GenFlowchart: Parsing and Understanding Flowchart Using Generative AI

Abdul Arbaz[1], Heng Fan[1], Junhua Ding[1], Meikang Qiu[2], and Yunhe Feng[1]

[1] University of North Texas, Denton, TX 76207, USA
abdulkareemarbazabdulkareemarbaz@my.unt.edu,
{heng.fan, junhua.ding, yunhe.feng}@unt.edu
[2] Augusta University, Augusta, GA 30912, USA qiumeikang@ieee.org

Abstract. Flowcharts serve as integral visual aids, encapsulating both logical flows and specific component-level information in a manner easily interpretable by humans. However, automated parsing of these diagrams poses a significant challenge due to their intricate logical structure and text-rich nature. In this paper, we introduce *GenFlowchart*, a novel framework that employs generative AI to enhance the parsing and understanding of flowcharts. First, a cutting-edge segmentation model is deployed to delineate the various components and geometrical shapes within the flowchart using the Segment Anything Model (SAM). Second, Optical Character Recognition (OCR) is utilized to extract the text residing in each component for deeper functional comprehension. Finally, we formulate prompts using prompt engineering for the generative AI to integrate the segmented results and extracted text, thereby reconstructing the flowchart's workflows. To validate the effectiveness of *GenFlowchart*, we evaluate its performance across multiple flowcharts and benchmark it against several baseline approaches. *GenFlowchart* is available at https://github.com/ResponsibleAILab/GenFlowchart.

Keywords: Flowchart, OCR, Generative AI, GenAI, Image Segmentation, AI-Generated Content

1 Introduction

Flowcharts serve as essential tools in system requirement analysis, preliminary planning, and thorough design [19, 5]. Composed of symbols such as start/end, process/action, decision, connector/arrow, and loop/iteration symbols, they visually represent sequences of actions or steps in an activity or complex system. While their simplicity aids in conveying straightforward steps, complex solutions can make comprehension challenging [8]. In various domains, from computer science to business process management, flowcharts are widely used to visualize complex processes, logical flows, and decision-making pathways. Despite their human-readable nature, automating their parsing poses significant challenges due to the intricate logical relationships and textual information embedded within each component.

Despite the growing demand for automating complex processes, current methods for computerized flowchart parsing often fall short, either due to inaccuracies

or an inability to capture the essence and logic of these specialized diagrams. The core challenge lies in effectively integrating segmented geometrical shapes with extracted text while preserving the flowchart's original intent. To address these challenges, this paper proposes *GenFlowchart*, a novel framework that harnesses Generative Artificial Intelligence (GenAI), to enhance automated flowchart parsing and interpretation. The framework follows a structured approach, beginning with the utilization of a state-of-the-art segmentation model to identify and outline individual components and shapes within the flowchart. Optical Character Recognition (OCR) is then employed to extract textual information from each segmented component, enabling a deeper functional understanding. Furthermore, the framework incorporates generative AI, guided by meticulously crafted prompts, to assimilate the segmented results and extracted text.

We evaluate the effectiveness of the suggested approach by carefully analyzing many flowchart datasets. Moreover, we conduct a thorough performance study by comparing our methodology to the current baseline techniques. Our contributions are summarized as follows:

- We propose *GenFlowchart*, a generative AI enhanced flowchart understanding framework. The code of *GenFlowchart* is available at https://github.com/ResponsibleAILab/GenFlowchart.
- We curate and create a public flowchart dataset for flowchart evaluation.
- We compare *GenFlowchart* with several baseline models, such as Pytesseract [1] and SAM [7], to demonstrate the effectiveness of *GenFlowchart*.

2 Related Work

As flowcharts play an important role in diverse applications, many approaches have been proposed to understand and parse flowcharts automatically. For example, Supaartagorn [16] proposed a web application-based tool for code generation from flowcharts containing shapes like start/end, input, process, output, and decision. The extraction of flowcharts initially relied on OCR technology for data extraction and a neural network for segmenting and detecting components within flowcharts. Extracted data is organized and stored as metadata, in XML format, providing algorithmic explanations of flowcharts.

Raghu et al. [11] proposed an end-to-end approach to parse flowcharts using dialogues with GPT-2. It used retrieval-augmented generation architecture to get contextually appropriate responses from generative AI models and enhance the natural language understanding of the LLM. This paper also curated the FLODIAL dataset which contained 3000+ conversations and was a valuable resource for refining and advancing conversational AI agents.

