

with the underlying data attributes. The generated code C_i is then executed to render a candidate visualization:

$$V_i = f_{\text{exec}}(C_i), \quad i = 1, 2, \ldots, K, \tag{3}$$

with f_{exec} serving as the code execution function. By executing the code directly, we ensure that the visualization accurately reflects the intended operations without reintroducing the dataset description D at this stage. In practice, some generated codes may not be executable. Rather than engaging in an explicit debugging loop, we record the execution status as a binary executability indicator:

$$\epsilon_i = \begin{cases} 1, & \text{if } C_i \text{ is executable} \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

To route the outputs appropriately, we introduce:

$$Z_i = \begin{cases} \text{plot image}(V_i), & \text{if } \epsilon_i = 1, \\ \text{error message from } C_i, & \text{if } \epsilon_i = 0. \end{cases} \tag{5}$$

The result Z_i (either the rendered visualization or the error message) is then provided, along with C_i and the original query Q, to the feedback model in the subsequent stage.

3.3 Feedback-Driven Code Optimization

Most code generation frameworks focus merely on syntactically correct scripts, but our framework goes further. As final stage, *VisPath* synthesizes the most robust and accurate visualization code by leveraging both the executability information and structured visual feedback. A Vision-Language Model (VLM) is employed to analyze each candidate by evaluating the initial query Q, the generated code C_i, and the routed output Z_i. This evaluation is formalized as:

$$F_i = f_{\text{feedback}}(Q, C_i, Z_i), \tag{6}$$

where F_i provides structured feedback on key aspects such as chart layout, the alignment between the intended request and the rendered visualization (or error context), and visual readability (including potential improvements). To capture the complete quality signal from each candidate, we pair the feedback with its corresponding generated code:

$$S_i = (C_i, F_i), \quad i = 1, 2, \ldots, K. \tag{7}$$

Leveraging the collective code-feedback pairs along with the original query Q and the dataset description D, an *Integration Module* synthesizes the final, refined visualization code:

$$C^* = f_{\text{integrate}}\left(Q, D, \{S_i\}_{i=1}^{K}\right), \tag{8}$$

where C^* represents the optimized visualization code and $f_{\text{integrate}}$ denotes the function that aggregates the strengths of each candidate code alongside its corresponding feedback. This formulation ensures that the final code is not only constructed based on the insights extracted from the candidate outputs but is also meticulously aligned with the original user query and the provided dataset description. The process harnesses the reasoning capabilities of LLM to systematically evaluate the strengths and pinpoint potential weaknesses across all candidate solutions, including those that may initially be non-executable. By synthesizing the most optimal elements from each candidate, the framework generates a final code that best captures the user's intent while maintaining high standards of reliability, robustness, and execution quality. A step-by-step breakdown of this process is detailed in Algorithm 1 below:

Algorithm 1 *Algorithm for VisPath*

Require: User query Q, dataset D, number of reasoning paths K, Multi-Path Agent f_{mpa}, Code Generation LLM f_{code}, Code Execution Function f_{exec}, Feedback Model f_{feedback}, Integration Module $f_{\text{integrate}}$

Ensure: Final visualization code C^*

```
// Step 1: Multi-Path Query Expansion
{R_1, R_2, ..., R_K} ← f_mpa(Q, D)    ▷ Generate K distinct reasoning paths; each R_i outlines a potential interpretation of Q given D.
// Step 2: Code Generation from Reasoning Paths
for i = 1 to K do
    C_i ← f_code(D, R_i)    ▷ Generate candidate visualization code C_i based on reasoning path R_i.
    V_i ← f_exec(C_i)    ▷ Execute C_i to obtain candidate visualization V_i.
    ε_i ← { 1, if C_i executes successfully; 0, otherwise }    ▷ Record executability indicator ε_i for C_i.
    Z_i ← { plot image(V_i), if ε_i = 1; error message from C_i, if ε_i = 0 }    ▷ Route output: either the rendered visualization or the error message.
end for
// Step 3: Feedback-Driven Code Optimization
for i = 1 to K do
    F_i ← f_feedback(Q, C_i, Z_i)    ▷ Obtain structured feedback F_i on candidate code C_i using the routed output Z_i.
    S_i ← (C_i, F_i)    ▷ Form a unified tuple S_i combining C_i, F_i.
end for
C* ← f_integrate(Q, D, {S_1, S_2, ..., S_K})    ▷ Aggregate the code-feedback pairs along with Q and D to synthesize the final visualization code C*.
return C*
```

4 Experiments

4.1 Setup

In this section, we detail our experimental configuration, including experimental datasets, model specifications, and baseline methods for evaluating the performance of the proposed *VisPath* framework in generating visualization code from natural language queries.

