Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 246 (2024) 1346–1354

Procedia Computer Science

www.elsevier.com/locate/procedia

28th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES 2024)

Automating Software Documentation: Employing LLMs for Precise Use Case Description

Lahbib Naimi[a]*, El Mahi Bouziane[a], Abdeslam Jakimi[a], Rachid Saadane[b] and Abdellah Chehri[c]

[a]*GL-ISI Team, Faculty of Science and Technology of Errachidia, UMI-Meknes, Morocco.*
[b]*Hassania School of Public Works, Casablanca, Morocco*
[c]*Department of Mathematics and Computer Science, Royal Military College of Canada, Kingston, Canada*

Abstract

The creation of software documentation is widely recognized as a critical and demanding undertaking within the rapidly changing realm of software development. This study introduces a novel method for generating software documentation by leveraging Large Language Models (LLM). The paper presents a novel system that extracts use cases from UML Use Case Diagrams and employs a Generative AI Model to generate descriptive text for each extracted use case. This approach aims to reduce the amount of time dedicated to documentation and encourage uniformity in the description of software functions. The results suggest that the level of manual labor and time needed can be substantially decreased by upholding elevated levels of clarity and comprehensiveness in software documentation. This study presents a use-case scenario that showcases the practical application of our methodology in real-world situations. The purpose of this example is to demonstrate the practicality and effectiveness of the method.

Keywords: LLMs; software documentation; prompt engineering; generative AI; user documentation;

1. Introduction

In the current software engineering context, solid documentation is essential. It guides development, supports maintenance, and ensures the longevity of software products. However, the development and maintenance of such documentation are often tedious and error-prone tasks. Recently, advancements in generative artificial intelligence have created a new opportunity to address this problem [1].

* Corresponding author. Abdellah Chehri, Email: chehri@rmc.ca; Lahbib Naimi E-mail: l.naimi@edu.umi.ac.ma

This paper presents a novel approach that leverages generative AI for the automated creation of software documentation, specifically, the automated interpretation of use case diagrams. Our novel method involves extracting use cases from the XML representations of these diagrams and feeding this structured data into a Generative AI model, which transforms the structured use cases into detailed, easily readable documentation.

An important aspect of our research is prompt engineering [2]. The ability to create precise prompts for the Large Language Model (LLM) [3] allows for the generation of accurate, relevant, and suitable documentation tailored to the requirements of the software project. As researchers, we iteratively refine these prompts, guiding the AI to produce informative and digestible documentation.

Our approach offers a host of benefits: it significantly reduces the time and resources required for documentation, while enhancing its quality and consistency. It also fosters a more dynamic, agile documentation process that can flexibly adapt to the evolving needs of software development.

In the following sections, we delve into the details of the approach, explain the system's workings, and then show its effectiveness through the case study. We begin in Section 1 with the background discussion: Software Documentation, LLMs, Generative AI, and Prompt Engineering. In Section 2, we review related works, followed by the presentation of our methodology in Section 3. In Section 4, we will demonstrate the application of our approach through a case study involving two smart tourism use cases. Finally, a discussion on the benefits and limitations of our approach will be presented in the discussion and conclusion section.

2. Background

In this section, we discuss important software documentation concepts and deep-dive into the power of Generative AI and prompt engineering techniques. With all these elements combined, we foresee changes in how we do software documentation to make it efficient, accurate, and adaptable.

2.1. Software Documentation

Software documentation [4] is not just a tool, but a crucial link that connects developers, end users, and maintainers. It serves as a beacon, providing essential information on the software systems, guiding their creation, understanding, and maintenance [5].

Internal documentation [6] is more than just a communication tool in the software development process. It's a lifeline, helping to maintain clear guidelines and responsibilities, fostering efficient collaboration among the development teams.

Comprehensive documentation is a game-changer for external stakeholders, such as user experience professionals and end users. User documentation is not just a guide, but a key to understanding the software's functionality and troubleshooting procedures. It empowers users to navigate a developed software product, enhancing overall user satisfaction [7].

Software documentation can be in various types, with each type serving a specific purpose for different phases during software development:

- *Process Documentation:* This documentation provides a roadmap for development teams in the software development and maintenance process, including the workflow, responsibilities, and procedures of the work. It helps the people concerned understand things more clearly and provides the project team with a guideline for smooth cooperation and efficient progress over the project's whole lifecycle [8].
- *Estimate Documentation:* Provides insight into the resource plan through estimates of time and cost for project tasks. This will help determine budgets, team selection, and pricing, especially in cost-effective project management and resource investment [9].
- *User Documentation:* User documentation is created for end-users. It provides guidelines on how to use and implement the software to meet their requirements. This includes user manuals, help guides, and FAQs. The user gains knowledge about the product for productive interaction, error detection, and recovery [10].