Arrow R-CNN [13, 14] is designed for detecting hand-created flowcharts and enhancing the capabilities of Faster R-CNN [4], by incorporating a dedicated arrow key point predictor. This method improved the detection of connector features, crucial for understanding the flowchart's structure, while also predicting bounding box information for chart containers.

Besides AI-based approaches [9, 23, 21], many traditional methods have been developed to handle flowcharts [18, 17]. Traditional methods relied on predefined

symbols and flow allowing little room for deviation, while it ensured precision, it suffered from limited functionality due to its rigid nature. On the other hand, AI-based approaches offer a broader range of functionality and perform well for most flowcharts, thanks to their adaptive and flexible nature. In our paper, we propose *GenFlowchart* which leverages generative AI to enhance the comprehension of flowcharts. Our approach combines the strengths of AI-based techniques with traditional methods, offering a better solution by employing generative AI, to interpret and understand flowcharts with better accuracy and flexibility, addressing the limitations of previous methods.

3 Methodology

3.1 Framework Overview

GenFlowchart represents an AI-enhanced methodology that integrates Generative AI, AI-augmented visual segmentation, and Optical Character Recognition (OCR) into a cohesive workflow, as shown in Figure 1. This integration facilitates an advanced comprehension of flowcharts. The initial input for *GenFlowchart* encompasses images of flowcharts across a variety of types. These images undergo a preliminary processing stage, where they are converted into grayscale to mitigate any adverse effects of color on OCR performance and segmentation accuracy.

After this preprocessing, OCR technology is used which is renowned for its exceptional accuracy and efficiency, we employ a configuration with PSM (Page Segmentation Mode) 4 and OEM (OCR Engine Mode) 3, leveraging Tesseract [1, 15] coupled with LSTM-based engines [6]. This setup adeptly extracts textual content and accompanying locational metadata from images of diverse formats. For flowchart shape recognition, the Segment Anything Model (SAM) [7] is utilized on images stripped of text to identify the geometric configurations of flowchart components. SAM was chosen for its superior performance, as it has been trained on 11 million images and 1.1 billion masks. *GenFlowchart* then synergizes the textual data retrieved via OCR with the geometric details identified by SAM to decode the flowcharts' logic and content comprehensively.

Specifically, we engage GPT 3.5 Turbo to integrate textual and geometric data from flowcharts, facilitating a step-by-step comprehension of their logic. GPT-3.5 Turbo was selected for its advanced NLP capabilities and its ability to generate coherent and contextually relevant outputs. The proposed *GenFlowchart* leverages AI-driven techniques for natural language understanding and generation, thereby enabling a deeper and more nuanced analysis of the information conveyed by flowcharts.

3.2 Flowchart Image Preprocessing

Given the wide variety of flowchart images, characterized by differences in color, size, and textual density, our initial step involves preprocessing these images to ensure their accurate and efficient analysis. Flowchart images often feature vibrant visual components, including both text and shapes, which can adversely

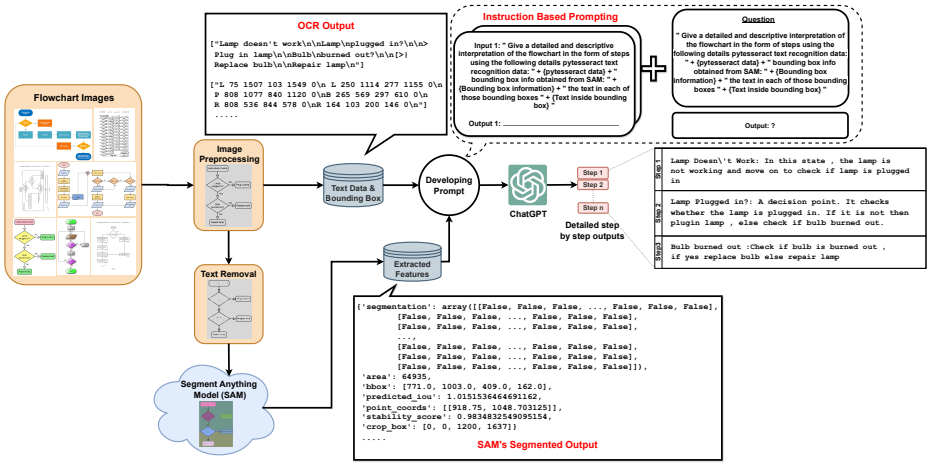

Fig. 1: Framework overview of *GenFlowchart*. Textual and geometric information from flowcharts is extracted using OCR and SAM, respectively. An approach to prompt engineering, including both zero-shot and instruction-based prompting techniques, has been devised to integrate this textual and geometric data, enabling the generation of a step-by-step understanding of the content and underlying logic of flowcharts.