4.1.1 Experimental Datasets

We evaluate our approach on two Text-to-Visualization benchmarks: *MatPlotBench* (Yang et al., 2024) and the *Qwen-Agent Code Interpreter Benchmark*. *MatPlotBench* comprises 100 items with ground truth images; we focus on its simple instruction subset for nuanced queries. In contrast, the *Qwen-Agent Code Interpreter Benchmark* includes 295 records: 163 related to visualization, and evaluates Python code interpreters on tasks such as mathematical problem solving, data visualization, and file handling based on Code Executability and Code Correctness.

4.1.2 Models Used

Large Language Models (LLMs): For the code inference stage, we experiment with *GPT-4o mini* (Achiam et al., 2023) and *Gemini 2.0 Flash* (Team et al., 2024) to generate candidate visualization code from the reasoning paths. Both models are configured with a temperature of 0.2 to ensure precise and focused outputs, in line with recommendations from previous work (Yang et al., 2024). To evaluate the generated code quality and guide the subsequent optimization process, we utilize *GPT-4o* (Achiam et al., 2023) and *Gemini 2.0 Flash* (Team et al., 2024) as our visualization feedback model, which provides high-quality reference assessments.

Vision Language Models (VLMs): In order to assess the visual quality and correctness of the rendered plots, we incorporate vision evaluation models into our framework. Specifically, *GPT-4o* (Achiam et al., 2023) is employed for detailed plot evaluation in all evaluation tasks. This setup ensures the thorough evaluation of both the syntactic correctness of the code and the aesthetic quality of the resulting visualizations.

4.1.3 Evaluation Metrics

In our experiments, we utilized evaluation metrics introduced by previous work to ensure consistency

Model	Methods	MatPlotBench		Qwen-Agent Code Interpreter benchmark		
		Plot Score	Executable Rate (%)	Visualization-Hard	Visualization-Easy	Avg.
GPT-4o mini	Zero-Shot	62.38	53	59.68	45.50	52.59
	CoT Prompting (Wei et al., 2022)	61.95	50	57.50	40.00	48.75
	Chat2VIS (Maddigan and Susnjak, 2023)	56.98	53	59.36	36.50	47.93
	MatPlotAgent (Yang et al., 2024)	63.90	58	67.50	53.25	60.38
	VisPath† (Ours)	**66.12**	**60**	**70.68**	**57.23**	**63.96**
Gemini 2.0 Flash	Zero-Shot	55.00	54	68.97	52.18	60.58
	CoT Prompting (Wei et al., 2022)	53.56	61	40.00	**63.89**	51.95
	Chat2VIS (Maddigan and Susnjak, 2023)	54.89	55	59.36	56.50	57.93
	MatPlotAgent (Yang et al., 2024)	56.31	58	77.62	51.50	64.56
	VisPath† (Ours)	**59.37**	**63**	**80.79**	57.17	**68.98**

Table 1: **Performance comparison of various methods across different benchmarks.** Zero-Shot refers to directly generating code. CoT Prompting utilizes Chain of Thought Prompting. Visualization-Hard and Visualization-Easy refer to the Accuracy of Code Execution Results on different subsets of the Qwen-Agent Code Interpreter benchmark. **Bold text** indicates the best performance, underlined text indicates the second-best performance. † denotes our proposed method.

and comparability. *MatPlotBench* assesses graph generation models using two key metrics: *Plot Score*, which measures similarity to the Ground Truth (0–100), and *Executable Score*, which represents the percentage of error-free code executions. *Qwen-Agent Code Interpreter benchmark* evaluates visualization models based on *Visualization-Hard* and *Visualization-Easy*, measuring how well generated images align with queries of different difficulty levels. Compared to *MatPlotBench*, *Qwen-Agent Code Interpreter benchmark* assesses image alignment via a code correctness metric. Previous studies show GPT-based VLM evaluations align with human assessments, hence VLM was used for evaluation.

4.1.4 Baseline Methods

We compare *VisPath* against competitive baselines. Specifically, (1) *Zero-Shot* directly generates visualization code without intermediate reasoning, (2) *CoT Prompting* uses chain-of-thought prompting to articulate its reasoning, while (3) *Chat2VIS* (Maddigan and Susnjak, 2023) employs guiding prefixes to mitigate ambiguity, and (4) *MatPlotAgent* (Yang et al., 2024) first expands the query and then refines the code via a self-debugging loop with feedback. For a fair comparison, *MatPlotAgent* is limited to three iterations, and uses critique-based debugging loop as well, and *VisPath* generates three reasoning paths with corresponding visual feedback to refine the final output.[2]

[2]Prompts are detailed in Appendix A.

