Follow the Flow: Fine-grained Flowchart Attribution with Neurosymbolic Agents

Anonymous ACL submission

Abstract

Flowcharts are a critical tool for visualizing decision-making processes, yet their nonlinear structure and complex visual-textual relationships make them challenging to interpret automatically. Vision language models frequently hallucinate non-existent connections and decision paths when analyzing these diagrams, compromising the reliability of automated flowchart processing. We introduce the task of Fine-grained Flowchart Attribution, a post-hoc strategy to mitigate visual flowchart hallucination by tracing specific components grounding a flowchart referring statement. Flowchart Attribution ensures the verifiability of model predictions and enhances explainability by linking generated responses to the flowchart's structure. We introduce `FlowExplainBench`, a novel benchmark for evaluating flowchart attribution, encompassing diverse styles, domains, and question types while incorporating high-quality attribution annotations. We introduce `FlowPathAgent`, a neurosymbolic agent that performs fine-grained attribution through graph-based reasoning. It first segments the flowchart, then converts it into a structured symbolic graph, and then employs an agentic approach to dynamically interact with the graph, to generate attribution paths. Experimental results show significant performance improvements, with `FlowPathAgent` outperforming existing methods by 6% to 65% on `FlowExplainBench`.

1 Introduction

Flowcharts are a fundamental tool for representing structured decision-making processes. Used across domains such as software engineering, business process modeling, and instructional design, flowcharts provide a visual roadmap of logical operations, guiding both human users and automated systems (Charntaweekhun and Wangsiripitak, 2006; Perols and Perols, 2024; Zimmermann et al., 2024; Ensmenger, 2016). Their structured yet visual nature makes them an effective medium for conveying procedural logic. However, interpreting flowcharts accurately is challenging due to their nonlinear structures (branching and loop-based control flow), where meaning emerges from the interplay between textual content, visual arrangement, and logical dependencies. Ambiguities in flowchart interpretation arise from diverse notational conventions, implicit relationships, and missing or inferred steps, making precise attribution of information sources difficult (Eppler et al., 2008).

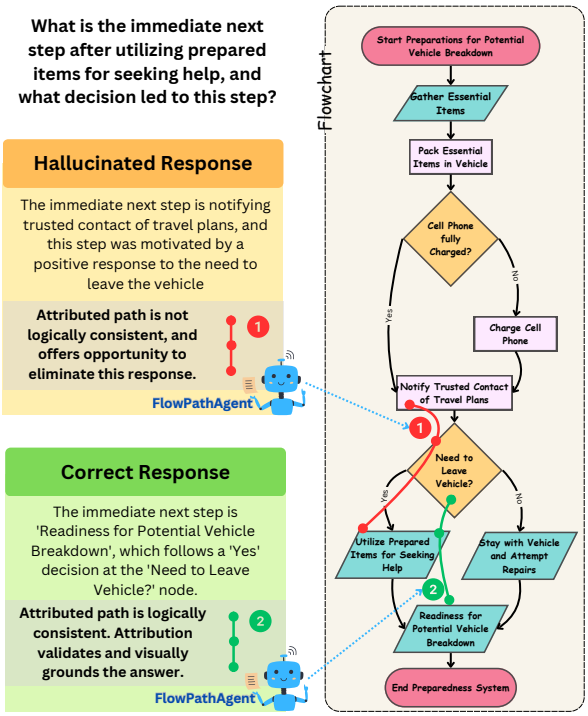


Figure 1: Attribution (represented by •–•–•) with `FlowPathAgent` ensures logical consistency in flowchart-based reasoning. `FlowPathAgent` uses a neurosymbolic approach to generate attribution paths (❶ & ❷) in the flowchart. This enhances interpretability and reliability in flowchart driven automated decision-making.

Recent advancements in Vision Language Mod-

els (VLMs) have enabled substantial progress in flowchart processing (Singh et al., 2024). These models leverage both textual and visual information, allowing them to extract structural relationships, recognize decision nodes, and generate answers based on flowchart content. However, despite their capabilities, VLMs struggle with hallucination: the tendency to generate information that is not grounded in the input (Huang et al., 2024; Guan et al., 2024). In the context of flowcharts, hallucination can manifest as misidentifying decision nodes, producing incorrect logical pathways, or fabricating connections that do not exist in the original structure. This issue severely impacts the reliability of automated flowchart reasoning, particularly in high-stakes applications such as software verification and process automation.

Although VLMs have made significant progress in understanding flowcharts, prior work has mainly concentrated on flowchart parsing (Arbaz et al., 2024), conversion (Shukla et al., 2023; Liu et al., 2022), and question-answering (Singh et al., 2024; Tannert et al., 2023), while overlooking the critical aspect of fine-grained attribution. While existing attribution methods (Huo et al., 2023; Chen et al., 2023) focus on textual grounding, attributing responses to visual-textual elements like flowcharts presents unique challenges. It involves not just text recognition, but also interpreting the interconnected decision nodes, hierarchical structures, and conditional pathways that define flowchart semantics. Attribution serves as a crucial mechanism for mitigating hallucination by explicitly tracing the paths in the flowchart that ground a particular response, enabling rigorous evaluation of the model's fidelity to the flowchart's logic, as illustrated in Fig 1. Such fine-grained attribution is fundamental for ensuring reliability, particularly when these systems are deployed in domains where verifiable decision-making is crucial.

Main Results. We introduce Flowchart Attribution as a post-hoc task aimed at identifying the minimal path within a flowchart that grounds the model's response. The minimal path refers to the shortest, most relevant sequence of nodes and edges that directly support the model's reasoning, encompassing all the key decision points and actions involved in the prediction. To facilitate the evaluation of this task, we propose `FlowExplainBench`, a novel benchmark that features a diverse set of flowcharts with varying styles, domains, question types, and high quality annotations.

We introduce `FlowPathAgent`, a neurosymbolic agent specifically designed to perform fine-grained flowchart attribution. Instead of relying solely on text-based or vision-based cues, `FlowPathAgent` integrates symbolic reasoning by using an agentic interface to interact with the flowchart as a graph object. `FlowPathAgent` begins with segmenting flowcharts into distinct components, followed by constructing symbolically operable flowchart representations. These graph-based representations have direct correspondence to visual regions of the flowchart, enabling the model to interoperate between the visual and symbolic representations. We leverage graph tools to extract and manipulate these representations, allowing for identification of relevant nodes and edges. Our methodology facilitates precise attribution of the model's reasoning steps to specific decision points within the flowchart, providing accurate and interpretable explanations of the model's output. Experimental results demonstrate that `FlowPathAgent` significantly outperforms existing baselines (Lai et al., 2024; Peng et al., 2023; Yuan et al., 2025), on fine-grained Flowchart attribution with improvements ranging from 6% to 65% on `FlowExplainBench`. Our **main contributions**[1] are:

- We introduce **Flowchart Attribution** as a post-hoc task, where the goal is to identify the minimal path within a flowchart that grounds the model's decision-making process.
- We present a novel evaluation benchmark, **`FlowExplainBench`**, consisting of 1k+ high quality attribution annotated flowcharts question-answer pairs. `FlowExplainBench` has diverse styles, domains, and questions.
- We introduce **`FlowPathAgent`**, a neurosymbolic agent capable of performing fine-grained attribution for flowcharts. `FlowPathAgent` uses a VLM based agentic approach to perform graph-based reasoning and symbolic manipulation to accurately trace the decision process within flowcharts.

2 Related Work

2.1 Flowchart Understanding

Research in flowchart understanding has evolved from basic image processing to complex reasoning

[1] Code and data will be released on acceptance.

tasks. Early work by (Ito, 1982) established methods for converting flowchart images to FORTRAN code. Modern deep learning approaches, such as FR-DETR (Sun et al., 2022a), have significantly improved symbol and edge detection through end-to-end architectures that combine CNN backbones with multi-scale transformers.

The emergence of large language models has led to benchmarks like FlowchartQA (Tannert et al., 2023) and FlowVQA (Singh et al., 2024), which assess geometric understanding, spatial reasoning, and logical progression capabilities of models on flowchart images, as a Question-Answer task. In parallel, code generation from flowcharts has developed as a distinct research direction (Liu et al., 2022; Shukla et al., 2023). Recent work like (Ye et al., 2024) has begun exploring alternatives to end-to-end VLMs; (Ye et al., 2024) introduced intermediate textual representations between visual processing and reasoning steps for Flowchart QA.

2.2 Attribution in LLMs

The rise of Large Language Models (LLMs) has brought unprecedented capabilities in generative tasks, yet challenges with factual accuracy persist (Zhang et al., 2023). While various solutions have emerged, including citation-aware training (Gao et al., 2023) and tool augmentation (Ye et al., 2023), ensuring reliable attributions remains crucial.

Three primary attribution strategies have emerged in literature. (1) Direct model-driven attribution generates answers and attributions simultaneously (Peskoff and Stewart, 2023; Sun et al., 2022b). (2) Post-retrieval answering retrieves information before answering (Ye et al., 2023; Li et al., 2023b; Huo et al., 2023; Chen et al., 2023). (3) Post-hoc attribution generates answers first and then searches for supporting references (Li et al., 2023a). Our work falls in the scope of Post-hoc attribution, as it serves as a modular approach integrable with existing system, without accessing the response generation mechanism.

Recent work has expanded attribution capabilities to handle diverse data formats. MATSA (Mathur et al., 2024) demonstrates fine-grained attribution for tabular data through a multi-agent approach that provides cell-level evidence for generated claims. Similarly, VISA (Ma et al., 2024) advances visual attribution by leveraging vision-language models to highlight specific regions in document screenshots that support generated responses, offering a novel solution for verifiable generation with precise source attribution.

