Repairing Responsive Layout Failures Using Retrieval Augmented Generation

***Abstract*—Responsive websites frequently experience distorted layouts at specific screen sizes, called Responsive Layout Failures (RLFs). Manually repairing these RLFs involves tedious trial-and-error adjustments of HTML elements and CSS properties. In this study, an automated repair approach, leveraging LLM combined with domain-specific knowledge is proposed. The approach is named ReDeFix, a Retrieval-Augmented Generation (RAG)-based solution that utilizes Stack Overflow (SO) discussions to guide LLM on CSS repairs. By augmenting relevant SO knowledge with RLF-specific contexts, ReDeFix creates a prompt that is sent to the LLM to generate CSS patches. Evaluation demonstrates that our approach achieves an 88% accuracy in repairing RLFs. Furthermore, a study from software engineers reveals that generated repairs produce visually correct layouts while maintaining aesthetics.**

***Index Terms*—Responsive Web Design, Automated Program Repair, RAG-based tool, LLM, Stack Overflow**

I. Introduction

Designing a webpage responsive to all screens is challenging due to limited screen space, causing elements to overlap, overflow their containers, or extend beyond the visible area. These issues, known as Responsive Layout Failures (RLFs), are often difficult to detect and fix since they require extensive trial and error and modifications to multiple HTML elements and CSS properties [5]. An automated solution that repairs layouts similar to human developers while preserving the original design would be beneficial.

In recent years, researchers have proposed techniques to detect, localize and repair RLFs. Walsh et al. [6] first introduced a technique to automatically detect RLFs by comparing the positioning of elements relative to one another at different screen sizes. LocaliCSS [7] offers fine-grained localization by identifying the specific elements and CSS properties responsible for a detected RLF. To repair the detected layout issues, Layout DR [5] creates hotfixes by modifying the CSS properties of the entire layout. However, existing approaches failed to combine localization with targeted repairs, causing inefficient repair solutions.

Recently, Large Language Models (LLMs) have shown promise in automated program repair [8]. Hence they can be applied to repair RLFs as well, however, they often lack developer domain knowledge and in-built mechanism for developer validation or correction. On the other hand, popular developer Q&A platforms such as Stack Overflow (SO) provide a vast space for knowledge sharing [9]. SO allows developers to access solutions to problems faced by others and facilitate the exchange of expertise between developers.

In this paper, we propose ReDeFix (Responsive Design Fix) [1], a Retrieval-Augmented Generation (RAG)-based approach for repairing RLFs. A knowledge base from SO discussions is constructed to enhance the quality of LLM-generated solutions. ReDeFix begins by localizing the faulty elements and CSS properties using LocaliCSS [7]. Next, it retrieves relevant SO discussions to help LLM understand how developers typically approach repairs. These discussions, along with the specific RLF context, are then combined into a prompt for LLM to generate an effective repair. This repair process is iterative, incorporating feedback after each attempt until a correct patch is produced.

Evaluation on 13 responsively designed webpages shows that ReDeFix successfully repaired 38 out of 43 RLFs, achieving an 88% repair accuracy. To measure the impact on layout and aesthetics, a study with software engineers (SE) was performed. It revealed that, 85% of the repaired layouts were preferred as correct, and 70% were deemed aesthetically pleasing. This implies that the generated patches are effective and rarely degrade layout and aesthetics. Moreover, ReDeFix improved 11 out of 12 original incorrect layouts to correct states and enhanced aesthetics for 9 out of 11 non-aesthetic layouts. This confirms its ability to repair accurately without degrading layout quality.

II. Literature Review

Detecting different types of layout failures in web applications has been extensively explored in the literature, including *Responsive Layout Failures* (RLFs). However, comparatively there has been less focus on automated approaches to repair them.

For responsively designed web pages, Walsh et al. [6] first introduced a method to automatically detect RLFs using the *Responsive Layout Graph* (RLG). It has HTML elements as nodes and their relationships as edges. This method compares RLGs of a page's old and new versions using ReDeCheck [10] to detect failures, but requires both versions.