4.2 Experimental Analysis

In this section, we present a detailed evaluation of our proposed *VisPath* framework, summarizing the performance results against four baselines: (1) *Zero-Shot*, (2) *Chain-of-Thought (CoT) Prompting*, (3) *Chat2VIS*, and (4) *MatPlotAgent*.

From Table 1, we find that the *Zero-Shot* method, which directly generates visualization code from the user query without intermediate reasoning, suffers from ambiguous interpretations and error-prone outputs. These issues are particularly pronounced when the input queries are underspecified, contain implicit assumptions, or require complex reasoning steps. *CoT Prompting*, on the other hand, mitigates some of these challenges by employing chain-of-thought (CoT) reasoning, which helps the model articulate its decision-making process step by step. While this structured reasoning approach improves interpretability and correctness, its reliance on a single reasoning trajectory limits its capacity to explore alternative solutions.

Meanwhile, *Chat2VIS* builds upon *CoT Prompting* by incorporating guided prefixes to better structure user queries and disambiguate input intent. This enhancement leads to more coherent code generation and reduces errors stemming from unclear specifications. However, its effectiveness still depends on the predefined guiding templates, which may not fully adapt to highly variable or novel query structures. *MatPlotAgent* further refines this approach through query expansion and an iterative self-debugging loop that enhances code robustness. However, its reliance on an iterative correction mechanism comes at the cost of computational efficiency and does not fully leverage diverse reasoning strategies that could lead to more creative and contextually accurate solutions.

In contrast, our novel framework, *VisPath*, overcomes these challenges by dynamically generating multiple reasoning paths. By exploring diverse interpretations of user intent simultaneously, *VisPath* significantly enhances the robustness of the generated visualization code. Furthermore, it integrates structured feedback from Vision-Language Models (VLMs) to refine its outputs, ensuring both higher execution accuracy and improved aesthetic quality. This feedback-driven optimization resolves ambiguities more effectively and selects the most appropriate solution from a wide range of possibilities. Evaluations conducted on benchmark datasets such as *MatPlotBench* and *Qwen-Agent Code Interpreter benchmark* demonstrate that *VisPath* consistently outperforms the baseline methods by average 17%. *VisPath* exhibits a superior ability to handle complex and ambiguous visualization requests by synthesizing multiple reasoning trajectories and incorporating structured feedback, making it a highly effective tool for automated visualization generation.

4.3 Ablation Study

This section further explores the impact of varying the number of reasoning paths on the performance of visualization code generation. We conduct a series of ablation studies to demonstrate the robustness of our *VisPath* framework under alternative settings. Specifically, we investigate (i) the effect of varying the number of generated reasoning paths and (ii) the impact of a simplified integration strategy for synthesizing the final visualization code.

4.3.1 Varying the Number of Reasoning Paths

To further assess the contribution of reasoning path diversity, we conducted ablation experiments varying the number of generated paths $K \in \{2, 3, 4\}$ to evaluate their effects on execution accuracy, visualization quality and interpretability. Evaluation has been conducted on both the *MatPlotBench* and *Qwen-Agent Code Interpreter benchmark* datasets. The study reveals an interesting pattern observed when the number of reasoning path is reduced. As illustrated in Table 2 and Table 3, when $K = 2$, the system exhibited limited diversity in which occasionally results in missing nuanced interpretations of complex user queries.

Conversely, increasing K to 4 introduces extra paths which can sometimes add noise as less relevant interpretations are considered. Specifically, we observed cases where redundant or overly complex visualization components were generated, hence making the final output harder to be interpreted. While the additional reasoning paths increased the overall exploration space, they also required more extensive filtering and selection, leading to suboptimal execution efficiency.

Model	K	MatPlotBench	
		Plot Score	Executable Rate (%)
GPT-4o mini	2	64.02	58
	3	**66.12**	**60**
	4	64.68	62
Gemini 2.0 Flash	2	56.59	56
	3	**59.37**	**61**
	4	54.60	54

Table 2: **Results of ablation study on MatPlotBench.**

Model	K	Qwen-Agent		
		Hard	Easy	Avg.
GPT-4o mini	2	67.50	55.72	61.61
	3	**70.68**	**57.23**	**63.96**
	4	66.42	50.25	58.34
Gemini 2.0 Flash	2	74.21	57.50	65.86
	3	**80.79**	**57.17**	**68.98**
	4	76.41	54.50	65.46

Table 3: **Results of ablation study on Qwen-Agent Code Interpreter benchmark.**

Notably, $K = 3$ consistently achieves the best balance by providing sufficient diversity without introducing excessive noise. With too few reasoning paths, some nuanced aspects of ambiguous queries are lost, while too many paths configuration result in unnecessary exploration of low relevancy solutions. The number of reasoning path diversity set as $K = 3$ optimally provides sufficient diversity without overcomplicating the reasoning process.