3 Post-hoc Flowchart Attribution

We formalize fine-grained post-hoc Flowchart Attribution as follows: Given a dataset $\mathcal{D}$ consisting of a set of flowchart images $\mathcal{F}$, each flowchart image $c_i \in \mathcal{F}, c_i = \mathcal{I}^{w \times h \times 3}$ corresponds to a logical graph representation $G_i = (V_i, E_i)$, where V_i represents the set of nodes and E_i represents the edges between them. Each node corresponds to a logical operation or directive statement, and the edges represent the flow between these operations. Additionally, the input includes a flowchart-referring statement s_i, which is a natural language description of a process or action to be grounded in the flowchart image.

The underlying goal is to find a path in the image that grounds the statement s_i. This path may be disjoint, but it should correspond to a set of regions in the flowchart image. The regions are the physical abstraction that corresponds to the logical nodes in the graph. Formally, the task can be represented as a mapping function:

$$F : (c_i, s_i) \mapsto \mathcal{R}_{s_i},$$

where F maps the flowchart image c_i and the statement s_i to a set of regions $\mathcal{R}_{s_i}$ in the image. $\mathcal{R}_{s_i} = \{r_{i1}, r_{i2}, \ldots, r_{in}\}$ represents the sequence of regions in the image that correspond to a path of logical nodes, and the edges included between consecutive nodes $v_{i1}, v_{i2}, \ldots, v_{in}$ in the graph G_i, grounding the statement s_i. The path may be disjoint, but it should satisfy the following criteria:

1. Optimality: The path should be the shortest sequence of regions that ground the statement s.

2. Contextual Alignment: The path should correspond to the relevant actions and decisions described in s, matching the flow of the process.

3. Exclusivity: No additional regions outside of the minimal path are necessary to fully explain the statement s.

4 FlowExplainBench

To enable systematic evaluation of flowchart attribution, we introduce FlowExplainBench, a comprehensive benchmark designed with four key criteria: diverse visual styles, varied question types, multiple flowchart domains, and faithful ground-truth attributions. Each entry in the dataset consists of the following components: the flowchart image c, a statement s (which, in this

	Code	Wiki	Instruct	Overall
# of Flowcharts	189	470	294	953
# of Questions	246	610	382	1238
Fact Retrieval	88	163	102	353
Applied Scenario	69	128	90	287
Flow Referential	43	128	87	258
Topological	46	191	103	340
Avg # of Nodes	11.85	24.49	21.59	21.08
Max # of Nodes	29	43	44	44
Avg Attributed Path Length	2.59	3.21	2.88	2.99
Max Attributed Path Length	15	35	21	35
Avg Words (Question)	26.99	26.12	26.56	26.43
Avg Words (Answer)	8.62	8.74	9.50	8.95

Table 1: Detailed overview of FlowExplainBench, showing the distribution and characteristics of its constituent splits.

context, is a Question-Answer pair), a set of attributed logical nodes $v_1, v_2, \ldots, v_n$, and their corresponding visual regions $\mathcal{R}_s = \{r_1, r_2, \ldots, r_n\}$. These visual regions represent the physical abstractions of the logical nodes, which are mapped from the flowchart image c as discussed in section 3. Table 1 summarizes the constitution of FlowExplainBench.

4.1 Data Sources

FlowExplainBench is constructed using the test split of the FlowVQA dataset (Singh et al., 2024). This dataset comprises high-quality flowchart images sourced from diverse domains, including the FloCo dataset, which emphasizes code-related flowcharts (Shukla et al., 2023), as well as widely recognized DIY platforms such as Wikihow and Instructables. These sources contribute to three distinct data splits: *Code*, *Wiki*, and *Instruct*. For each flowchart, corresponding Mermaid code and metadata (e.g., original code and process summaries) are included. The dataset contains four question types: Fact retrieval, Applied Scenario, Flow Referential, and Topological.

4.2 Visual Diversity

While FlowVQA contains flowcharts of high quality, the visual diversity is limited since all flowcharts are designed using the default Mermaid style template. To ensure that FlowExplainBench represents a broader spectrum of flowchart styles encountered in real-world applications, we introduce four distinct style types for flowchart generation: **1. Single Color:** Flowcharts that use a single color for all nodes throughout the chart for simplicity and visual cohesion. **2. Multi Color:** Flowcharts that utilize multiple colors, sampled from a palette to represent different nodes. **3. Default Mermaid:** The standard Mermaid styling. **4. Black and White:** Flowcharts designed using only black and white elements. For the **Single Color** style, we incorporate 40 unique colors, while for the **Multi Color** style, we use 35 curated color palettes each containing 4 to 5 colors.

To apply style options to the source Mermaid code, we first generate SVG flowcharts from the Mermaid code. Subsequently, we inject templated CSS into the SVGs to implement the desired styles. Finally, the SVGs are converted into PNG images, and the regions of interest (i.e., the flowchart nodes) are defined using the positional and shape information derived from the SVG metadata.

4.3 Attribution Annotation

The attribution annotation process follows a two-step pipeline:

Step 1: Automatic Labeling. We begin by using GPT-4 to perform the initial attribution process. Given the Mermaid code and the corresponding QA pair, GPT-4 is prompted to generate initial attributions of the flowchart's nodes to the question-answer pair. By analyzing the nature of different question types, we generalize the attribution patterns and provide GPT-4 with few-shot examples to guide its understanding.

Step 2: Human Verification. Two human evaluators are involved in further attribution, where they interact with an attribution platform that allows them to select nodes from the flowchart to be attributed. The inter-annotator agreement is measured using Cohen's Kappa (κ), which shows a high level of agreement both between the two annotators (κ = 0.89) and between the annotators and the initial GPT-4-generated labels (κ = 0.72, 0.80). Further details on human annotation are discussed in the Appendix.

4.4 Data Filtering

To ensure high-quality and diverse samples, a multi-step filtering process was applied. First, two generic questions, "How many nodes exist in the flowchart?" and "How many edges exist in the flowchart?" were excluded, as they required trivially attributing the entire graph rather than reasoning over its fine-grained individual components. This excluded 1792 samples, which were not included in the annotation exercise described above.

Next, after annotation, for each flowchart image, the question with the highest agreement among annotators was selected, prioritizing cases where

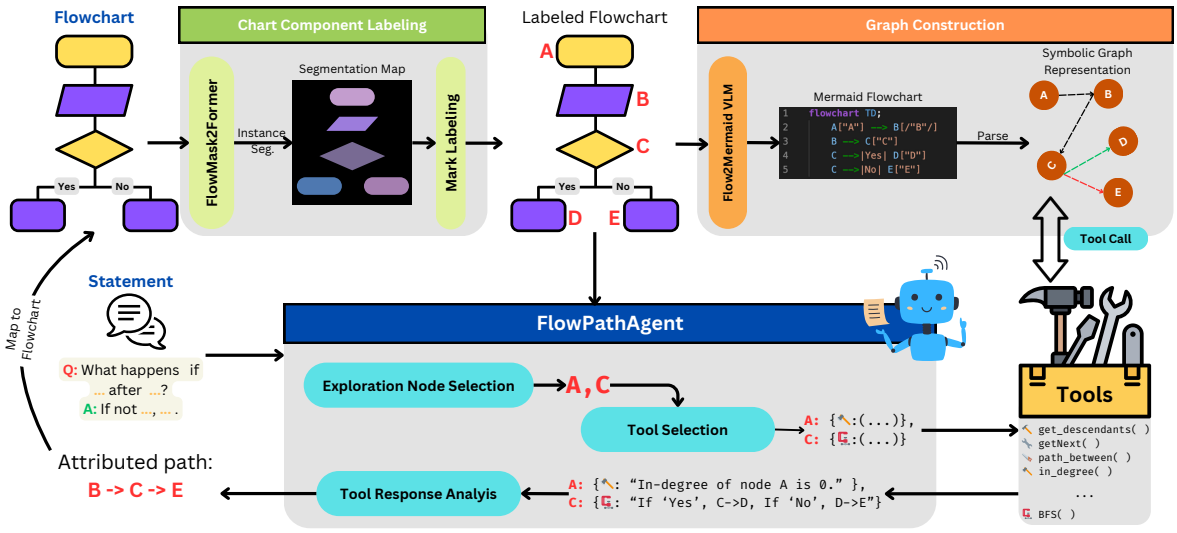


Figure 2: Overview of **FlowPathAgent**. FlowPathAgent processes a flowchart image through segmentation-based component labeling, constructs a symbolic graph representation using Mermaid, and employs a neurosymbolic agent, that treats the flowchart as a symbolic graph to attribute nodes based on an input statement. The agent interacts with predefined tools to analyze and traverse the flowchart structure, producing attributions as interpretable mappings of relevant nodes back onto the original flowchart.

both human annotators concurred. This yielded an initial set of 953 samples. To achieve balance across three domains and four question types, additional high-agreement samples were selected from underrepresented categories, resulting in a final benchmark of 1,238 samples.

5 FlowPathAgent

FlowPathAgent (Fig 2), is a neurosymbolic agent designed for structured reasoning over flowcharts for fine-grained flowchart attribution. The approach consists of three key stages: **Chart Component Labeling**, which segments and labels chart components; **Graph Construction**, which constructs a symbolic graph from the labeled flowchart; and **Neurosymbolic Agent-based Analysis**, which uses graph-based tools to interact with the symbolic flowchart to generate attributed paths. Each stage plays a critical role in bridging the gap between visual representations and symbolic reasoning over structured workflows.

5.1 Chart Component Labeling

The primary objective of this stage is to identify and label individual flowchart components, ensuring an explicit correspondence between visual elements and the symbolic representations generated in subsequent steps.