• *Technical Documentation:* This document contains detailed information for developers, administrators, and maintainers. It includes system architecture, APIs, data models, and a description of the system's deployment. This documentation will provide insight and ease in managing the software system [11, 12].

• *System Documentation:* System documentation provides stakeholders with the complete software system design and its components and how these components interact. Through diagrams, flowcharts, and detailed narrations, system documentation will provide full insight into the structure of the system and the operation of the decision-making and system management system.

2.2. Generative AI and Large Language Models

Generative AI is one of the most powerful forces in the vast world of artificial intelligence. It enables computers to generate content across different domains independently [13]. Whether it be the generation of images, text, code, or synthetic data, the generative AI model can produce new and relevant output. These models are trained to learn the patterns of the existing data and then to use these learned patterns to generate new instances based on certain contexts and requirements.

LLMs, the cornerstone of generative AI, are the key to its applications. These models, pre-trained on massive amounts of textual data, delve into the intricacies of syntax and semantics that form human languages. Using the transformer architecture [14], the models excel at processing sequential data, making them adept at understanding and generating natural language. Through their training, LLMs develop a sophisticated ability to predict the next word in a sentence based on the preceding context, thus gaining a deep understanding of language nuances and subtleties [15]. This robust understanding enables such models to generate contextually appropriate and coherent text, paving the way for new advancements in natural language processing and generation.

2.3. Prompt Engineering

Prompt engineering, a crucial process in the realm of AI, involves refining prompts and input for generative, artificial AI services to produce text or images [16]. This process, which transforms raw queries into meaningful AI-generated text, is instrumental in achieving the desired output. Whether it's about making generative models like ChatGPT [17] or DALL-E [18] more effective, or refining LLMs with specific prompts, prompt engineering is the key to success.

Understanding the various types of prompts [19] is essential as they play pivotal roles in guiding AI models. These prompts offer a spectrum of approaches, from simple instruction to iterative refinement, and are instrumental in steering AI models toward accurate and contextually relevant outputs.

• *Direct Prompting (Zero-shot):* Zero-shot direct prompting is instructional prompting in which no examples are provided [20].

• *Role Prompting (a variant):* It indicates a specific role for a model that helps understand the goals and objectives. It is a type of prompting that actually prompts the model to take on a specific role and, therefore, assists its response in that role type [21].

• *Prompting with Examples (One-, Few-, and Multi-shot):* This prompting type provides examples to the model, which helps the model understand better and focus. These examples range from one-shot to multi-shot, which assist in yielding accurate results by providing content and context [22].

• *Contextual Prompts:* Contextual prompts provide extra background information, especially with complex tasks or instructions in multiple steps. This aids the model by providing even more context in responding appropriately [23].

• *Bias Mitigation:* It guides the avoidance of biases and hence prevents inappropriate or offensive outputs. By guiding the model to provide fair and neutral responses, it will lead to consistent and accurate services [24].

• *Fine-tuning and Interactive Prompts:* This allows for an iterative improvement in AI responses. The user can evaluate and check responses provided by the model before the final response generation so that desired outcomes are obtained through cooperative interaction [25].

3. Related Work

This section presents very varied insights into software documentation and automation, including perspectives from practitioners, advances in AI-powered code documentation, consideration within policy frameworks, and the intricacies of documentation practices in healthcare. Each of the works adds to our comprehension of the challenges and opportunities in software documentation in a unique fashion, offering great insights to improve efficiency and relevance in documentation processes.

This paper [9] reports a study with 146 practitioners, surveying prevalent problems with documentation and identifying the types of documentation crucial for tasks. The results from the practitioners are not just informative, but also serve as a practical guide for the design of advanced recommender systems. These systems are designed to alleviate documentation problems, enhancing the quality and relevance of automated documentation, and making the research findings directly applicable in real-world scenarios.

This paper [26] explores the potential of automatic code documentation using Codex, a GPT-3 based model. The research demonstrates that Codex can be effective in six different programming languages, showcasing the exciting possibility of automating the process within software development. GPT-3 opens up a world of opportunities for better quality and faster speed of coding within projects, offering the potential for efficient and accurate software documentation that was previously unimaginable.

This research [27] concentrates on data and AI documentation within the European policy framework as a basis for documentation practices harmonized with legal requirements in European data processing organizations. Less emphasis is placed on technical details; instead, the research emphasizes the need for policy-based context-aware documentation methods that can help us understand some key dimensions surrounding documentation as a whole, as brought out through this study.