4.3.2 Robustness with a Simple Integration Strategy

To further validate the robustness of *VisPath*, we also evaluate an alternative integration strategy that simplicies the aggregation of multiple reasoning paths. In contrast to our full feedback-driven iterative approach which refines visualization code through multiple rounds of vision-language feedback, this streamlined method directly aggregates candidate outputs from different reasoning trajectories without intermediate corrections.

Specifically, in this alternative strategy, three candidate codes, each derived from different reasoning paths, are directly combined to generate the final visualization output. This methodology reduces computational overhead and processing time while maintaining a degree of interpretability and correctness. Through examining the impact of this approach, we gain insight into the extent to which *VisPath's* core strength stems from its multi-path reasoning capability versus its iterative refinement process.

Model	Feedback	MatPlotBench		Qwen-Agent
		Plot Score	Executable Rate (%)	Avg.
GPT-4o mini	(w/o) feedback	63.76	56	58.00
	(w) feedback	**66.12**	**60**	**63.96**
Gemini 2.0 Flash	(w/o) feedback	55.28	57	64.03
	(w) feedback	**59.37**	**63**	**68.98**

Table 4: **Performance comparison of VisPath with and without visual feedback**. The MatPlotBench scores (Plot Score and Executable Rate) and the average score from the Qwen-Agent Code Interpreter benchmark are shown for two llm models.

As shown in Table 4, even under this simplified integration, the performance of *VisPath* significantly outperforms all baseline methods, demonstrating its robustness and adaptability. This result suggests that the diversity of reasoning paths alone ensures a resilient and effective outcome, even in the absence of extensive intermediate corrections. This finding shows that the strength of our framework largely stems from its multi-path reasoning design. Even when the integration is simplified, the diversity of reasoning paths ensures that the aggregated output remains robust and effective, thus showing the overall adaptability and effectiveness of our approach.

5 Discussion

Our main experiment and ablation studies decisively demonstrate that *VisPath's* multi-path exploration revolutionizes visualization code generation by making it not just robust, but truly interpretable. [3] Unlike traditional methods that often yield simplistic, one-dimensional results, *VisPath* embraces complexity—delving deeper to capture nuances of user intent that would otherwise be lost. By integrating multiple perspectives during reasoning, our framework ensures that final visualizations are not only data-accurate but also rich in context, inherently intuitive, and unmistakably aligned with the user's true objectives.

The real breakthrough with *VisPath* is how it transforms visualization into a structured yet fluid process. Traditional methods often act like static translators, converting user queries into plots while neglecting the vital auxiliary components—legends, labels, line styles, and other elements—that enable true interpretability. They assume that a single pass can sufficiently capture the essence of a visualization, but this leads to visualizations that might technically be correct but fail at communication. In contrast, *VisPath*'s multi-path approach systematically explores different ways to represent data, integrating feedback at each step. This ensures that even subtle yet critical design elements are not just included, but optimized, making the resulting visualizations feel complete, polished, and immediately comprehensible.

Beyond accuracy, *VisPath* introduces a level of adaptability that is indispensable in today's fast-evolving data landscape. It's not just a tool for static reports—it is built for real-time streaming data, dynamic dashboards, and interactive analytics. Even with broad or loosely defined prompts, our framework intelligently synthesizes high-quality visualizations, closing the gap between intent and execution. As industries continue to demand clearer, faster, and more adaptable insights, VisPath is poised to lead the way, setting a new standard for intelligent, context-aware data visualization.

6 Conclusion

In this work, we present *VisPath*, a groundbreaking framework that redefines the landscape of automated visualization code generation over existing methods. Unlike prior methods, our approach seaminglessly combines multi-path reasoning with feedback-driven optimization. Accurately capturing diverse user intents and refine generated code, *VisPath* achieves notable improvements in both execution success and visual quality on challenging benchmarks such as *MatPlotBench* and the *Qwen-Agent Code Interpreter Benchmark*. By prioritizing adapability and user intent alignment, *VisPath* is uniquely positioned to handle compledities of real-world data visualization tasks. Looking ahead, future work could explore *VisPath*'s adaptability in more dynamic, real-world visualization scenarios, further broadening its scope and enhancing its practical utility in complex data analysis contexts.

[3] Case study is provided in Appendix B.