FlowMask2Former. To achieve flowchart component recognition, we construct a synthetic dataset using the training split of FlowVQA (Singh et al., 2024), incorporating style diversification techniques similar to those described in Section 4.2, but with different color schemes. Further the node content is replaced with randomized text. These augmentations improve domain generalization and ensure robust performance across diverse flowchart styles. We fine-tune Mask2Former (Cheng et al., 2022) on this dataset for instance segmentation, specifically targeting node recognition. The fine-tuned model, FlowMask2Transformer, generates segmentation maps, from which individual nodes are sequentially labeled using alphabetical identifiers, rendered in red text on the flowchart image, to serve as visual anchors for graph construction, reasoning, and node referencing.

5.2 Graph Construction

Flowcharts inherently encode structured logical processes, making graph-based representations ideal for symbolic reasoning. By converting flowcharts into symbolic graph structures, we facilitate efficient graph-based operations such as traversal, topological analysis, and conditional evaluation.

Flow2Mermaid VLM. To convert the labeled flowchart to a symbolic graph representation, we first convert the visual flowchart to a Mermaid code, and then parse the Mermaid code to generate the symbolic graph. For the Flowchart to

Mermaid transformation, we fine-tune Qwen2-VL(7B) (Wang et al., 2024) using supervised fine-tuning (SFT) on a style-diversified projection of the FlowVQA (Singh et al., 2024) training set, with marked alphabetic node labels sourced from SVG metadata. Flow2Mermaid VLM is trained to generate Mermaid flowchart code directly from flowchart images, using the alphabetical node labels as anchors to maintain consistency between visual and symbolic representations. We perform fine-tuning to improve the ability to generate accurate and semantically robust Mermaid syntax, minimize structural inconsistencies that could affect graph analyses, and adapt to the varied aspect ratios and visual styles found in flowcharts.

The generated Mermaid representation is then parsed into a symbolic graph, tailored to capture the specific properties of flowcharts, including boolean conditional edges and node-level statement mappings. Additionally, we define a comprehensive suite of tools to operate on this symbolic graph, enabling structured function calls for reasoning over the flowchart's logical structure. The list of tools and their API is described in the Appendix.

5.3 Neurosymbolic Agent

`FlowPathAgent` employs a neurosymbolic reasoning approach to attribute relevant nodes based on an input statement. In this context, neurosymbolic reasoning combines neural models i.e. VLMs to plan, reason and attribute in a discrete token space, based on observations made via tool use with a symbolic graph, representing a flowchart. The agent operates on a sequence of interdependent steps:

1. Node Selection: During the initial planning stage, our agent identifies nodes to be explored by referencing their corresponding labels in the flowchart image. Additionally, it clarifies the expectation and underlying rationale for each node selection. This is the only step where the labeled flowchart image is passed to the underlying VLM.

2. Tool Selection: Our agent employs reasoning-based prompting to determine the necessary symbolic tools and their respective functional parameters for the selected nodes.

3. Tool Execution: The selected tools are executed on the symbolic graph representation to extract relevant insights.

Note: Multiple sequential cycles of Tool Selection and Tool Execution may occur, with each cycle selecting and executing a single tool at a time.

4. Tool Response Analysis: The agent interprets observations from tool-use, in relation to the given statement, generating a path of nodes in the flowchart that attribute the statement.

5. Mapping to Original Flowchart: Finally, the attributed path's node labels are mapped back onto the flowchart image using the segmentation regions obtained during the labeling stage.

6 Experimental Set-up

6.1 Baselines

Zero-shot GPT-4o Bounding Box We use GPT-4o (OpenAI, 2024) to predict normalized bounding box coordinates for chart components based on text and the visual chart, following established methods for zero-shot localization (Yang et al., 2023b).

Kosmos-2:(Peng et al., 2023) is a multimodal large language model that combines text-to-visual grounding, supporting tasks like referring expression interpretation and bounding box generation by linking objects in images with text.

LISA: (Li et al., 2023b) is a model for generating segmentation masks from textual queries, extending VLM capabilities to segmentation tasks, and excels in zero-shot performance with minimal fine-tuning on task-specific data.

SA2VA(Yuan et al., 2025) is a unified model for dense grounded understanding of both images and videos, combining SAM-2 for segmentation and LLaVA for vision-language tasks, enabling robust performance in referring segmentation.

GPT4o + FlowMask2Former SoM Prompting, as an ablation study, we incorporate this baseline where GPT-4o utilizes the segmented flowchart generated by FlowMask2Former, and applies Set-of-Marks (SoM) (Yang et al., 2023a) prompting to guide the model's predictions.

6.2 Evaluation

To map the segmented regions to the ground truth nodes, we apply an Intersection over Union (IoU) threshold of 0.7 to ensure high fidelity between ground-truth and predicted nodes. The ground-truth node with the maximum overlap is selected as the reference for the segmented node. This process is crucial for fine-grained attribution, where accurate identification of individual flowchart components is required. For each baseline, we collect the nodes identified by the model and treat the set of ordered nodes as the attributed path. We then compute the micro-averaged Precision, Recall, and F1 scores to assess the model's performance.

Baseline	Overall			FEBench-Code			FEBench-Wiki			FEBench-Instruct		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Kosmos-2 (Peng et al., 2023)	37.14	1.76	3.36	41.41	6.45	11.16	20.69	0.31	0.60	38.30	1.64	3.14
LISA (Lai et al., 2024)	18.01	14.34	15.97	35.36	19.18	24.87	14.09	11.74	12.81	18.45	16.18	17.24
SA2VA (Yuan et al., 2025)	66.36	9.88	17.20	79.35	19.34	31.10	58.47	7.40	13.14	65.99	8.82	15.56
GPT4o Bounding Box	58.82	1.90	3.68	80.00	1.89	3.69	53.19	1.29	2.51	57.89	3.00	5.70
GPT4o SoM	74.10	67.69	70.75	67.32	70.28	68.77	74.55	65.03	69.47	77.84	70.91	74.22
`FlowPathAgent`	77.19	77.21	77.20	74.18	80.62	77.27	76.29	74.21	75.23	80.28	80.19	80.23

Table 2: Performance comparison of `FlowPathAgent` with baselines on `FlowExplainBench`. Best and second-best results have been highlighted.

6.3 Implementation Details

The `facebook/mask2former-swin-tiny-coco-instance` model is fine-tuned for 20 epochs for FlowMask2Former, employing a learning rate of 1×10^{-5} with a cosine annealing scheduler. A batch size of 4 is used, and gradient accumulation occurs over 4 steps to address memory constraints. Training is conducted using 16-bit precision to improve computational efficiency. In the case of the Mermaid2Graph Vision-Language Model (VLM), fine-tuning is performed on the `unsloth/Qwen2-VL-7B-Instruct` checkpoint, focusing on vision, language, attention, and MLP layers. This model is trained for 3 epochs with a batch size of 1 and gradient accumulation over 5 steps. A learning rate of 2×10^{-4} is applied with a linear scheduler. To optimize memory usage, the model is loaded in 4-bit precision, and AdamW is used as the optimizer with a weight decay of 0.01. The baseline models are initialized as follows: LISA from `xinlai/LISA-13B-llama2-v1`, Kosmos-2 from `microsoft/kosmos-2-patch14-22`, and Sa2VA from `ByteDance/Sa2VA-8B`. Default settings and parameters are used for all baselines.

7 Results and Discussion

Baseline Comparison. `FlowPathAgent` demonstrates a significant improvement over all baseline models when evaluated on the `FlowExplainBench`, outperforming them by a margin of 6-65 percentage points, as shown in Table 2. Visual grounding models, including Kosmos-2, LISA, and SA2VA, exhibit suboptimal performance. This is primarily due to their limited ability to process visual logic, which is crucial for fine-grained flowchart attribution. These models struggle to correctly map logical relationships between elements in the flowchart, resulting in less accurate attributions.

Among the baselines, GPT4o Zero Shot Bounding Box shows the poorest performance. This model lacks inherent capabilities for mask generation, and instead generates bounding boxes in the textual token space, which is not well-suited for the task of flowchart attribution.

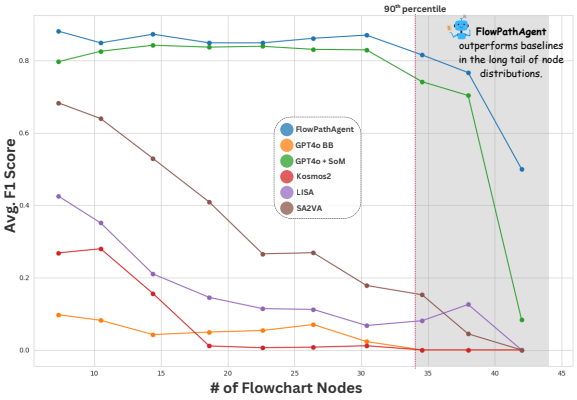


Figure 3: Performance comparison of different baselines against node count in flowcharts. `FlowPathAgent` demonstrates superior effectiveness, particularly in the long-tail of the distribution

In contrast, GPT4o SoM achieves a comparatively stronger performance. This can be attributed to the effective segmentation abilities of FlowMask2Former, which ensures that the elements to be attributed are accurately captured in the candidate set available to the model. Additionally, the reasoning capabilities of GPT4o contribute to improved performance by leveraging these segmented components in a logical manner.