affect the subsequent recognition of text and shapes within the flowchart components. To facilitate easier data interpretation and to mitigate the effects of color variability, *GenFlowchart* initially transforms the colorful flowcharts into grayscale images. Grayscale images are particularly advantageous for analyzing flowcharts as they accentuate key features and simplify the image analysis and understanding process.

Upon conversion to grayscale, the images are further processed to become binary images, consisting solely of black and white pixels. This step is particularly beneficial for flowchart analysis, where color detail is less critical, thereby enhancing computational efficiency through the simplification of the image format to a binary representation. The transition to binary images not only streamlines the model's efficiency but also ensures uniformity across all input images by employing image scaling techniques. This standardization facilitates the consistent processing of the dataset, ensuring that each image is analyzed uniformly.

To further refine the image quality, morphological operations and noise reduction techniques[3] are applied. These techniques play a crucial role in eliminating undesired artifacts and enhancing the clarity of the images, making them more conducive to accurate analysis. An illustrative comparison is provided in Figure 2, where the left panel presents the original image, and the right panel showcases the preprocessed outcome, highlighting the effectiveness of the preprocessing steps.

[3] https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html

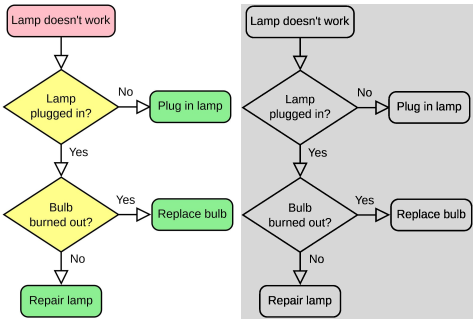

Fig. 2: Flowchart image preprocessing. The left subfigure shows the original flowchart, while the right subfigure depicts the flowchart post-preprocessing.

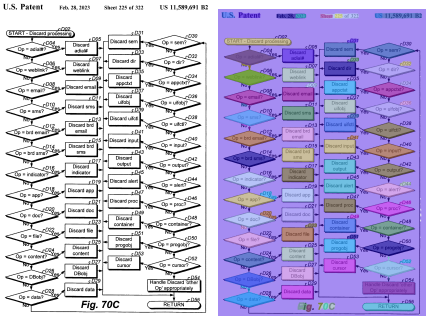

Fig. 3: SAM's performance on flowcharts. The left subfigure shows the original flowchart, while the right subfigure depicts the segmentation results by SAM.

3.3 Text Extraction and Removal

Following the preprocessing of flowchart images, *GenFlowchart* performs two critical text-related operations: text extraction and text removal. The extraction of text from flowchart images is an essential step, as understanding the content within these flowcharts is crucial. To this end, Pytesseract OCR is utilized for the extraction of text and the identification of their respective locations within the images, as demonstrated by the OCR outputs in Figure 1. This spatial information is pivotal for subsequent stages of flowchart image analysis.

The second operation involves the removal of textual elements from the flowchart images. While this approach may appear counterproductive at first glance, it is motivated by the challenges posed by the unique shapes of textual characters, such as 'O', 'P', 'Q', etc., which can impede the segmentation model's analytical capabilities. The English language's textual characters, with their diverse structures and resemblances to closed shapes, can significantly complicate the segmentation process. This complexity is particularly pronounced when employing the SAM model, known for its precision in identifying intricate features [7]. Furthermore, textual components, often similar in dimensions to other flowcharts or document elements, challenge the segmentation process due to the variability in font types, sizes, and layouts. Although the SAM model is highly capable of detecting fine details, precise text separation from other elements necessitates meticulous processing. Image processing techniques such as Median Blur [3] are applied to blur text-containing regions, thereby facilitating more accurate segmentation and enhancing the model's overall performance.