7 Limitations

Despite its effectiveness, the current iteration of our framework relies on a limited feedback mechanism, evaluating visualizations primarily based on two aspects: query-code alignment and query-plot image alignment. While these aspects provide valuable insights, they may not fully capture the granular elements that contribute to overall interpretability. To enhance the depth of feedback itself, future research could focus on decomposition of assessment process, assessing individual plot components separately. This granular analysis would render for a more nuanced assessment of readability, element appropriateness and visual coherence, ultimately leading to more refined visualization code generation. Strengthening the feedback mechanism in such way will be crucial for maximizing *VisPath's* effectiveness in diverse and complex visualization scenarios.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Paul Barrett, John Hunter, J Todd Miller, J-C Hsu, and Perry Greenfield. 2005. matplotlib–a portable python plotting package. In *Astronomical data analysis software and systems XIV*, volume 347, page 91.

Ekaba Bisong and Ekaba Bisong. 2019. Matplotlib and seaborn. *Building machine learning and deep learning models on google cloud platform: A comprehensive guide for beginners*, pages 151–165.

Sabrina Bresciani and Martin J Eppler. 2015. The pitfalls of visual representations: A review and classification of common errors made while designing and interpreting visualizations. *Sage Open*, 5(4):2158244015611451.

Qiaochu Chen, Shankara Pailoor, Celeste Barnaby, Abby Criswell, Chenglong Wang, Greg Durrett, and Işil Dillig. 2022a. Type-directed synthesis of visualizations from natural language queries. *Proceedings of the ACM on Programming Languages*, 6(OOPSLA2):532–559.

Yiru Chen, Ryan Li, Austin Mac, Tianbao Xie, Tao Yu, and Eugene Wu. 2022b. Nl2interface: Interactive visualization interface generation from natural language queries. *arXiv preprint arXiv:2209.08834*.

Weiwei Cui, Xiaoyu Zhang, Yun Wang, He Huang, Bei Chen, Lei Fang, Haidong Zhang, Jian-Guan Lou, and Dongmei Zhang. 2019. Text-to-viz: Automatic generation of infographics from proportion-related natural language statements. *IEEE transactions on visualization and computer graphics*, 26(1):906–916.

Raul de Araújo Lima and Simone Diniz Junqueira Barbosa. 2020. Vismaker: a question-oriented visualization recommender system for data exploration. *arXiv e-prints*, pages arXiv–2002.

Çağatay Demiralp, Peter J Haas, Srinivasan Parthasarathy, and Tejaswini Pedapati. 2017. Foresight: Recommending visual insights. *arXiv preprint arXiv:1707.03877*.

Victor Dibia. 2023. Lida: A tool for automatic generation of grammar-agnostic visualizations and infographics using large language models. *arXiv preprint arXiv:2303.02927*.

Victor Dibia and Çağatay Demiralp. 2019. Data2vis: Automatic generation of data visualizations using sequence-to-sequence recurrent neural networks. *IEEE computer graphics and applications*, 39(5):33–46.

Yan Ge, Victor Junqiu Wei, Yuanfeng Song, Jason Chen Zhang, and Raymond Chi-Wing Wong. 2023. Automatic data visualization generation from chinese natural language questions. *arXiv preprint arXiv:2309.07650*.

Kanika Goswami, Puneet Mathur, Ryan Rossi, and Franck Dernoncourt. 2025. PlotGen: Multi-Agent LLM-based Scientific Data Visualization via Multimodal Feedback. *Preprint*, arXiv:2502.00988.

Yucheng Han, Chi Zhang, Xin Chen, Xu Yang, Zhibin Wang, Gang Yu, Bin Fu, and Hanwang Zhang. 2023. Chartllama: A multimodal llm for chart understanding and generation. *arXiv preprint arXiv:2311.16483*.

Jaeyoung Kim, Sihyeon Lee, Hyeon Jeon, Keon-Joo Lee, Hee-Joon Bae, Bohyoung Kim, and Jinwook Seo. 2024. Phenoflow: A human-llm driven visual analytics system for exploring large and complex stroke datasets. *IEEE Transactions on Visualization and Computer Graphics*.

Guozheng Li, Xinyu Wang, Gerile Aodeng, Shunyuan Zheng, Yu Zhang, Chuangxin Ou, Song Wang, and Chi Harold Liu. 2024a. Visualization generation with large language models: An evaluation. *arXiv preprint arXiv:2401.11255*.

Haotian Li, Yong Wang, Songheng Zhang, Yangqiu Song, and Huamin Qu. 2021. Kg4vis: A knowledge graph-based approach for visualization recommendation. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):195–205.