Addressing this, to detect RLFs, Walsh et al. [11] later

presented another version of REDECHECK where the layout of a responsive webpage is compared against itself. It compares the relative positioning of elements at different screen sizes, also known as viewport widths. This study introduced five types of RLFs prevalent in real-world web pages. These types have been extensively used in the following works, including this paper. They are, *Element Collision, Element Protrusion, Viewport Protrusion, Small-Range* and *Wrapping Elements*. Although REDECHECK reliably detects these types of RLFs, it does not provide automatic repair of them.

For localizing the properties causing an RLF, Tasmia et al. [7] proposed a method that heuristically searches for HTML elements and CSS properties potentially responsible to cause an RLF. The output of their approach is a ranked list, consisting of elements and their properties that have the most impact on an RLF.

To repair RLFs, Althomali et al. [5] proposed a tool called LAYOUT DR. It sources layouts from either side of the affected viewport range, free from RLFs. The layouts are scaled and transformed within the failure range to create two *"hotfixes"*. Since the entire layout is modified, it causes a change to almost every property value explicitly defined by developers.

On the other hand, recent advancements in automated program repair have benefited significantly from LLM-based approaches [12], [13]. While LLMs have improved code generation and repair tasks, these models are retrained infrequently, generate solutions non-deterministically, and often lack mechanisms for combining community-driven knowledge [14]. To overcome these limitations, recent studies have explored Retrieval-Augmented Generation (RAG), which leverages external knowledge sources such as Stack Overflow and other Q&A platforms [15]–[17]. However, repairing RLFs using LLMs remains an unexplored area.

III. METHODOLOGY

In this study, we propose a novel RAG-based system to repair RLFs by leveraging SO as a data source. As shown in Figure 1, our approach starts with the problematic HTML elements and CSS properties as input. To address these RLFs, relevant SO queries, with their answers and comments are retrieved from the knowledge base to provide additional context on the repair styles of developers. Then, LLM is prompted to fix the failure along with the additional context of SO queries.

A. Knowledge Base Construction

We utilized SO to extract relevant RLF-related questions, their answers and comments. SO is potentially useful to repair many failures as it contains millions of posts with years of accumulation [18]. We searched questions based on two tags, `CSS` and `HTML`, as majority of the RLF-related questions have one of these tags associated with them. To identify relevant RLF discussions, we used RAKE algorithm [19] to extract keywords from the definition of each RLF mentioned in [7]. To search each RLF-specific SO questions, we chose the keywords that uniquely identify an RLF type, e.g., *elements collide* for element collision, *appear outside screen* for viewport protrusion. We then broaden the range of questions by adding similar keywords, such as *elements overlap* alongside *elements collide* as *overlap* is a synonym for *collide*. As SO has too many queries in various formats, we couldn't analyze them all. Therefore, we created this list to specifically identify a large, relevant set of questions for each RLF within the scope of our study.

For each RLF type, after applying these filters, we extracted the answers along with all their associated comments. Questions having no answers and comments were discarded. Similarly, answers having a SCORE of 0 or less than 0 were considered non-verified and thus discarded. Lastly, to find relevant answers and comments, we filtered the ones having mention of any of the RLF-specific properties. Since all collected questions are originally in HTML format, we performed standard data cleaning procedures. We removed all HTML tags except for `<code>` tags, which were preserved to maintain the integrity of code blocks within the posts.

These results are stored in the vector database as documents, where each RLF type contains a separate set of SO results. Each of the results contains the question's ID, LINK, TITLE and BODY in its metadata. Our knowledge base contains 334 extracted questions, with 522 answers and 1855 comments. This serves by capturing collective developers' feedback while they faced similar issues during the repair of RLFs.