Further analysis of performance trends reveals an interesting behavior when we examine the performance of different models against the number of nodes in the flowchart, as illustrated in Fig 3. As the complexity of the flowchart increases (i.e., as the number of nodes decreases), a performance dip is observed across all methods. This is likely due to the increasing difficulty in processing larger, more complex flowcharts, especially for models relying heavily on the visual presentation of the flowchart. In contrast, `FlowPathAgent` maintains a more consistent performance, with a relatively smaller dip in performance despite the increased

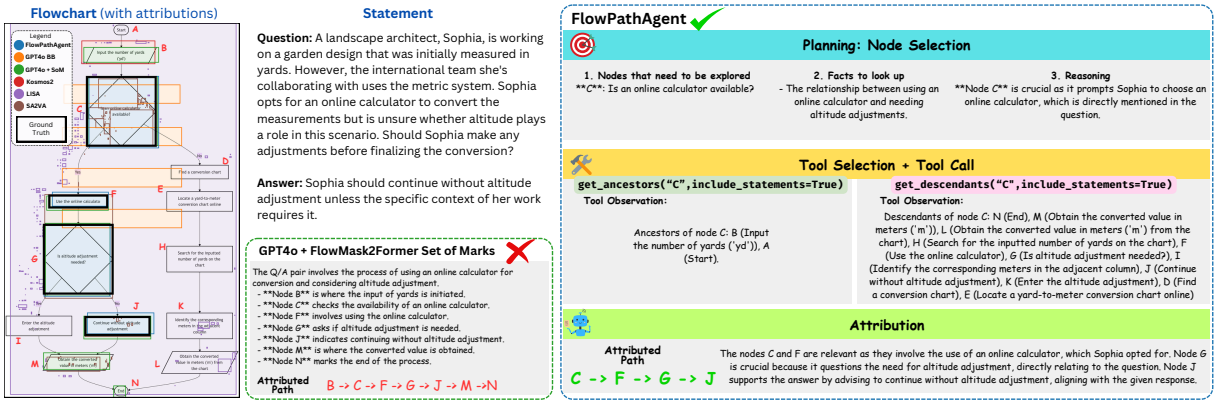


Figure 4: Qualitative comparison of FlowPathAgent with baseline methods. The flowchart illustrates attributions generated by various baselines, highlighting the agentic trace of FlowPathAgent. We contrast its output with the next strongest baseline, GPT-4o+SoM, to showcase differences in attribution quality and interpretability.

.

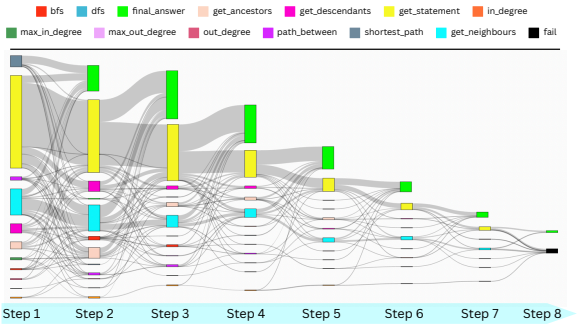


Figure 5: Flow diagram of the sequence of tools used by FlowPathAgent on FlowExplainBench. Each *Step* refers to a cycle of Tool Selection + Call.

complexity. This can be attributed to the model's ability to treat flowchart elements as logical entities, rather than solely relying on their visual representation. By leveraging its neurosymbolic approach, FlowPathAgent is able to more effectively process and attribute complex flowchart structures, providing robust and reliable attributions even in the long tail of node distributions.

Qualitative Analysis. Fig 4 presents a qualitative comparison between FlowPathAgent and various baseline models. The GPT4o Zero Shot Bounding Box baseline fails to generate bounding boxes that overlap with or match the shape and dimensions of any flowchart nodes. On the other hand, LISA tends to overgeneralize by attributing the entire flowchart image, producing small, noisy masks that cover irrelevant areas, which reduces the clarity and precision of its attributions. Kosmos-2 also struggles with segmenting the nodes associated with the statement; it segments a single irrelevant node. SA2VA, while performing better than the other visual grounding models, still exhibits limitations. It generates low IoU masks around some correct nodes. Additionally, it sometimes produces extraneous masks that are not relevant to the flowchart's logical structure. GPT4o with SoM shows some improvement, but tends to over-attribute by including steps that are further ahead in the flowchart than necessary. In contrast, FlowPathAgent excels by accurately detecting and attributing the entire flowchart path, identifying all the relevant nodes with high precision.

Figure 5 displays the sequence of tools the agent employs across tool selection and execution steps, capped at 8 steps. The frequent use of the get_statement tool highlights its vital role in verifying fact retrieval and scenario-based QA pairs without relying on visual input. Notably, Step 3 emerges as the most common final stage (evident by final_answer), with nearly half of the penultimate tool calls dedicated to analyzing the graph structure. Additional agent behavior analysis and qualitative examples are provided in the appendix.

8 Conclusion

We introduced the task of Flowchart Attribution and proposed FlowExplainBench, a benchmark tailored for evaluating fine-grained attribution in flowcharts. We presented FlowPathAgent, a neurosymbolic agent that leverages graph-based reasoning to accurately identify the minimal path underpinning model responses. Experimental results demonstrate significant improvements over existing baselines, highlighting the effectiveness of our approach.

9 Limitations

While our approach demonstrates strong performance, there are areas for further improvement. First, although `FlowPathAgent` effectively integrates symbolic reasoning, it builds on FlowMask2Former for segmentation and Flow2Mermaid VLM for converting visual flowcharts to mermaid code. As with any modular system, potential errors in these components may influence overall performance. However, our framework remains flexible, allowing for seamless integration of alternative models better suited to specific scenarios.

Second, our benchmark, `FlowExplainBench`, captures a diverse range of flowchart structures but does not yet encompass all real-world variations, such as hand-drawn diagrams. The primary challenge lies in the availability of high-quality datasets with comprehensive annotations. While existing methods address hand-drawn flowchart segmentation, scaling them for attribution remains an open area of research. Future work could explore semi-supervised or automated annotation strategies to enhance coverage.

Lastly, our approach is designed for static flowcharts, and extending it to dynamic or interactive systems presents an opportunity for further research. Many real-world applications involve evolving decision-making processes, which could benefit from models that handle sequential updates and conditional dependencies.

Future work could address these limitations by improving segmentation robustness, expanding the benchmark to include more diverse flowchart types, and developing models capable of handling dynamic and interactive flowcharts. Additionally, integrating reinforcement learning or self-supervised learning techniques could enhance model adaptability and generalization across various flowchart formats.

10 Ethics Statement

In this study, we utilize the publicly accessible FlowVQA dataset, which is distributed under the MIT License[2]. We ensure that the identities of human evaluators remain confidential, and no personally identifiable information (PII) is used at any stage of our research. This work is focused exclusively on applications for fine-grained visual flowchart attribution and is not intended for other use cases. We also recognize the broader challenges associated with large language models (LLMs), including potential risks related to misuse and safety, and we encourage readers to consult the relevant literature for a more detailed discussion of these issues (Kumar et al., 2024; Cui et al., 2024; Luu et al., 2024).

[2] https://github.com/flowvqa/flowvqa?tab=MIT-1-ov-file

References

Abdul Arbaz, Heng Fan, Junhua Ding, Meikang Qiu, and Yunhe Feng. 2024. Genflowchart: parsing and understanding flowchart using generative ai. In *International Conference on Knowledge Science, Engineering and Management*, pages 99–111. Springer.

Kanis Charntaweekhun and Somkiat Wangsiripitak. 2006. Visual programming using flowchart. In *2006 International Symposium on Communications and Information Technologies*, pages 1062–1065. IEEE.

Jifan Chen, Grace Kim, Aniruddh Sriram, Greg Durrett, and Eunsol Choi. 2023. Complex claim verification with evidence retrieved in the wild. *arXiv preprint arXiv:2305.11859*.

Bowen Cheng, Ishan Misra, Alexander G. Schwing, Alexander Kirillov, and Rohit Girdhar. 2022. Masked-attention mask transformer for universal image segmentation.

Tianyu Cui, Yanling Wang, Chuanpu Fu, Yong Xiao, Sijia Li, Xinhao Deng, Yunpeng Liu, Qinglin Zhang, Ziyi Qiu, Peiyang Li, Zhixing Tan, Junwu Xiong, Xinyu Kong, Zujie Wen, Ke Xu, and Qi Li. 2024. Risk taxonomy, mitigation, and assessment benchmarks of large language model systems. *Preprint*, arXiv:2401.05778.

Nathan Ensmenger. 2016. The multiple meanings of a flowchart. *Information & Culture*, 51(3):321–351.

Martin J Eppler, Jeanne Mengis, and Sabrina Bresciani. 2008. Seven types of visual ambiguity: On the merits and risks of multiple interpretations of collaborative visualizations. In *2008 12th International Conference Information Visualisation*, pages 391–396. IEEE.

Tianyu Gao, Howard Yen, Jiatong Yu, and Danqi Chen. 2023. Enabling large language models to generate text with citations. *arXiv preprint arXiv:2305.14627*.

Tianrui Guan, Fuxiao Liu, Xiyang Wu, Ruiqi Xian, Zongxia Li, Xiaoyu Liu, Xijun Wang, Lichang Chen, Furong Huang, Yaser Yacoob, et al. 2024. Hallusionbench: an advanced diagnostic suite for entangled language hallucination and visual illusion in large vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14375–14385.

Wen Huang, Hongbin Liu, Minxin Guo, and Neil Zhenqiang Gong. 2024. Visual hallucinations of multimodal large language models. *arXiv preprint arXiv:2402.14683*.

Siqing Huo, Negar Arabzadeh, and Charles Clarke. 2023. Retrieving supporting evidence for generative question answering. In *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region*, pages 11–20.