Shuaimin Li, Xuanang Chen, Yuanfeng Song, Yunze Song, and Chen Zhang. 2024b. Prompt4Vis: Prompting Large Language Models with Example Mining and Schema Filtering for Tabular Data Visualization. *Preprint*, arXiv:2402.07909.

Can Liu, Yun Han, Ruike Jiang, and Xiaoru Yuan. 2021. Advisor: Automatic visualization answer for natural-language question on tabular data. In *2021 IEEE 14th Pacific Visualization Symposium (PacificVis)*, pages 11–20. IEEE.

Yuyu Luo, Nan Tang, Guoliang Li, Jiawei Tang, Chengliang Chai, and Xuedi Qin. 2021. Natural language to visualization by neural machine translation. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):217–226.

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

Dominik Moritz, Chenglong Wang, Greg L Nelson, Halden Lin, Adam M Smith, Bill Howe, and Jeffrey Heer. 2018. Formalizing visualization design knowledge as constraints: Actionable and extensible models in draco. *IEEE transactions on visualization and computer graphics*, 25(1):438–448.

Xin Qian, Ryan A Rossi, Fan Du, Sungchul Kim, Eunyee Koh, Sana Malik, Tak Yeon Lee, and Joel Chan. 2021. Learning to recommend visualizations from data. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pages 1359–1369.

Md Mahinur Rashid, Hasin Kawsar Jahan, Annysha Huzzat, Riyasaat Ahmed Rahul, Tamim Bin Zakir, Farhana Meem, Md Saddam Hossain Mukta, and Swakkhar Shatabda. 2022. Text2chart: A multi-staged chart generator from natural language text. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 3–16. Springer.

Subham Sah, Rishab Mitra, Arpit Narechania, Alex Endert, John Stasko, and Wenwen Dou. 2024. Generating analytic specifications for data visualization from natural language queries using large language models. *arXiv preprint arXiv:2408.13391*.

Bahador Saket, Alex Endert, and Çağatay Demiralp. 2018. Task-based effectiveness of basic visualizations. *IEEE transactions on visualization and computer graphics*, 25(7):2505–2512.

Vidya Setlur, Sarah E Battersby, Melanie Tory, Rich Gossweiler, and Angel X Chang. 2016. Eviza: A natural language interface for visual analysis. In *Proceedings of the 29th annual symposium on user interface software and technology*, pages 365–377.

Ather Sharif, Joo Gyeong Kim, Jessie Zijia Xu, and Jacob O Wobbrock. 2024. Understanding and reducing the challenges faced by creators of accessible online data visualizations. In *Proceedings of the 26th International ACM SIGACCESS Conference on Computers and Accessibility*, pages 1–20.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

Yuan Tian, Weiwei Cui, Dazhen Deng, Xinjing Yi, Yurun Yang, Haidong Zhang, and Yingcai Wu. 2024. Chartgpt: Leveraging llms to generate charts from abstract natural language. *IEEE Transactions on Visualization and Computer Graphics*.

Antony Unwin. 2020. Why is data visualization important? what is important in data visualization? *Harvard Data Science Review*, 2(1):1.

Carl Vondrick, Aditya Khosla, Tomasz Malisiewicz, and Antonio Torralba. 2013. Hoggles: Visualizing object detection features. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1–8.

Chenglong Wang, John Thompson, and Bongshin Lee. 2023a. Data formulator: Ai-powered concept-driven visualization authoring. *IEEE Transactions on Visualization and Computer Graphics*.

Lei Wang, Songheng Zhang, Yun Wang, Ee-Peng Lim, and Yong Wang. 2023b. Llm4vis: Explainable visualization recommendation using chatgpt. *arXiv preprint arXiv:2310.07652*.

Lidong Wang, Guanghui Wang, and Cheryl Ann Alexander. 2015. Big data and visualization: methods, challenges and technology progress. *Digital Technologies*, 1(1):33–38.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2015. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE transactions on visualization and computer graphics*, 22(1):649–658.

Chenfei Wu, Jian Liang, Lei Ji, Fan Yang, Yuejian Fang, Daxin Jiang, and Nan Duan. 2022. Nüwa: Visual synthesis pre-training for neural visual world creation. In *European conference on computer vision*, pages 720–736. Springer.

Shishi Xiao, Suizi Huang, Yue Lin, Yilin Ye, and Wei Zeng. 2023. Let the chart spark: Embedding semantic context into chart with text-to-image generative model. *IEEE Transactions on Visualization and Computer Graphics*.

Yupeng Xie, Yuyu Luo, Guoliang Li, and Nan Tang. 2024. Haichart: Human and ai paired visualization system. *arXiv preprint arXiv:2406.11033*.