B. Retrieval System

Given a list of problematic HTML elements and CSS properties, we retrieve discussions related to these properties from our knowledge base (Figure 1). Here we combine the problematic property names as a query to find repair discussions only for those properties. The query is used to retrieve the top five questions. Then with each question, its answers and associated comments are included as context and sent with the prompt.

We have used an Ensemble Retriever [20], combining BM25 [21] and VectorStore Retrievers [2]. This ensembling enables to use both, dense (VectorStore) and sparse (BM25) retrievers. The sparse retriever is good at finding relevant documents based on keywords, while the dense retriever is good at finding relevant documents based on semantic similarity. We selected BM25 because prior studies [22] have demonstrated its effectiveness for code-to-code retrieval. On the other

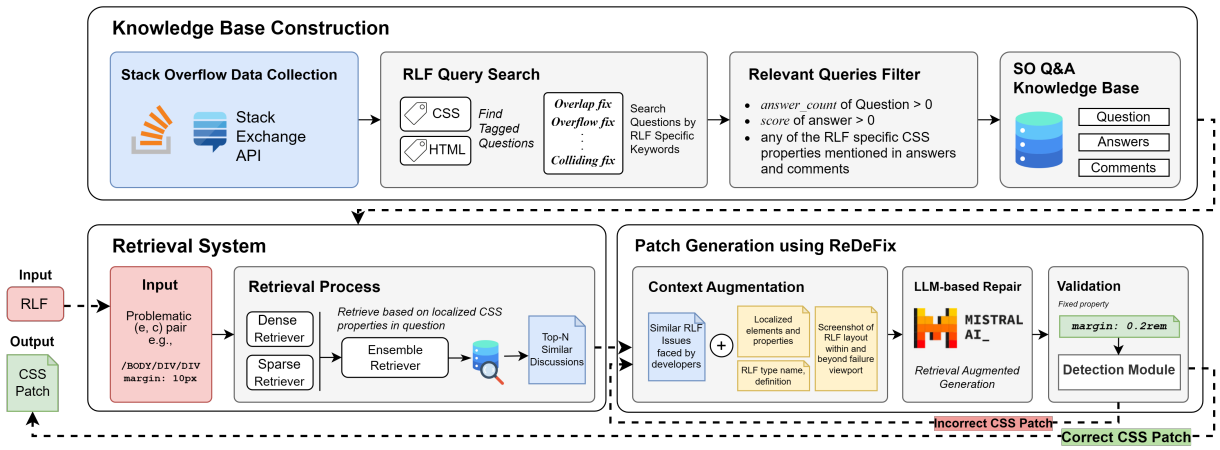

Figure 1: Overview of ReDeFix

hand, to find the properties in the discussion text, similarity search of VectorStore can be very effective.

C. Patch Generation and Validation

The main goal of this component is to augment the original prompt with similar issues collected from SO. The original prompt contains the context of an RLF, which includes the involved elements and properties, RLF type, its definition as stated in [7], the failure elements with their coordinates, and screenshots of the RLF segment inside and outside of the failure viewport range. These components are obtained by using the localization approach of LocaliCSS [7]. The target of this context is to help LLM understand the layout and generate a patch accordingly. The augmented context is then sent as a prompt to the LLM. The prompt used in our approach involves five important components as illustrated in Figure 2:

1. Role Designation: `You are an automated program repair tool which works as an expert in CSS and HTML.`

2. Task Description: `Fix the following responsive layout failure (RLF) using the provided context:`

3. RLF Context: `RLF type, type definition, failure element XPaths, failure region coordinates, failure viewport range, localized properties which are causing the failure (ranked from most to least problematic), screenshot of the failure region and failure free layout`

4. Relevant Stack Overflow Posts: `Example Stack Overflow threads to help you understand how developers solve these failures.`

5. CoT Indicator: `Let's think step by step.`

Figure 2: Example Prompt

- Role Designation: Our approach starts with an instruction to assign LLM as an automated program repair tool [23]. The assigned role provides context about the LLM's identity and background.