This paper [28] critically analyzes general practitioners' documentation practices based on ethnographic research in healthcare settings. It goes beyond the technical aspects to examine the societal complexities that come with documenting. Taking an approach that is not centered on automated functions, this research reveals different ways of viewing medical records, hence stressing the importance of considering human as well as technical issues with regard to documents.

4. Methodology

This section describes the systematic process adapted to implement automation in use case extraction and documentation from UML diagrams.

The process, represented in Figure 1, starts with the entry of an XML file, which contains structured data of usually a use case class diagram. Due to its flexibility, XML can capture very complex data structures.

The next step is parsing the XML file into the system to extract the needed data for the use cases. This involves identifying and isolating some elements of the use case diagram that carry valid information on the use cases.

Once the use case data is extracted, a crucial step is the creation of a prompt. This prompt serves as a guide for the Generative AI, instructing it on the specific outputs to generate. In our case, the prompt is designed to produce text for software documentation, ensuring the AI's output aligns with our requirements.

This prompt is then transmitted to the Generative AI as an API request. The Gemini API [29], one of the available interfaces, serves as the conduit, connecting the user's requirements to the AI engine. This request triggers the AI model to generate the text as per the prompt, which will form the content of the software documentation.

The last step is saving the documentation text or explanation generated to an output file. This will give structured output that clearly explains the use cases derived from the diagram.

Moreover, the methodology will be further explained through a concrete use case detailed in the following section. This use case will provide a clear example of how the methodology presented in the following section is instantiated into a real-world problem and how it gains insights into its validity.

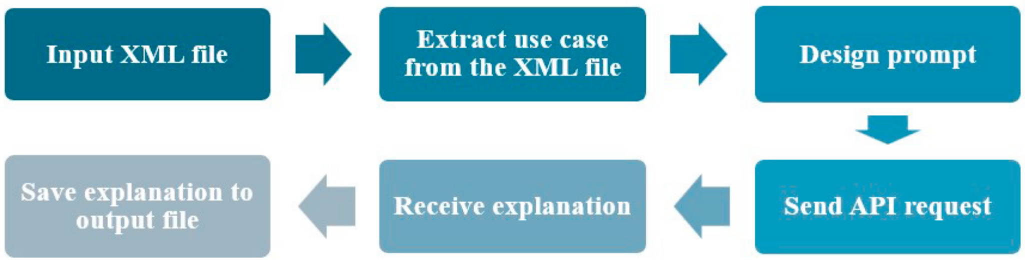

Input XML file
Extract use case from the XML file
Design prompt
Send API request
Receive explanation
Save explanation to output file

Fig. 1. Documentation generation process**.**

5. Case Study

In this section, we consider a more detailed examination of two use cases concerning smart tourism applications: "Share an Attraction" and "Search for a Nearby Restaurant." These two use cases significantly contribute to the user experiences that smart tourism ecosystems offer through interaction with local attractions and eating places. We use Visual Paradigm [30] to create the corresponding use case diagram and export it as an XML file. In this case study, we use PHP functions to extract the necessary use case information from the XML file. The extracted information is used to design prompts relevant to providing context to the LLM in generating use case explanations specific to smart tourism applications. We then send these prompts to the Gemini API in order to obtain more detailed explanations that will, in turn, enrich and increase user engagement in smart tourism. The application interface is depicted in Figure 2.

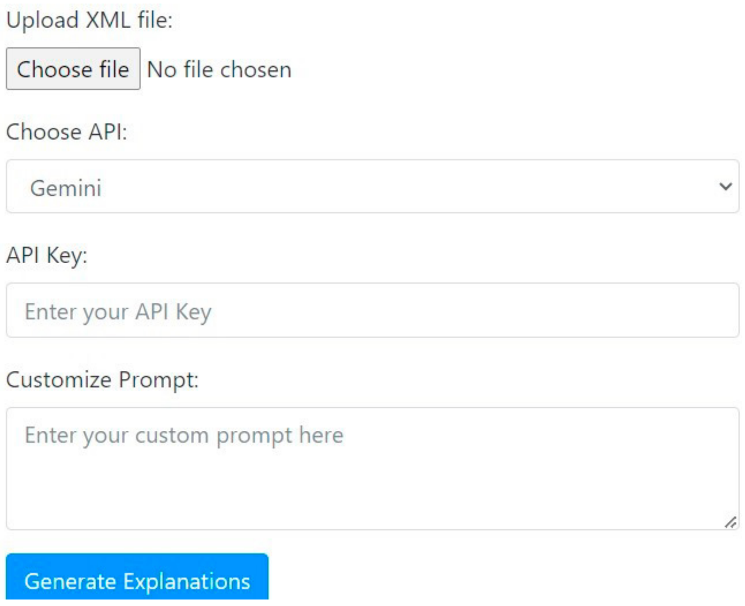


Fig. 2. System user interface.