Zhiyu Yang, Zihan Zhou, Shuo Wang, Xin Cong, Xu Han, Yukun Yan, Zhenghao Liu, Zhixing Tan, Pengyuan Liu, Dong Yu, et al. 2024. Matplotagent: Method and evaluation for llm-based agentic scientific data visualization. *arXiv preprint arXiv:2402.11453*.

Songheng Zhang, Lei Wang, Toby Jia-Jun Li, Qiaomu Shen, Yixin Cao, and Yong Wang. 2024a. Chartifytext: Automated chart generation from data-involved texts via llm. *arXiv preprint arXiv:2410.14331*.

Zhehao Zhang, Weicheng Ma, and Soroush Vosoughi. 2024b. Is gpt-4v (ision) all you need for automating academic data visualization? exploring vision-language models' capability in reproducing academic charts. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 8271–8288.

Nick Qi Zhu. 2013. *Data visualization with D3. js cookbook*. Packt Publishing Ltd.

This appendix briefly outlines our additional methodologies and experimental setups, detailing the prompts used for *VisPath*, providing a statistical overview of the experimental data, and presenting a case study illustrating our approach in practice.

A Appendix A. Prompts Used

Prompt for *Multi-Path Reasoning*

[**System Prompt**] According to the user query, expand and solidify the query into detailed instruction on how to write python code to fulfill the user query's requirements. Import the appropriate libraries. Pinpoint the correct library functions to call and set each parameter in every function call accordingly.
[**User Prompt**] Think step by step. Generate three distinct extended queries based on the given query. Ensure that you first analyze the given data description and create queries that align with the data. If no data description is provided, follow the original query as is. You must follow the Python list format for the output. Do not modify the detailed instructions from the original user query. Original query: ori_query Data description: data_description
Output format: [extended_path_1, extended_path_2, extended_path_3]

Prompt for *Code Generation*

[**System Prompt**] You are an expert in data visualization code generation. Think step by step and write the generated code in the format "'python..."', where "..." represents the generated code. The code must end with 'plt.show()'.
[**User Prompt**] Think step by step. Based on the user's query and the provided data description, generate Python code using 'matplotlib.pyplot' and 'seaborn' to create the requested plot. Ensure that the code is formatted as "'..."', where "..." represents the generated code.
User query: {query}
Data description: {data_description}

Prompt for *Visual Feedback*

[**System Prompt**] Given a code, a user query, and an image of the current plot, please determine whether the plot accurately follows the user query. Provide detailed instructions on how to enhance the plot using Python code.
[**User Prompt**] Carefully analyze the provided Python code, the user query, and the plot image (if available) to assess whether the generated plot meets the user query requirements. If the plot image is missing, check the error message that occurred in the code. Compare the plot with the user query, highlight discrepancies, and provide clear, actionable instructions to modify the code. Additionally, suggest improvements for better visualization, focusing on clarity, readability, and alignment with the user's objectives.
Code: {code}
User query: {ori_query}

Prompt for *Synthesis*

[**System Prompt**] You are an expert on data visualization code judgement and aggregation.
[**User Prompt**] Think step by step. Given the provided user query, data description, multiple data visualization codes generated for the query, and feedback for each code's generated image. Your task is to:
1. Carefully review the user query and the data description.
2. Examine each version of the data visualization code along with the feedback provided for each version.
3. Synthesize the feedbacks for each code, user query insights, data description to create a final version of the code.
4. Your goal is to produce a final version of code that more effectively fulfills the user query by integrating the best elements from all versions and applying necessary corrections.
User Query: {ori_query}
Data Description: {data_description}
Code for aggregation with corresponding feedback: {code_for_aggregation}

Prompt for Evaluation: *MatplotBench*

You are an excellent judge at evaluating visualization plots between a model generated plot and the ground truth. You will be giving scores on how well it matches the ground truth plot.
The generated plot will be given to you as the first figure. If the first figure is blank, that means the code failed to generate a figure. Another plot will be given to you as the second figure, which is the desired outcome of the user query, meaning it is the ground truth for you to reference. Please compare the two figures head to head and rate them. Suppose the second figure has a score of 100, rate the first figure on a scale from 0 to 100. Scoring should be carried out in the following aspect:
1. Plot correctness:
Compare closely between the generated plot and the ground truth, the more resemblance the generated plot has compared to the ground truth, the higher the score. The score should be proportionate to the resemblance between the two plots. In some rare occurrence, see if the data points are generated randomly according to the query, if so, the generated plot may not perfectly match the ground truth, but it is correct nonetheless.
Only rate the first figure, the second figure is only for reference.
If the first figure is blank, that means the code failed to generate a figure.
Give a score of 0 on the Plot correctness.
After scoring from the above aspect, please give a final score. The final score is preceded by the [FINAL SCORE] token.
For example [FINAL SCORE]: 40.