- Task Description: LLM is provided with the description as illustrated in Figure 2.

- RLF Context: Our approach provides LLM with the components working as context for the RLF.

- Relevant SO Posts: The retrieved SO posts are augmented with the original prompt to add external knowledge in the repair process.

- Chain-of-Thought (CoT) Indicator: LLM is instructed to think step-by-step to fix an RLF. Here, we follow the best practice to use CoT for enhancing the reasoning of LLM [24] and adopt the same prompt named "`Let's think step by step`".

Following this, we can obtain the model output, which contains the thought process of LLM and the candidate patch created with the fixed properties of the problematic elements. These will be verified in the validation step as shown in Figure 1.

The generated patch is injected into the CSS of the target page by first creating a selector for each affected HTML element. To ensure these new rules override existing styles, every property in the patch is marked with `!important`. To restrict the patch to the failure range, the element and its properties are encapsulated within a media rule spanning the failure range. This patch is added to the webpage and then sent back to the detection module, which verifies that the original RLF has been eliminated without introducing any new layout failures. If both conditions are met, the patch is accepted. Otherwise, ReDeFix reconstructs the prompt and appends the failed patch to the end of the original prompt. Here, the failing information is added into

a template: "`The fixed version is still not correct-{last generated patch}. Please fix it again. Let's think step by step.`" Then, ReDeFix interacts with LLM using the new prompt to generate a new fixed solution. This iterative process continues until a correct patch is obtained or the prompt exceeds its maximum token limit.

IV. Experimental Setup and Result Analysis

We designed an empirical study to evaluate the performance of the proposed ReDeFix by investigating two research questions (RQs):

RQ1: *How accurate is ReDeFix in repairing responsive layout failures in webpages?*

RQ2: *How do users perceive the quality of the repaired version generated by ReDeFix?*

A. Subjects

We selected our test subjects from the set of web pages previously used by LocaliCSS [7]. Their study localized 58 failures across 20 webpages; however, we excluded small-range failures since these require only simple media-query adjustments. Furthermore, the study of LocaliCSS revealed that some failures were marked as No Problem (NP) by manual inspection. Additionally, some pages had a single RLF, which was later labelled as NP. Excluding above mentioned cases leaves us 13 webpages and a total of 43 RLFs for the evaluation.

B. Answer to RQ1

1) ***Experimental Setup****:* To evaluate the accuracy, we ran our approach on each of the 43 RLFs. Our automated validation phase gave us an output of the successful and failed repairs. Besides this, a manual inspection was performed. As one failure can have multiple ways to fix it, instead of having a manually annotated ground truth, the author added each generated patch to its corresponding webpage and inspected whether it fixes the RLF or not, without introducing any new RLFs.

We implemented our approach by using LangChain framework [25] with the API endpoint of Mistral Small 3.1 (24B) [3]. It is a lightweight open-source LLM model, having multimodal understanding and an expanded context window of up to 128k tokens. We used the default values for all the hyper-parameters for our LLM model. For retrieval, we combined BM25 and semantic similarity search in an ensemble and experimentally evaluated various weight combinations to collect the most relevant discussions. This resulted in weights of 0.8 and 0.2, respectively. Finally, we retrieved SO posts via the Stack Exchange API [4], ensuring access to the up-to-date content.

TABLE I: Evaluation of ReDeFix on different settings

Subject Name	Total RLFs	Zero-Shot Repair	ReDeFix Repair
3MinuteJournal	4	4	**4**
Ardour	2	2	**2**
Bower	3	3	**3**
BugMeNot	3	2	**3**
Django	3	1	**3**
DjangoREST	1	0	**0**
HotelWiFiTest	1	1	**1**
MantisBT	6	3	**6**
MidwayMeetup	1	0	**0**
PepFeed	5	1	**2**
PDFescape	2	2	**2**
Selenium	6	4	**6**
WillMyPhoneWork	6	3	**6**
Total	**43**	**26 (60.4%)**	**38 (88.3%)**

2) ***Results****:* Our approach successfully repaired 38 out of 43 RLFs, as detailed in Table I. To evaluate the impact of augmenting the LLM with external knowledge, we compared the baseline LLM using zero-shot reasoning against the same LLM enhanced with additional context retrieved from our Stack Overflow knowledge base.