3.4 Image Segmentation and Analysis

Flowcharts usually consist of multiple visual components that are linked together using the flow arrows. Beyond only understanding the contents of the component, we also take care to record important information like the component's exact shape, its connection to previous parts, the bounding box information,

and the specific point coordinates that indicate its placement and orientation. The comprehensive segmentation procedure guarantees accurate flowchart component representation. It enables *GenFlowchart* to comprehend not only the written or graphical data inside each component but also their spatial and logical relationships, facilitating a greater knowledge of the flowchart's structure and content.

GenFlowchart achieves accurate image segmentation by utilizing the advanced capabilities of the SAM model. With this strategic process, we can carefully break down intricate flowchart images into discrete parts that allow for a thorough analysis of each element. Using SAM guarantees that our segmentation procedure is precise and sensitive to the details of the flowchart's visual components, improving *GenFlowchart*'s ability to decipher and work with its aspects. An example of its implementation is provided in Figure 3. Utilizing the SAM model, we delve into segment information and gather bounding box coordinates, facilitating logical parsing and enhancing comprehension of the flowchart's structure and content. Through the process of assigning a shape to each bounding box, the model can obtain a comprehensive grasp of the composition of the flowchart, allowing for a more in-depth and sophisticated examination of its content. This method plays a crucial role in deciphering the complex interaction between the written and visual components, leading to a deeper understanding of the flowchart's composition and significance.

3.5 Understanding Flowchart with Generative AI

Considering the heterogeneity of textual and image segmentation data extracted from flowcharts, it is challenging to transform that data into a meaningful and descriptive output. To solve this, we adopt Generative AI methods to create a detailed set of steps that explain the flowchart's logic and process by carefully creating instructional requests and prompts. Specifically, *GenFlowchart* uses ChatGPT (GPT-3.5 Turbo) APIs to map textual content and image segmentation together, generating meaningful descriptions of flowcharts. By combining the capabilities of natural language creation and interpretation, this method makes it feasible to convert a flowchart's visual representation into a legible text format. Another crucial component of the procedure is token management, which gives us effective control over the output's length and complexity. *GenFlowchart* can guarantee that the generated steps give a succinct but comprehensive explanation of the flowchart by adjusting the number of tokens, which will help close the gap between data and human perception. To guide ChatGPT to generate a high-quality explanation of a flowchart, *GenFlowchart* incorporates the following two prompt engineering methods.

Zero-shot Prompting LLMs have been extensively trained on massive amounts of data, enabling them to perform zero-shot prompting tasks with fundamental proficiency. Zero-shot prompting involves presenting a prompt to LLMs without the aid of example-based guidance, aiming to elicit a specific task execution [20]. Zero-shot learning seeks to generate an output from the LLM without furnish-

ing any contextual background for the prompt. The following is an example of a zero-shot prompt utilized in *GenFlowchart*:

> "Give a detailed and descriptive interpretation of the flowchart in the form of steps using the following details pytesseract text recognition data:" + {*pytesseract OCR data*} + "bounding box info obtained from Segment Anything Model (SAM):" + {*Bounding box information*} + "the text in each of those bounding boxes" + {*Text inside bounding box*}

Instruction-based Prompting Instruction-based prompting is a highly effective strategy for guiding LLMs like ChatGPT in text-generation tasks. By providing clear instructions or examples, this approach directs the model's output, shaping its behavior according to input directives. OpenAI's ChatCompletion technique facilitates organized dialogues, setting the stage for productive interactions [2]. Through a conversational structure involving input and instruction messages, the LLM is trained to understand nuanced responses. Contextual establishment enhances ChatGPT's ability to produce intelligent and contextually relevant outputs, imbuing the model with higher-level thinking and improving overall output quality for various applications. Below is the prompt used for instruction-based prompting in *GenFlowchart*:

> **System Instruction**: "You are a helpful assistant for flowchart understanding."
> **Input 1**: "Give a detailed and descriptive interpretation of the flowchart in the form of steps using the following details pytesseract text recognition data: " + {*pytesseract data*} + " bounding box info obtained from Segment Anything Model (SAM):" + {*Bounding box information*} + "the text in each of those bounding boxes" + {*Text inside bounding box*}
> **Output 1**: 1. Lamp Doesn't Work: In this state, the lamp is not working and move on to check if the lamp is plugged in 2. Lamp Plugged in?: A decision point. It checks whether the lamp is plugged in. If it is not then plugin lamp, else check if the bulb burned out. 3. Bulb burned out: Check if the bulb is burned out, if yes replace the bulb else repair the lamp.
> **Input 2**: .....
> **Output 2**: .....
> .....
> **Input N**: .....
> **Output N**: .....