S. Ito. 1982. Automatic input of flow chart in document image. In *Proceedings of the 6th International Conference on Software Engineering*, ICSE '82, page 319–328, Washington, DC, USA. IEEE Computer Society Press.

Ashutosh Kumar, Sagarika Singh, Shiv Vignesh Murty, and Swathy Ragupathy. 2024. The ethics of interaction: Mitigating security threats in llms. *Preprint*, arXiv:2401.12273.

Xin Lai, Zhuotao Tian, Yukang Chen, Yanwei Li, Yuhui Yuan, Shu Liu, and Jiaya Jia. 2024. Lisa: Reasoning segmentation via large language model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9579–9589.

Dongfang Li, Zetian Sun, Xinshuo Hu, Zhenyu Liu, Ziyang Chen, Baotian Hu, Aiguo Wu, and Min Zhang. 2023a. A survey of large language models attribution. *arXiv preprint arXiv:2311.03731*.

Xiaonan Li, Changtai Zhu, Linyang Li, Zhangyue Yin, Tianxiang Sun, and Xipeng Qiu. 2023b. Llatrieval: Llm-verified retrieval for verifiable generation. *arXiv preprint arXiv:2311.07838*.

Zejie Liu, Xiaoyu Hu, Deyu Zhou, Lin Li, Xu Zhang, and Yanzheng Xiang. 2022. Code generation from flowcharts with texts: A benchmark dataset and an approach. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 6069–6077, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Quan Khanh Luu, Xiyu Deng, Anh Van Ho, and Yorie Nakahira. 2024. Context-aware llm-based safe control against latent risks. *Preprint*, arXiv:2403.11863.

Xueguang Ma, Shengyao Zhuang, Bevan Koopman, Guido Zuccon, Wenhu Chen, and Jimmy Lin. 2024. Visa: Retrieval augmented generation with visual source attribution. *Preprint*, arXiv:2412.14457.

Puneet Mathur, Alexa Siu, Nedim Lipka, and Tong Sun. 2024. Matsa: Multi-agent table structure attribution. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 250–258.

OpenAI. 2024. Hello, gpt-4o! `https://openai.com/index/hello-gpt-4o/`.

Zhiliang Peng, Wenhui Wang, Li Dong, Yaru Hao, Shaohan Huang, Shuming Ma, and Furu Wei. 2023. Kosmos-2: Grounding multimodal large language models to the world. *arXiv preprint arXiv:2306.14824*.

Rebecca R Perols and Johan L Perols. 2024. The impact of auditors creating flowcharts on auditors' understanding of the flow of transactions and internal control evaluation. *Managerial Auditing Journal*, 39(7):779–798.

Denis Peskoff and Brandon M Stewart. 2023. Credible without credit: Domain experts assess generative language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 427–438.

Shreya Shukla, Prajwal Gatti, Yogesh Kumar, Vikash Yadav, and Anand Mishra. 2023. Towards making flowchart images machine interpretable. In *International Conference on Document Analysis and Recognition*, pages 505–521. Springer.

Shubhankar Singh, Purvi Chaurasia, Yerram Varun, Pranshu Pandya, Vatsal Gupta, Vivek Gupta, and Dan Roth. 2024. FlowVQA: Mapping multimodal logic in visual question answering with flowcharts. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 1330–1350, Bangkok, Thailand and virtual meeting. Association for Computational Linguistics.

Lianshan Sun, Hanchao Du, and Tao Hou. 2022a. Fr-detr: End-to-end flowchart recognition with precision and robustness. *IEEE Access*, 10:64292–64301.

Zhiqing Sun, Xuezhi Wang, Yi Tay, Yiming Yang, and Denny Zhou. 2022b. Recitation-augmented language models. *arXiv preprint arXiv:2210.01296*.

Simon Tannert, Marcelo G. Feighelstein, Jasmina Bogojeska, Joseph Shtok, Assaf Arbelle, Peter W. J. Staar, Anika Schumann, Jonas Kuhn, and Leonid Karlinsky. 2023. FlowchartQA: The first large-scale benchmark for reasoning over flowcharts. In *Proceedings of the 1st Workshop on Linguistic Insights from and for Multimodal Language Processing*, pages 34–46, Ingolstadt, Germany. Association for Computational Lingustics.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. 2024. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*.

Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. 2023a. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v. *arXiv preprint arXiv:2310.11441*.

Zhengyuan Yang, Linjie Li, Kevin Lin, Jianfeng Wang, Chung-Ching Lin, Zicheng Liu, and Lijuan Wang. 2023b. The dawn of lmms: Preliminary explorations with gpt-4v (ision). *arXiv preprint arXiv:2309.17421*, 9(1):1.

Junyi Ye, Ankan Dash, Wenpeng Yin, and Guiling Wang. 2024. Beyond end-to-end vlms: Leveraging intermediate text representations for superior flowchart understanding. *Preprint*, arXiv:2412.16420.

Xi Ye, Ruoxi Sun, Sercan Ö Arik, and Tomas Pfister. 2023. Effective large language model adaptation for improved grounding. *arXiv preprint arXiv:2311.09533*.

Haobo Yuan, Xiangtai Li, Tao Zhang, Zilong Huang, Shilin Xu, Shunping Ji, Yunhai Tong, Lu Qi, Jiashi Feng, and Ming-Hsuan Yang. 2025. Sa2va: Marrying sam2 with llava for dense grounded understanding of images and videos. *Preprint*, arXiv:2501.04001.

Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. 2023. Siren's song in the ai ocean: a survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219*.

Anthony E Zimmermann, Ethan E King, and Diptiman D Bose. 2024. Effectiveness and utility of flowcharts on learning in a classroom setting: A mixed-methods study. *American Journal of Pharmaceutical Education*, 88(1):100591.

A Further Details

A.1 Computational Budget

Table 3 shows the computational budget for this paper, broken down by associated tasks.

Task	Time (hours)	# of GPUs	GPU Spec
FlowMask2Former Training	14	1	NVIDIA RTX A6000
Flow2Mermaid VLM Training	8	1	NVIDIA RTX A6000
Baseline, Trained Model Inference	3	1	NVIDIA RTX A6000

Table 3: Computational Budget for experiments in the paper

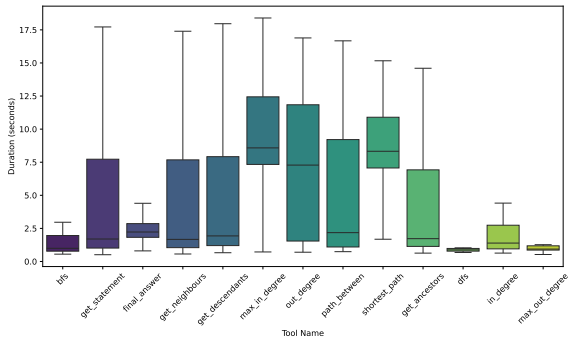


Figure 6: Box-plot distribution of time taken in each tool call, in seconds.

A.2 Dataset Examples

Fig 8 represents examples from the training set for FlowMask2Former and Flow2Mermaid VLM, displaying different style types.

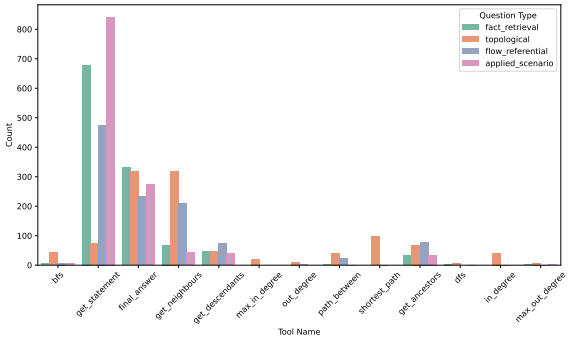


Figure 7: Distribution of count of tool calls, segregated by question type.

Figs 9-11 represent examples from `FlowExplainBench` from different domains, with different styles and question types.

A.3 Qualitative Examples

Fig. 13 and 13 present qualitative comparisons of the baseline methods. While these examples do not comprehensively represent the overall performance ranking, they have been deliberately chosen to highlight specific limitations and failure cases of each method. This selection aims to provide insights into the scenarios where certain approaches struggle, offering a clearer understanding of their weaknesses.

B Agent Analysis

B.1 API Description

Fig. 14 shows the class diagram of the data structure used to represent Nodes, Edges, and the Flowchart. The `FlowChart` class serves as the primary structure, managing a collection of `Node` objects, each identified by a unique ID and containing a statement. Nodes are interconnected through `Edge` objects, which define directed relationships with optional conditions (`Yes`, `No`, or unconditional).

Table 4 summarises the API for the tools provided to `FlowPathAgent`. Except for `final_answer` which returns the final answer and reasoning involved, all other tools operate on a global `FlowChart` object initiated from mermaid code generated by Flow2Mermaid VLM.