Table I shows promising results for RAG-based LLMs. Specifically, LLM's repair accuracy increased from 60.4% in the zero-shot setting to 88.3% using RAG in ReDeFix. Zero-shot patches frequently lacked advanced CSS properties and introduced unnecessary additional properties alongside the required changes. Incorporating developer knowledge from SO significantly improved the LLM's ability to adopt developer perspective and generate accurate repairs with essential properties only. One element protrusion RLF was repaired by adding `box-sizing: border-box`, but zero-shot patch was missing this property, thus unable to repair.

However, ReDeFix failed to repair 5 RLFs as seen in Table I. Here, for *DjangoREST* RLF, our generated repair distorted the entire layout and introduced a new failure. In one of the three *PepFeed* RLFs, it produced a hallucinated repair using a class from SO discussion. It failed to remove the original RLF in the other two RLFs and the RLF of *MidwayMeetup*. Manual inspection revealed that these RLFs were not properly localized by LocaliCSS [7], hence localized properties were absent. Our repair approach achieves 95% accuracy when cases failing due to LocaliCSS limitations are excluded. Hence, we can conclude that using a RAG with SO knowledge base enhances LLMs in code repair.

C. Answer to RQ2

1) ***Experimental Setup****:* To evaluate the quality of our proposed approach, we conducted a user study with five front-end software engineers (SE) having 1 to 5 years of professional experience.

Initially, we had 43 RLFs across various elements and viewport ranges. Some RLFs appeared multiple times on different elements, and others occurred within the same viewport range. For the user study, we combined such RLFs, and omitted the ones that we were unable to repair. This resulted in 20 unique pairs of layout screenshots to assess in our survey. Each pair featured the original (before repair) and repaired (after repair) view, presented side-by-side in the survey. To ensure unbiased responses, we labeled the screenshots simply as version 1 and 2. Additionally, a reference image was provided for each of the layout. It is an ideal layout of the corresponding webpage free from RLFs. We captured all screenshots at the lowest viewport size of the failure range using the browser's inspect tool. Since in a failure viewport range, the lowest viewport size contains the most constrained, worst-case rendering of the layout.

We asked each SE to select (1) Is the layout correct? (2) Is the layout aesthetically pleasing?. We briefed them on the definition of a layout as "correct" if (1) it is positioned correctly (e.g., no visible RLFs, correctly aligned), (2) it is not deviated from the reference layout. Similarly, a layout as "aesthetic" if it is visually appealing (e.g., good spacing and margins).

*2) **Results**:* The user study results, depicted in Table II illustrate the satisfaction ratings for the 20 layouts. For considering a layout as *correct* and *aesthetic*, we counted the maximum vote for both. In case of aesthetic, one layout had same vote for multiple decisions. It was resolved by an additional vote by the first author. Finally, the study revealed that before repair, 8 layouts or 40% were considered as correct based on the maximum vote. After repair, this increased to 17 or 85% layouts as correct. Similarly for aesthetics, before repair there were 9 or 45% aesthetic layouts, which increased to 14 or 70% after repair.