4 Evaluation

4.1 Flowchart Dataset

We have collected a dataset of 550 flowcharts and used all of them to evaluate *GenFlowchart*. The dataset is compiled from flowcharts extracted from diverse patent documents in PDF format, covering diverse types of flowchart designs. Leveraging PyMuPDF[4], a high-performance Python library renowned for its efficiency in data extraction, analysis, conversion, and manipulation of PDFs, we develop a script specifically designed for extracting image flowcharts from these documents. To ensure reliable and accurate image analysis, we also recommend maintaining a resolution range of 300 to 600 DPI (dots per inch). This resolution range is chosen to facilitate precise and detailed image processing, thereby ensuring that the visual clarity of the flowcharts is preserved.

[4] https://pymupdf.readthedocs.io/

4.2 Evaluation Results

Our evaluation involves two primary components: text extraction and the description of the entire flowchart. While the Pytesseract library demonstrates high precision under normal conditions, its accuracy may falter when dealing with handwritten text. First, we evaluate the effectiveness of generative AI in enhancing text extracted via Pytesseract. Second, we provide a comprehensive explanation of the entire flowchart across various tasks and scenarios.

Evaluating AI-enhanced Text We conduct a comparative analysis between the outputs generated by Pytesseract and ChatGPT. This study deepens our understanding of Pytesseract results and offers insights into enhancing its performance using generative AI. We evaluate both the traditionally obtained Pytesseract results and the generative AI-enhanced outputs using the BERT Score metric [22], a robust measure for assessing sentence similarity. This metric calculates token similarity, providing crucial insights into text quality. Our results reveal that the generative AI-enhanced outputs outperform the standard Pytesseract outputs, as shown in Table 1. This finding underscores the effectiveness of GPT-3.5 in enhancing the precision and quality of text recognition outcomes. The ability of generative AI models to improve Pytesseract results suggests their significant potential in refining and optimizing text recognition tasks.

Table 1: Comparison of results of text recognition model.

Text Recognition Method	BERT-P	BERT-R	BERT-F1
Pytesseract	0.7553	0.8373	0.7940
GPT Enhanced Pytesseract	0.8029	0.8388	0.8200

Evaluating Logical Description of Flowcharts In a study on Web-based flowchart description, the code generated from flowcharts was manually reviewed using functionality, usability, and performance tests, resulting in an overall score of 4.27 out of 5, indicating a high level of satisfaction [16]. Inspired by this work, our research concentrates on evaluating flowchart explanations, involving a manual assessment of 550 flowchart images to determine satisfaction levels. Ratings ranged from 0 to 5, with 0 indicating no output, 1 denoting output unrelated to the flowchart, 2 representing partial summary (30-50% accuracy), 3 indicating moderate accuracy (50-70% accuracy), 4 representing high accuracy (70-90% accuracy), and 5 indicating complete accuracy. Our model achieved an average satisfaction level of 4.24, as seen in Table 2.

Ablation Study An ablation study was conducted to understand the significance of the SAM in the model's understanding of flowcharts. The study included

the details of the pytesseract text recognition model and the shapes in the image, similar to our previous experiment. The results of the study revealed a significant drop in the model's understanding to 3.14 points out of 5 of its evaluation without the inclusion of SAM (see the bottom line in Table 2). This suggests the crucial role of the model in accurately identifying objects in the flowchart to understand the flowchart's overall flow. Therefore, the study highlights the importance of SAM in aiding the model's ability to comprehend effective communication through flowcharts. By including SAM, the model can focus on the relevant areas of the image crucial to its understanding, resulting in better accuracy and interpretation.

Table 2: Satisfaction level on various modes of implementation.

Method	**Satisfaction Level**
Web Application Based Code Generation [16]	4.27
Zero-shot Prompting	4.24
Instruction-based Prompting	4.68
Ablation Study (W/o SAM)	3.14

Bert Score The later part of the evaluation is done using non-manual metrics. For that, we have chosen the Bert Score as one of the metrics. Our model performed best in Instruction-based prompting, where the SAM model's integration yielded superior results. This highlights the effectiveness of our approach, evident in the precision (BERT-P), recall (BERT-R), and F1 (BERT-F1) score [22]. Leveraging the BERT Score confirmed the model's capacity to capture semantic nuances and contextual significance, facilitating nuanced evaluation. This validation shows the ability of *GenFlowchart* to generate high-quality, contextually relevant text, further bolstering its credibility and efficacy, as shown in Table 3.