Prompt for Evaluation: *Qwen-Agent Code Interpreter benchmark*

Please judge whether the image is consistent with the [Question] below, if it is consistent then reply "right", if not then reply "wrong".
Question: {query}

B Appendix B. Case Study

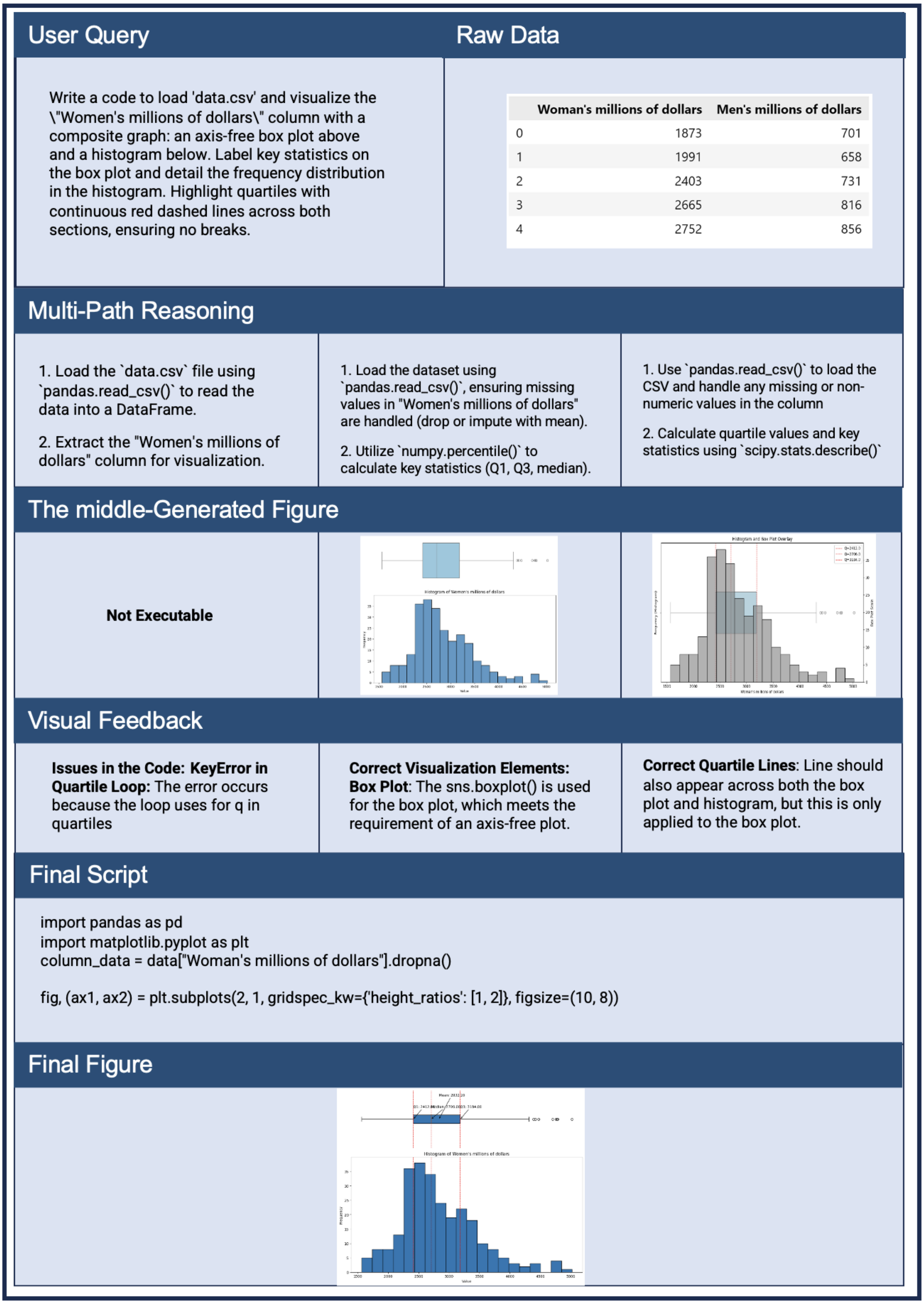

Figure 3: **Overview of end-to-end process of visualizing women's million-dollar earnings data using *VisPath*.** It includes the user query, raw data, multi-path reasoning, generated figures, visual feedback, final script, and the completed visualization.