TABLE II: Score Comparisons: Before vs After Repair

Layout Score				Aesthetics Score			
	After Repair				After Repair		
Before Repair	L	L^0	Total	Before Repair	A	A^0	Total
L	6	2	8	A	5	4	9
L^0	11	1	12	A^0	9	2	11
Total	17	3	20	Total	14	6	20

L=Correct Layout | L^0=Not Correct Layout
A=Aesthetic | A^0=Not Aesthetic

For a more rigorous analysis, we employed the Wilcoxon Signed-Rank test [26] separately on the set of total response votes for correctness and aesthetics of each layout and measured the difference level between before and after repair. The test yielded a p-value of approximately $p \approx 0.026 < 0.05$ for correct layout, which indicates a significant difference in before and after repair. Regarding aesthetics, although no statistical significance was found at $p < 0.05$, we observed a slight numerical improvement in the proportion of layouts rated as aesthetic after repair. As opinions on aesthetics are subjective, improvements tend to be smaller, and people's opinions vary more.

However, we investigated the layouts where the repaired version was not preferred by the SEs. In one example, two divs were overlapping on one another, but was not visible. When the overlap was fixed, a small gap appeared between them, which made the original layout and aesthetics better. On the other hand, engineers marked one layout as incorrect both before and after repair, and two layouts as not aesthetic in either version. Such layouts had incorrect placements before repair, which got fixed, but then introduced a different problem. Some of the other votes varied from person to person due to the subjective nature of aesthetic preferences. Overall, our approach demonstrates its quality clearly through both statistical and descriptive analyses.

D. Threats to Validity

Generalizability of webpages used in our study is one of the threats to validity. Hence we selected webpages of varying sizes, as shown in Table I. This approach is consistent with prior studies [5], [7]. Another possible threat comes from LOCALICSS [7]. Since we rely on this for our input, its accuracy directly influences the results. LLMs typically generate non-deterministic results, causing a potential threat to validity. To mitigate this, we ran the model five times and took the most frequently generated patch. Moreover, as LLMs have limitations of max tokens, we restricted our input to the top five SO discussions to prevent overflow of token limits. Finally, subjectivity remains a potential threat when manually validating patches. Hence, we took opinions from five expert SEs to check the effectiveness.

V. CONCLUSION AND FUTURE WORK

In this work, we present ReDeFix, an LLM-based approach using RAG to leverage Stack Overflow (SO) discussions for an effective repair of RLFs. Including the context of an RLF, a prompt is sent to LLM alongside the relevant SO posts. The generated patch is then validated to ensure the RLF is repaired and no new RLFs exist. Evaluations show that it successfully repaired 88% of the existing 43 RLFs. A study on software engineers resulted in preference for 85% of the repaired layouts as correct, while 70% as aesthetic. Overall, these show the effectiveness of ReDeFix in repairing RLFs without distorting the layouts and aesthetics. Future research can be directed to understand layout aesthetics during repair to preserve the visual appeal of a webpage.

REFERENCES

[1] Replication package for this paper. [Online]. Available: https://anonymous.4open.science/r/ReDeFix

[2] LangChain Vectorstores. Online: https://js.langchain.com/docs/concepts/vectorstores/.

[3] Mistral AI. Mistral Small-3.1. https://huggingface.co/mistralai/Mistral-Small-3.1-24B-Base-2503.

[4] Stack Exchange API. https://api.stackexchange.com/.

[5] I. Althomali, G. M. Kapfhammer, and P. McMinn, "Automated repair of responsive web page layouts," in *15th IEEE Conference on Software Testing, Verification and Validation, ICST 2022, Valencia, Spain, April 4-14, 2022*. IEEE, 2022, pp. 140–150. [Online]. Available: https://doi.org/10.1109/ICST53961.2022.00025

[6] T. A. Walsh, P. McMinn, and G. M. Kapfhammer, "Automatic detection of potential layout faults following changes to responsive web pages (N)," in *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*. IEEE Computer Society, 2015, pp. 709–714. [Online]. Available: https://doi.org/10.1109/ASE.2015.31

[7] T. Zerin, B. M. M. Hossain, and K. Sakib, "A heuristic approach to localize css properties for responsive layout failures," in *Proceedings of the 20th International Conference on Evaluation of Novel Approaches to Software Engineering - ENASE*, INSTICC. SciTePress, 2025, pp. 292–303.