Table 3: BERT Score of each of the implementations.

Method	**BERT-P**	**BERT-R**	**BERT-F1**
Zero-shot Prompting	0.8333	0.8314	0.8468
Instruction-based Prompting	0.8654	0.8648	0.8649
Zero-shot Prompting (w/o SAM)	0.8027	0.8249	0.8184
Instruction-based Prompting (w/o SAM)	0.8320	0.8242	0.8278

Cosine Similarity - word2vec The word2vec [10] is a technique for obtaining the vector representation for a particular word and cosine similarity is a key metric for assessing the similarity between sentences or paragraphs in natural language processing, with scores ranging from -1 to 1. We measure the similarity

of two vectors in a multi-dimensional space to determine phrase similarity. In this context, sentences are represented as numerical vectors, where each dimension represents a unique element or attribute of the phrase. We obtained a score of 72.38% in cosine similarity which illustrates the effectiveness of instruction-based prompting in producing high-quality outputs. Table 4 highlights the effectiveness of the prompting which yields outputs that closely align with target criteria or specifications. By utilizing this as an assessment measure, we gain valuable insights into the effectiveness of different prompting strategies, particularly highlighting the advantages of instruction-based approaches in achieving desired results.

Table 4: Cosine Similarity Score (word2vec) of each of the implementations.

Method	**Cosine Similarity (word2vec)**
Zero-shot Prompting	0.6505
Instruction-based Prompting	0.7283
Zero-shot Prompting (W/o SAM)	0.6438
Instruction-based Prompting (W/o SAM)	0.6536

Cosine Similarity - Sentence Transformers We also utilize the Sentence Transformer model [12] from HuggingFace to assess the similarity between flowchart descriptions. A Sentence Transformer operates in two phases:

- The text is passed through a pre-trained transformer from HuggingFace, which produces contextual embeddings for the input text.
- The embeddings then pass through a pooling layer to obtain fixed-length embeddings.

Unlike word2vec, Sentence Transformers generate embeddings for entire sentences or paragraphs rather than individual words, thus providing more contextual information. Using Sentence Transformers, we assess the effectiveness of various prompting strategies in producing desired results. Among these strategies, the instruction-based prompting method yields the highest similarity score, reaching an impressive 76.25%.

Table 5 highlights the effectiveness of instruction-based prompting over other methods due to providing better context, although acknowledging limitations, especially in handling multi-sentence contexts. We also see how various evaluation metrics compare with each other as seen in Figure 4. While sentence transformers show promise as a metric, further research is needed for broader applicability. By combining assessment techniques like word2vec - Cosine Similarity and BERT Score, we gain comprehensive insights into the model's functionality, validating its effectiveness and identifying areas for improvement. This approach offers a holistic understanding of the model's strengths and weaknesses, guiding ongoing optimization efforts.

Table 5: Cosine Similarity Score (Sentence Transformers) of each of the implementations.

Method	Cosine Similarity (Sentence Tx.)
Zero-shot Prompting	0.7111
Instruction-based Prompting	0.7625
Zero-shot Prompting (W/o SAM)	0.6996
Instruction-based Prompting (W/o SAM)	0.7031