Furthermore, users can choose their favorite LLM API, for example, ChatGPT API or Gemini API, and then fill in the API key that should be used for authentication. Of course, it is noted that sometimes the keys for APIs will become invalid, which means the user needs to update the key they filled in. Also, users can compose prompts that would describe the diagram and the given code generation context, along with the components of the diagram, to further specialize the generated explanations according to the requirements.

5.1. Use Case Diagram Design

Figure 3 illustrates the use case diagram developed using a Visual Paradigm to represent user interaction and the smart tourism application. Notable use cases include 'Share an Attraction' and 'Search for a Nearby Restaurant', which detail important functionalities of the application.

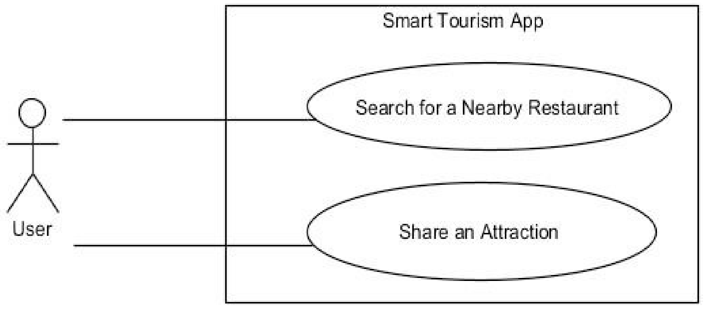
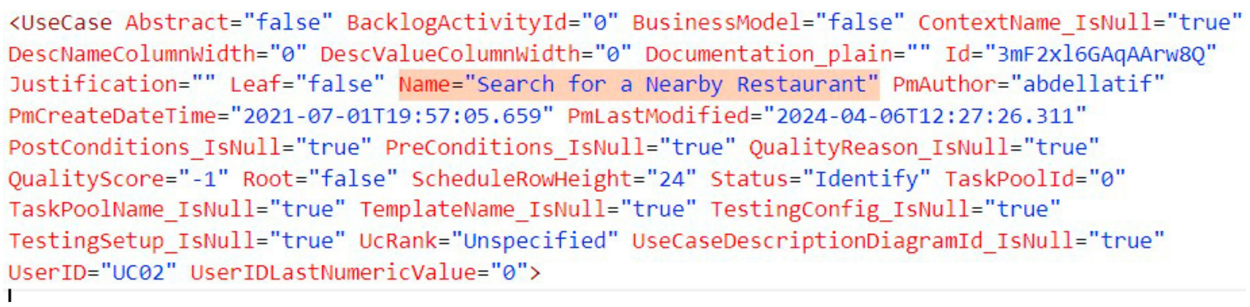


Fig. 3. Simplified use case diagram.

5.2. XML Data Extraction

Figure 4 is the XML code exported from the use case diagram, with most of the structured data that is required for further processing encompassed in the XML structure. The <UseCase> element carries important information in the XML structure about every separate use case. More specifically, within the <UseCase> element, the names of the use cases are contained in an attribute "Name". The parsing of this attribute will allow us to detect and extract the names of the use cases, thus supporting the following steps of our methodology.