[8] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 8, pp. 1–79, 2024.

[9] S. Kabir, D. N. Udo-Imeh, B. Kou, and T. Zhang, "Is stack overflow obsolete? an empirical study of the characteristics of chatgpt answers to stack overflow questions," in *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, 2024, pp. 1–17.

[10] T. A. Walsh, G. M. Kapfhammer, and P. McMinn, "Redecheck: an automatic layout failure checking tool for responsively designed web pages," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, Santa Barbara, CA, USA, July 10 - 14, 2017*. ACM, 2017, pp. 360–363. [Online]. Available: https://doi.org/10.1145/3092703.3098221

[11] ——, "Automated layout failure detection for responsive web pages without an explicit oracle," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, Santa Barbara, CA, USA, July 10 - 14, 2017*. ACM, 2017, pp. 192–202. [Online]. Available: https://doi.org/10.1145/3092703.3092712

[12] X. Yin, C. Ni, S. Wang, Z. Li, L. Zeng, and X. Yang, "Thinkrepair: Self-directed automated program repair," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2024, pp. 1274–1286.

[13] I. Bouzenia, P. Devanbu, and M. Pradel, "Repairagent: An autonomous, llm-based agent for program repair," *arXiv preprint arXiv:2403.17134*, 2024.

[14] J. Kaddour, J. Harris, M. Mozes, H. Bradley, R. Raileanu, and R. McHardy, "Challenges and applications of large language models," *arXiv preprint arXiv:2307.10169*, 2023.

[15] D. Abrahamyan and F. H. Fard, "Stackrag agent: Improving developer answers with retrieval-augmented generation," in *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2024, pp. 893–897.

[16] E. Mansur, J. Chen, M. A. Raza, and M. Wardat, "Ragfix: Enhancing llm code repair using rag and stack overflow posts," in *2024 IEEE International Conference on Big Data (BigData)*. IEEE, 2024, pp. 7491–7496.

[17] M. Mukherjee and V. J. Hellendoorn, "Sosecure: Safer code generation with rag and stackoverflow discussions," *arXiv preprint arXiv:2503.13654*, 2025.

[18] X. Liu and H. Zhong, "Mining stackoverflow for program repair," in *2018 IEEE 25th international conference on software analysis, evolution and reengineering (SANER)*. IEEE, 2018, pp. 118–129.

[19] S. Rose, D. Engel, N. Cramer, and W. Cowley, "Automatic keyword extraction from individual documents," *Text mining: applications and theory*, pp. 1–20, 2010.

[20] H.-L. Hsu and J. Tzeng, "Dat: Dynamic alpha tuning for hybrid retrieval in retrieval-augmented generation," *arXiv preprint arXiv:2503.23013*, 2025.

[21] S. Robertson, H. Zaragoza *et al.*, "The probabilistic relevance framework: Bm25 and beyond," *Foundations and Trends® in Information Retrieval*, vol. 3, no. 4, pp. 333–389, 2009.

[22] M. M. Rahman, S. Chakraborty, G. Kaiser, and B. Ray, "Toward optimal selection of information retrieval models for software engineering tasks," in *2019 19th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2019, pp. 127–138.

[23] A. Kong, S. Zhao, H. Chen, Q. Li, Y. Qin, R. Sun, X. Zhou, E. Wang, and X. Dong, "Better zero-shot reasoning with role-play prompting," *arXiv preprint arXiv:2308.07702*, 2023.

[24] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[25] O. Topsakal and T. C. Akinci, "Creating large language model applications utilizing langchain: A primer on developing llm apps fast," in *International Conference on Applied Engineering and Natural Sciences*, vol. 1, no. 1, 2023, pp. 1050–1056.

[26] R. F. Woolson, "Wilcoxon signed-rank test," *Encyclopedia of biostatistics*, vol. 8, 2005.