B.2 Tool-use Analysis

Fig 6 shows the distribution of run-time of tool cals, called from within the agentic framework. `max_in_degree` has the maximum median runtime, which can be explained by the fact that

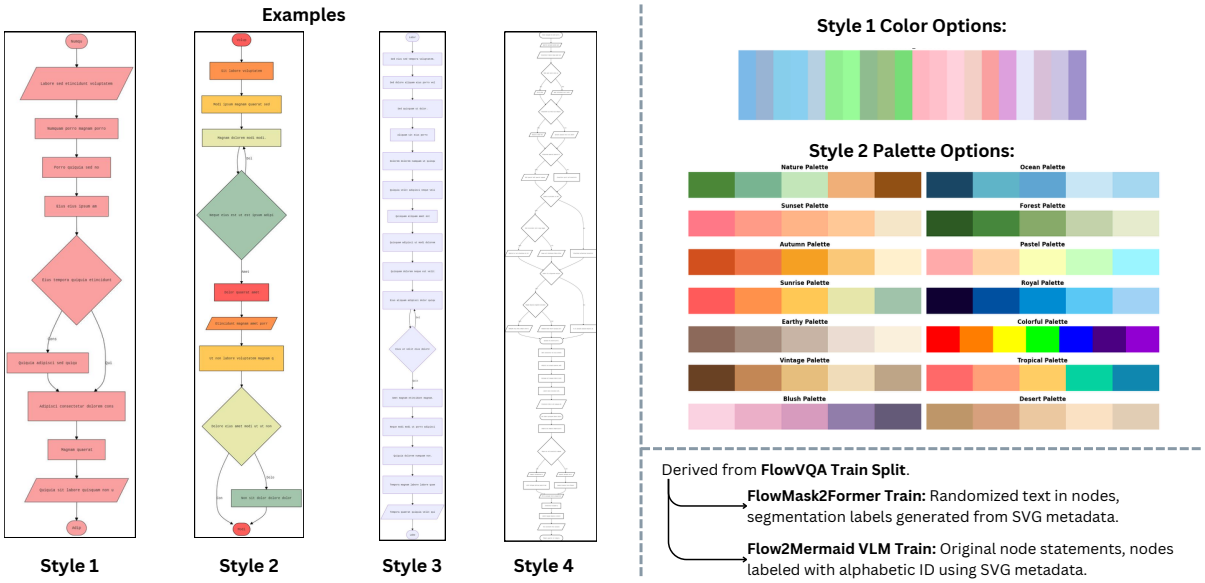


Figure 8: Overview of training split used for FlowMask2Former, and Flow2Mermaid VLM. The figure demonstrates the style options, color palettes used, and distinction between both training sets.

the `Node` data-structure employed by us only has outgoing edges, meaning all nodes have to be iterated to find the node with maximum in-degree. The second highest median run-time belongs to `shortest_path`, which is implemented as a O(V+E) breadth-first-search based algorithm.

We analyze the distribution of tool calls by question type in Fig 7. Topological questions show the most diversity in terms of tool calls, since they require interpreting structural aspects of the FlowChart. `get_statement` is the most common tool for the other question types. This is because all the other questions require content from inside the flowchart, and often involve multiple calls of `get_statement` in a single agentic run. For flow-referential questions, `get_neighbours` is a popular tool, since this tool allows downstream flow analysis from an anchor node.

B.3 Prompts and Implementation

Fig 15 and 16 represent the system prompt template, and planning prompt template used by `FlowPathAgent`. We implemented `FlowPathAgent` using HuggingFace's smolagents [3] library. We patched the library to ensure that visual tokens are only used in the planning step (node selection step), and removed from the conversation template thereafter.

[3] https://github.com/huggingface/smolagents

C Benchmark Construction

C.1 Automatic Labeling

Fig 17 represents the prompt template used to perform automatic annotations, in step 1 of our ground truth annotation process.

C.2 Style Diversity

Fig 18 represents the color schemes used to augment style diversity in `FlowExplainBench`. Styles for auxiliary datasets used in this paper are presented in Fig 8.

C.3 Human Annotation

We employed two graduate student annotators, aged 22-25. The annotators were proficient in English, and were exposed to flowchart QA samples from the training set before the annotation exercise, to make them comfortable with the flowcharts involved. The annotators were fairly compensated at the standard Graduate Assistant hourly rate, following their respective graduate school policies.

Fig 19 shows a summary of the annotator guidelines, and Fig 20 shows the annotation platform used.

Name	Description	Arguments
get_statement	Returns the statement associated with a node.	node_id (string): Identifier of the node.
get_ancestors	Identifies all nodes that have paths leading to the specified node.	node_id (string): Identifier of the target node. levels (integer, optional): Maximum levels to traverse. include_statements (boolean, optional): If True, includes statements. Defaults to False.
get_descendants	Identifies all nodes that can be reached from the specified node.	node_id (string): Identifier of the starting node. levels (integer, optional): Maximum levels to traverse. include_statements (boolean, optional): If True, includes statements. Defaults to False.
get_neighbours	Returns all nodes connected to the given node by outgoing edges.	node_id (string): Identifier of the node. include_statements (boolean, optional): If True, includes statements. Defaults to False.
in_degree	Returns the number of incoming edges to a node.	node_id (string): Identifier of the node.
out_degree	Returns the number of outgoing edges from a node.	node_id (string): Identifier of the node.
max_in_degree	Identifies nodes with the highest incoming edges.	None
max_out_degree	Identifies nodes with the highest outgoing edges.	None
bfs	Performs breadth-first search from a starting node.	start_id (string, optional): Identifier of the node. conditions (object, optional): Dictionary of edge conditions. include_statements (boolean, optional): If True, includes statements. Defaults to False.
dfs	Performs depth-first search from a starting node.	start_id (string, optional): Identifier of the node. conditions (object, optional): Dictionary of edge conditions. include_statements (boolean, optional): If True, includes statements. Defaults to False.
path_between	Finds a path between two nodes, considering edge conditions.	start_id (string): Start node. end_id (string): End node. conditions (object, optional): Edge conditions. include_statements (boolean, optional): If True, includes statements. Defaults to False.
shortest_path	Finds the shortest path between two nodes using BFS.	start_id (string): Start node. end_id (string): End node. conditions (object, optional): Edge conditions. include_statements (boolean, optional): If True, includes statements. Defaults to False.
final_answer	Provides a final answer to the given problem.	answer (any): The final answer.

Table 4: Tools provided to `FlowPathAgent`.

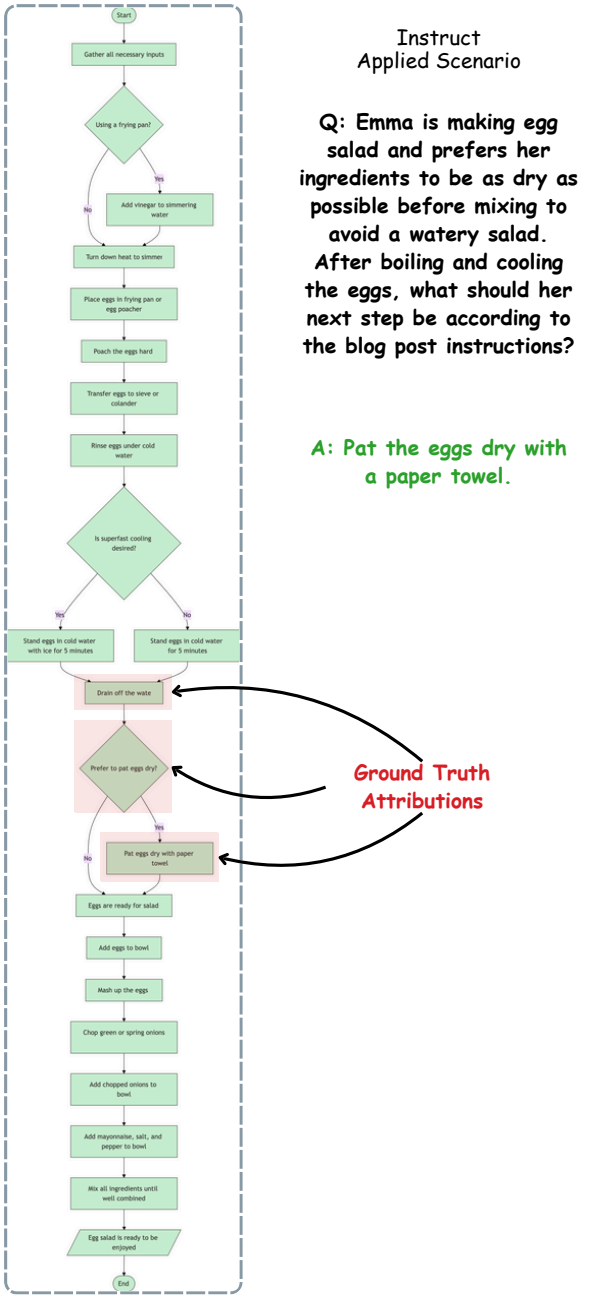


Figure 9: Example from `FlowExplainBench-Instruct`. This example represents an *Applied Scenario* question, and has a style type 1 (single color).