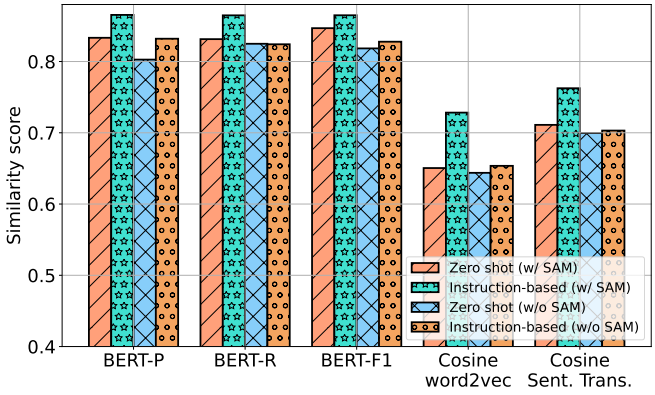

Fig. 4: Comparing results of evaluation methods

5 Conclusion

This paper introduces *GenFlowchart*, a novel framework designed to enhance the automated parsing and understanding of flowcharts using generative AI. The challenges associated with the intricate logical structure and text-rich nature of flowcharts are effectively addressed through a combination of advanced segmentation techniques, OCR, and prompt engineering. The integration of the Segment Anything Model and GPT-3.5 Turbo demonstrates significant improvements in the accuracy and comprehension of flowchart components and their interrelationships. The evaluation of *GenFlowchart*, utilizing a dataset of 550 diverse flowchart images, showcases its superior performance compared to baseline approaches. The comprehensive analysis reveals that generative AI can significantly enhance text recognition outputs, as evidenced by the improved BERT Scores and cosine similarity metrics. Furthermore, the incorporation of instruction-based prompting, combined with SAM, leads to a notable increase in the accuracy of flowchart interpretations.

References

1. pytesseract — pypi.org. `https://pypi.org/project/pytesseract/`. [Accessed 16-01-2024].

2. Tariq Alhindi, Tuhin Chakrabarty, Elena Musi, and Smaranda Muresan. Multitask instruction-based prompting for fallacy recognition. *arXiv preprint arXiv:2301.09992*, 2023.
3. Gary Bradski, Adrian Kaehler, et al. OpenCV: Open source computer vision library. `https://opencv.org/`, 2020. Accessed: 2024-01-10.
4. Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.
5. David Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.
6. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
7. Alexander Kirillov, Eric Mintun, et al. Segment anything. In *IEEE/CVF CVPR*, pages 4015–4026, 2023.
8. Donald E Knuth. Runcible—algebraic translation on a limited computer. *Communications of the ACM*, 2(11):18–21, 1959.
9. C. Ling, J. Jiang, et al. Deep graph representation learning and optimization for influence maximization. In *ICML*, 2023.
10. Tomas Mikolov, Ilya Sutskever, et al. Distributed representations of words and phrases and their compositionality. *Advances in neural infor. proc. sys.*, 26, 2013.
11. Dinesh Raghu, Shantanu Agarwal, Sachindra Joshi, et al. End-to-end learning of flowchart grounded task-oriented dialogs. *arXiv preprint arXiv:2109.07263*, 2021.
12. Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.
13. Bernhard Schäfer, Margret Keuper, and Heiner Stuckenschmidt. Arrow r-cnn for handwritten diagram recognition. *International Journal on Document Analysis and Recognition (IJDAR)*, 24(1):3–17, 2021.
14. Bernhard Schäfer and Heiner Stuckenschmidt. Arrow r-cnn for flowchart recognition. In *Intl. Conf. on Docu. Analysis and Recog. Workshops (ICDARW)*, volume 1, pages 7–13, 2019.
15. Joshua A Suitter. Accuracy of optical character recognition software google tesseract. 2015.
16. Chanchai Supaartagorn. Web application for automatic code generator using a structured flowchart. In *8th IEEE Intl. Conf. on Software Engi. and Service Sci. (ICSESS)*, pages 114–117, 2017.
17. Axel Winkelmann and Burkhard Weiß. Automatic identification of structural process weaknesses in flow chart diagrams. *Business Process Management Journal*, 17(5):787–807, 2011.
18. Xiang-Hu WU, Ming-Cheng QU, et al. A code automatic generation algorithm based on structured flowchart. *Appl. Math*, 6(1S):1S–8S, 2012.
19. Stelios Xinogalos. Using flowchart-based programming environments for simplifying programming and software engineering processes. In *2013 IEEE Global Engineering Education Conference (EDUCON)*, pages 1313–1322. IEEE, 2013.
20. Wenpeng Yin, Jamaal Hay, and Dan Roth. Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach. *arXiv preprint arXiv:1909.00161*, 2019.
21. Y. Zeng, M. Pan, et al. Narcissus: A practical clean-label backdoor attack with limited information. In *ACM CCS*, 2023.
22. Tianyi Zhang, Varsha Kishore, Felix Wu, et al. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*, 2019.
23. Y. Zhang et al. Communication-efficient stochastic gradient descent ascent with momentum algorithms. In *IJCAI 2023.*, 2023.