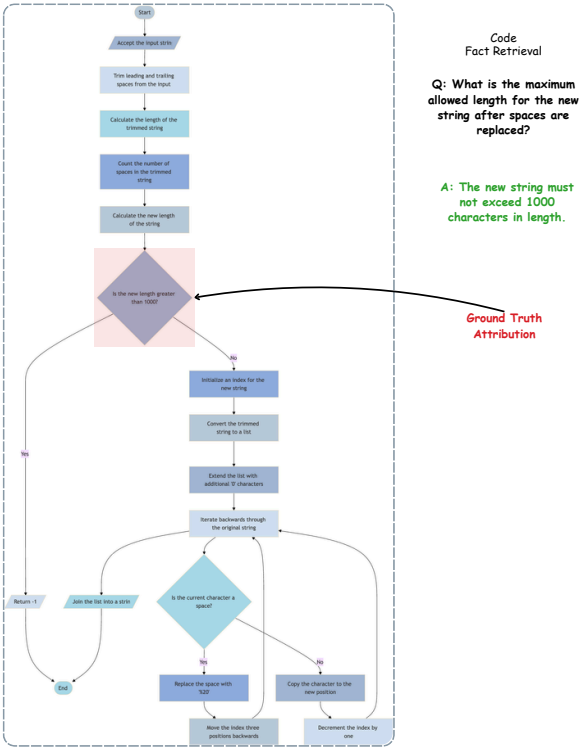
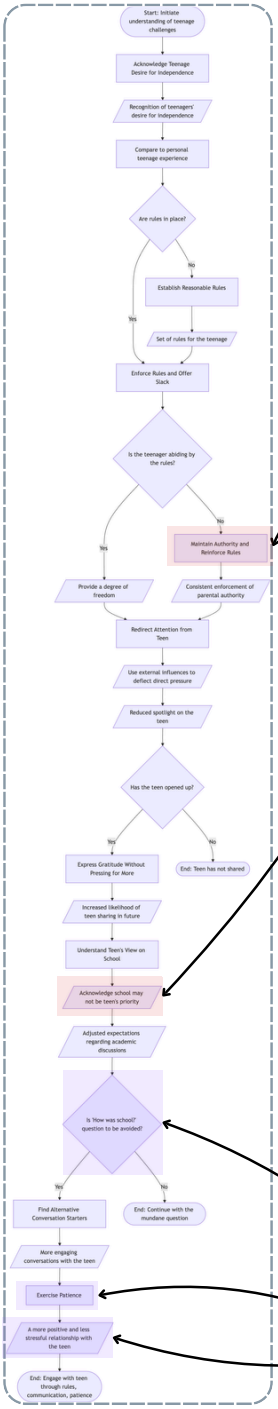


Figure 10: Example from `FlowExplainBench-Code`. This example represents a *Fact Retrieval* question, and has a style type 2 (multiple colors).



Wiki
Topological

Q: Is the node "Maintain Authority and Reinforce Rules" direct successor of the node "Acknowledge school may not be teen's priority"?

A: No, the node 'Maintain Authority and Reinforce Rules' does not directly succeed the node 'Acknowledge school may not be teen's priority'.

Ground Truth Attribution

Flow referential

Q: If a parent is currently at the step of exercising patience, what decision must have been made previously regarding the 'How was school?' question, and what is the expected outcome of this step?"

A: Previously, the decision to find alternative conversation starters was made, which results in a less stressful relationship with the teen.

Ground Truth Attribution

Figure 11: Example from `FlowExplainBench-Instruct`. This example represents *Tolopolgical* and *Flow Referential* questions, and has a style type 3 (mermaid default).

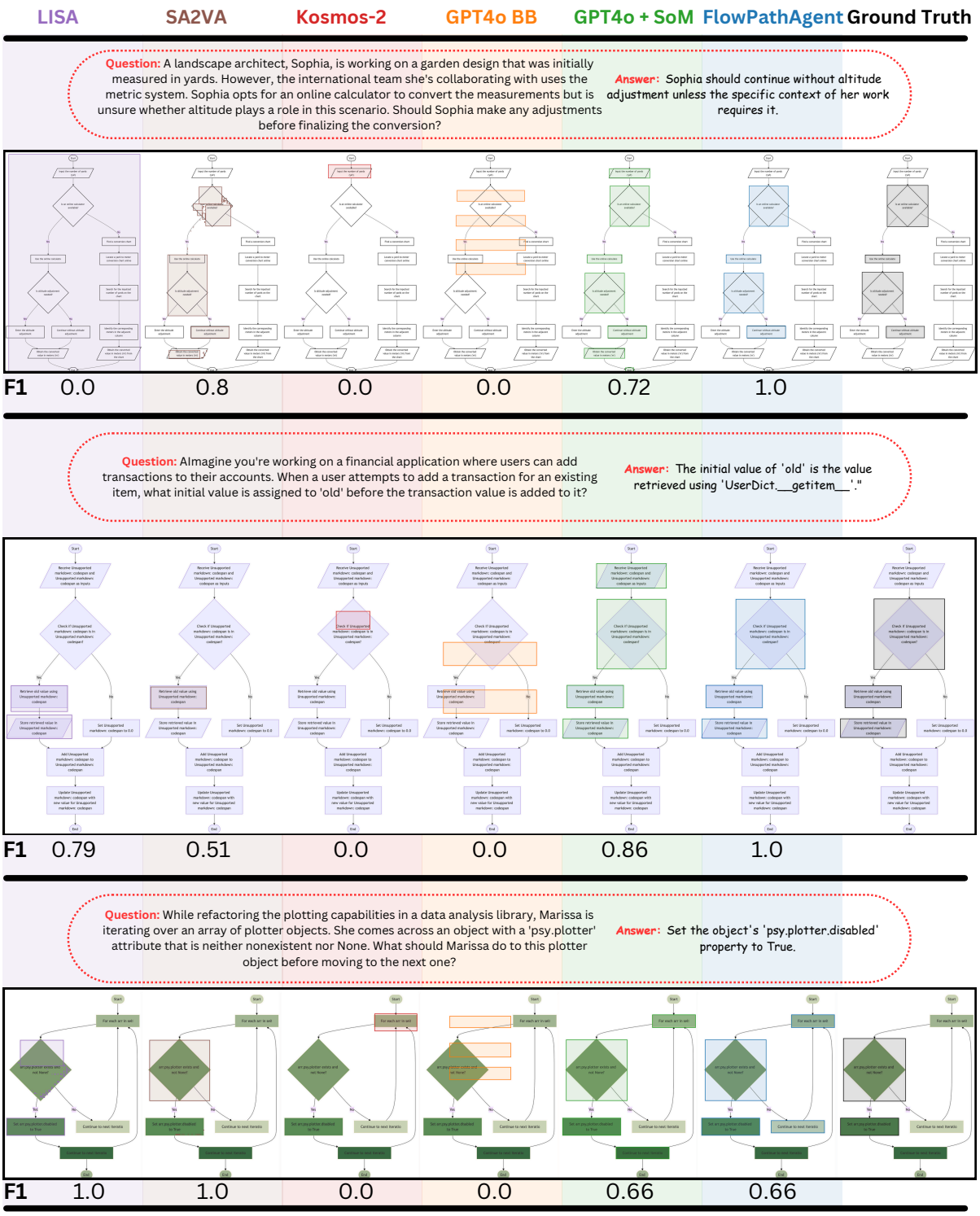


Figure 12: Qualitative comparison of `FlowPathAgent` with baselines via examples.

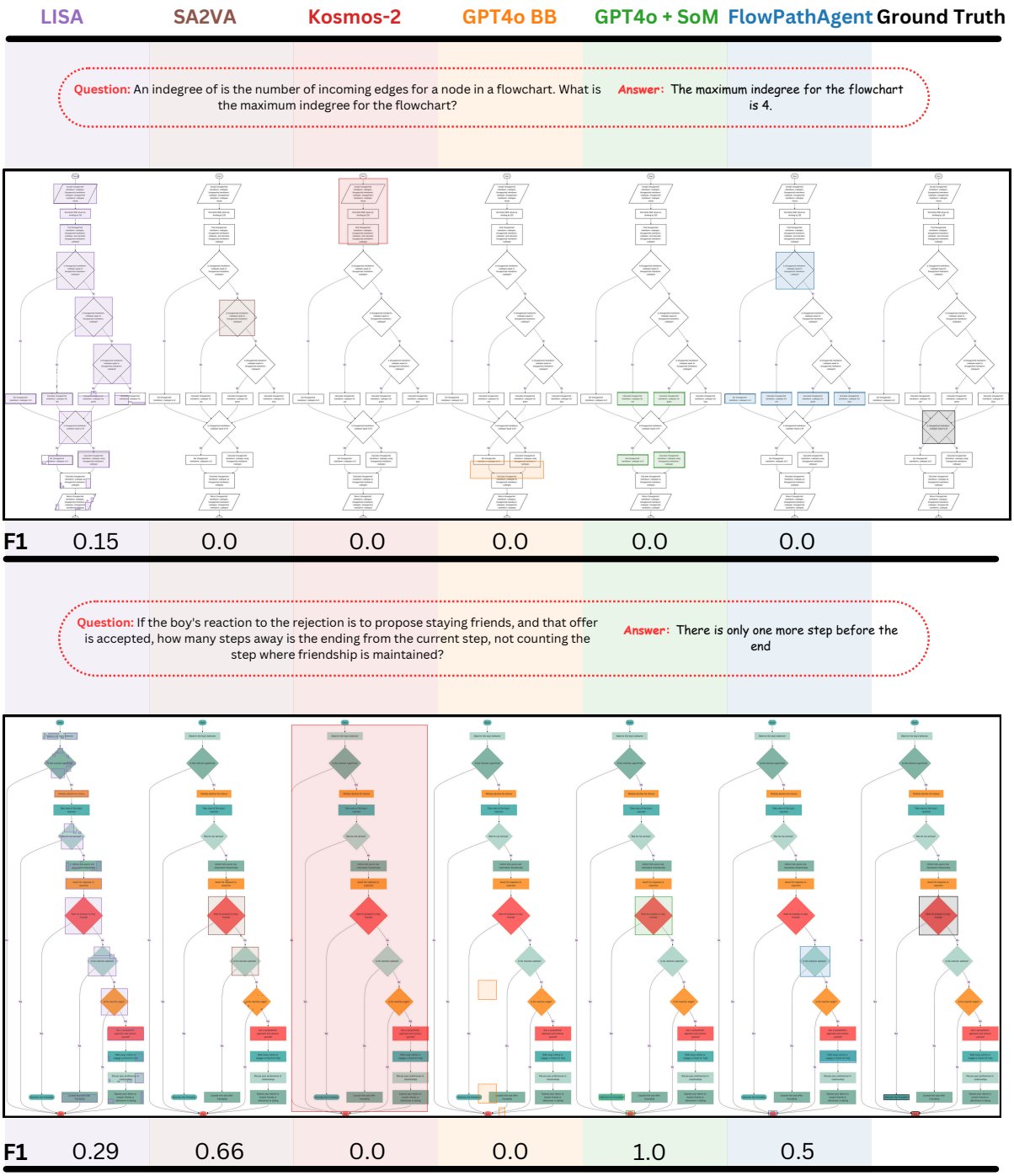


Figure 13: (Continued) Qualitative comparison of `FlowPathAgent` with baselines via examples.

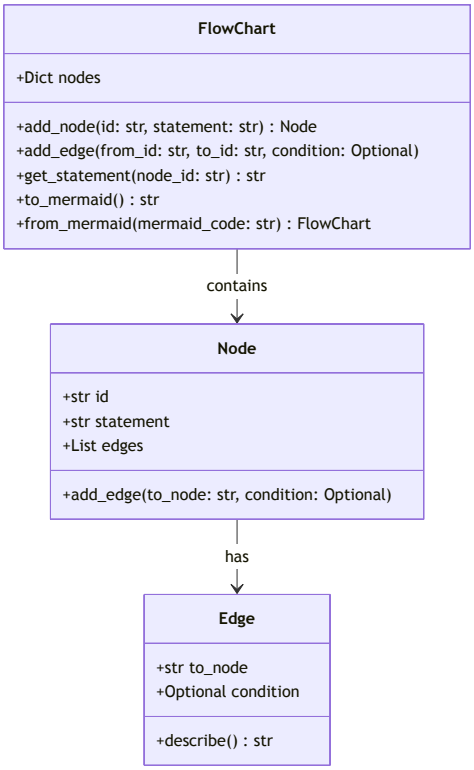


Figure 14: Class Diagram of the `FlowChart` data structure representing directed graphs with conditional edges.

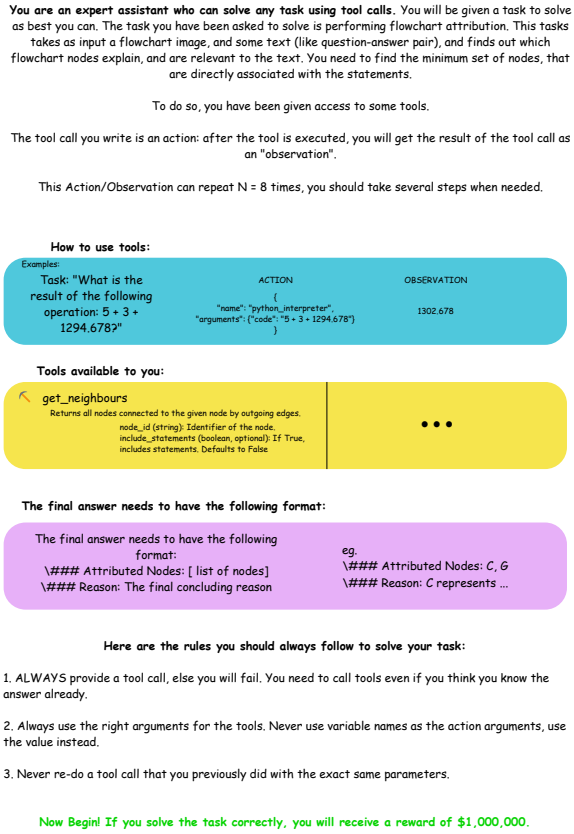


Figure 15: System prompt template provided to `FlowPathAgent`.

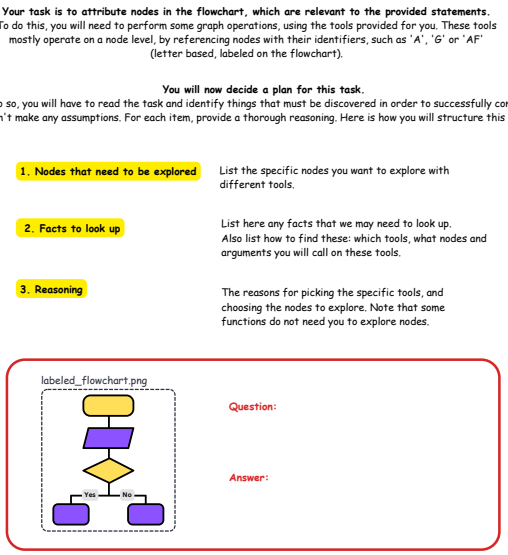


Figure 16: Planning prompt template provided to `FlowPathAgent`.

Figure 17: Prompt Template used for initial automatic ground truth annotation using GPT4.

Figure 18: Diversity of color schemes used to augment `FlowExplainBench` flowchart styles

Annotator Guidelines: Flowchart Attribution

1. Task Overview

Flowchart attribution involves identifying and selecting the relevant nodes in a flowchart that correspond to a given natural language statement. Annotators will interact with an attribution platform to highlight the appropriate nodes that form a logical path grounding the statement in the flowchart.

2. Annotation Process

Step 1: Understanding the Statement

- Carefully read the natural language statement provided.
- Identify key actions, decisions, or processes described in the statement.

Step 2: Examining the Flowchart

- Analyze the structure of the flowchart to understand the logical flow.
- Identify nodes that contain relevant operations or directives that match the statement.

Step 3: Selecting the Attributed Nodes

- Highlight nodes that directly contribute to fulfilling the statement's described process.
- Ensure the selected nodes follow a logical sequence, even if they are not directly connected.
- Use the following criteria to determine node inclusion:
    - **Optimality**: Select the minimal set of nodes required to fully attribute the statement.
    - **Contextual Alignment**: Ensure the selected nodes accurately represent the described actions or decisions.
    - **Exclusivity**: Avoid selecting unnecessary nodes that do not contribute to the statement's meaning.

Rubric for Flowchart Attribution

Criterion	Description	Score (0-2)
Optimality	The minimal set of nodes required to ground the statement is selected.	0 = Excessive or missing nodes, 1 = Some unnecessary nodes, 2 = Only essential nodes chosen
Contextual Alignment	The selected nodes logically represent the statement's described actions or decisions.	0 = Poor alignment, 1 = Partial alignment, 2 = Complete alignment
Exclusivity	No extra, irrelevant nodes are included.	0 = Many irrelevant nodes, 1 = Some irrelevant nodes, 2 = No irrelevant nodes

Scoring Interpretation:

- **5-6 points**: Excellent annotation
- **3-4 points**: Acceptable but may require minor adjustments
- **0-2 points**: Needs review and refinement

Figure 19: Summary of instructions given to human annotators.

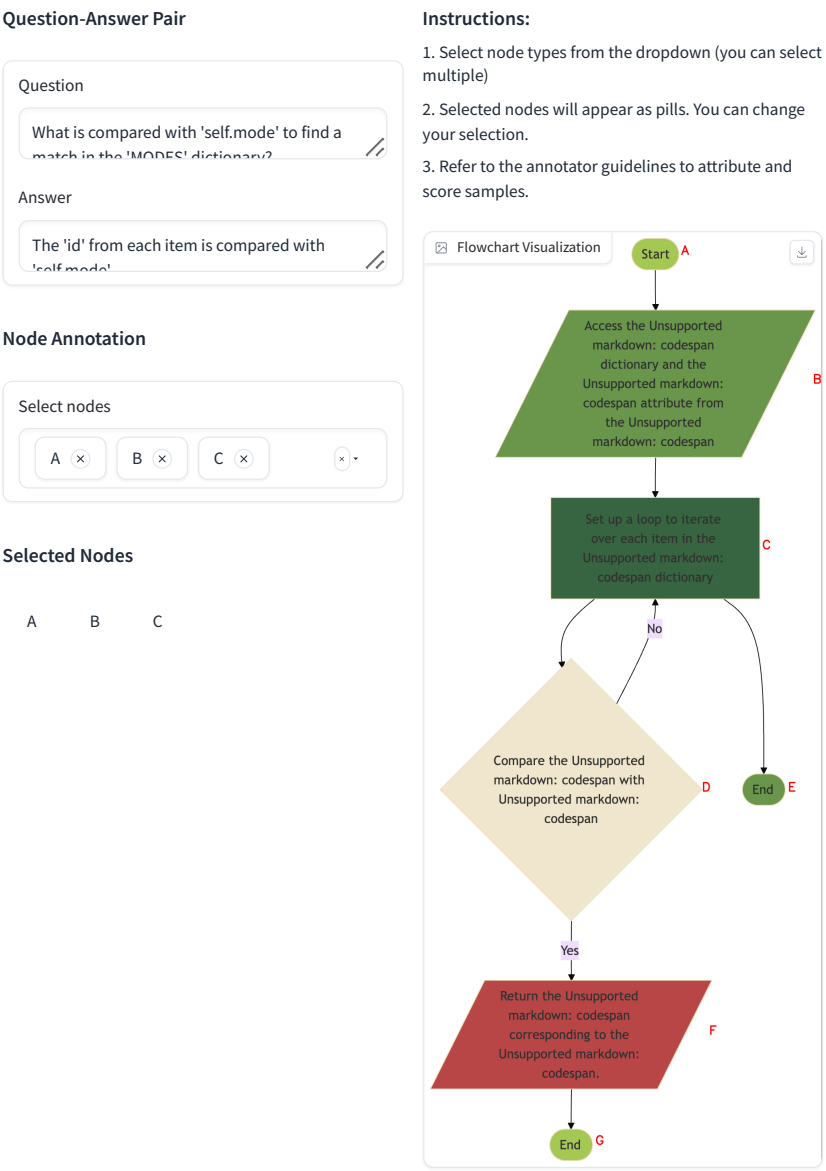


Figure 20: Human annotation platform for attribution annotation.