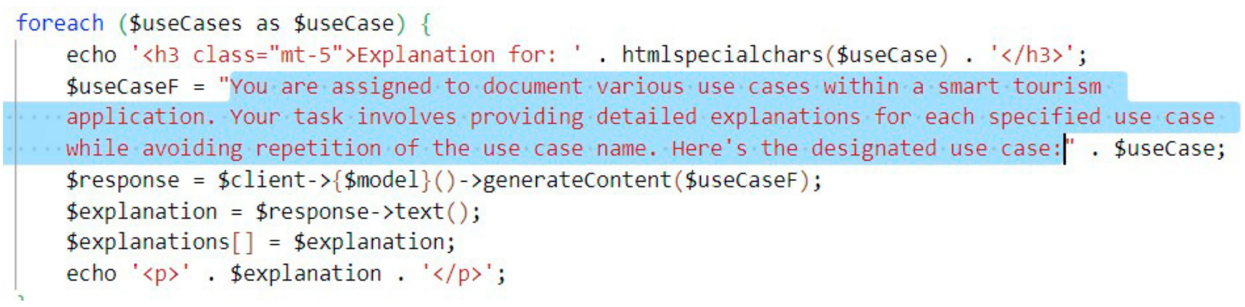


```
<UseCase Abstract="false" BacklogActivityId="0" BusinessModel="false" ContextName_IsNull="true"
DescNameColumnWidth="0" DescValueColumnWidth="0" Documentation_plain="" Id="3mF2xl6GAqAArw8Q"
Justification="" Leaf="false" Name="Search for a Nearby Restaurant" PmAuthor="abdellatif"
PmCreateDateTime="2021-07-01T19:57:05.659" PmLastModified="2024-04-06T12:27:26.311"
PostConditions_IsNull="true" PreConditions_IsNull="true" QualityReason_IsNull="true"
QualityScore="-1" Root="false" ScheduleRowHeight="24" Status="Identify" TaskPoolId="0"
TaskPoolName_IsNull="true" TemplateName_IsNull="true" TestingConfig_IsNull="true"
TestingSetup_IsNull="true" UcRank="Unspecified" UseCaseDescriptionDiagramId_IsNull="true"
UserID="UC02" UserIDLastNumericValue="0">
```

Fig. 4. XML code of the exported use case.

5.3. Prompt Design for LLM

In this step, prompts are designed to give context to the LLM for generating use case explanations specific to smart tourism applications. These prompts put forth specific requirements and objectives that are associated with the use case, guiding the model to answer relevant and accurate explanations. The PHP code and the corresponding prompt sent to the model are illustrated in Figure 5.



```php
foreach ($useCases as $useCase) {
    echo '<h3 class="mt-5">Explanation for: ' . htmlspecialchars($useCase) . '</h3>';
    $useCaseF = "You are assigned to document various use cases within a smart tourism
    application. Your task involves providing detailed explanations for each specified use case
    while avoiding repetition of the use case name. Here's the designated use case:" . $useCase;
    $response = $client->{$model}()->generateContent($useCaseF);
    $explanation = $response->text();
    $explanations[] = $explanation;
    echo '<p>' . $explanation . '</p>';
}
```

Fig. 5. Prompt sent to the LLM.

5.5. Request to Gemini API

These designed prompts are then sent as requests to the Gemini API, using its advanced capabilities to create allen compassing use case explanations. This pivotal step is important since it effectively bridges user requirements with the capabilities of AI to enable the creation of informative and contextually relevant content

6. Discussion and Conclusion

This paper introduced a new approach to automated software documentation generation, harnessing the power of LLMs. When compared to conventional methods like MDA [31], this approach offers significant advantages. First, the quality of the generated documentation is remarkably human-like, providing clear and relevant explanations that closely resemble expertly written manuals. This represents a substantial leap forward from MDA, where frequently produced documents lack the nuances and clear explanations a human writer provides [32, 33].

One other advantage of LLMs is that they speed up the documentation process. LLMs can document the work on time, which is very much needed in the development environment these days, which is characterized by fast software development cycles. This does not come at the cost of quality, as these models can churn out high-quality documentation in a fraction of the time their traditional counterparts need.

However, it's important to note that this approach does have its limitations. The use of LLMs is generally limited to a certain number of tokens that can be used for free. Every token used beyond the free limit incurs a cost, which can become significant as the number of tokens increases. This can pose a challenge for projects with extensive documentation requirements.

Furthermore, those models need a lot of fine-grained context to generate proper documentation. The more such context is provided, the better the output generally will be. That requires understanding the domain and effective communication with the language model.

While this paper focused on use-case explanations, there is enormous potential in applying LLMs to documentation. This technique can be extended in future work to include a wider variety of documentation types, such as user guides, technical specifications, and more. Again, the process can be optimized in terms of extracting more detailed context from the XML files to yield better-quality documentation.

In conclusion, integrating LLMs into the documentation generation process should revolutionize software development. It will enable quality, speed, and adaptability close to human ones, which traditional methods can't meet. As we investigate further and tune this method, it promises to improve software documentation.

Acknowledgements

The National Center for Scientific and Technical Research (CNRST) provided funding for this research as part of the Moroccan project titled "Al-Khawarizmi program in AI and its Applications".

References

[1] M. Jovanovic and M. Campbell, "Generative artificial intelligence: Trends and prospects," *Computer,* vol. 55, no. 10, pp. 107-112, 2022.

[2] L. Giray, "Prompt engineering with ChatGPT: a guide for academic writers," *Annals of biomedical engineering,* vol. 51, no. 12, pp. 26292633, 2023.

[3] H. Naveed *et al.*, "A comprehensive overview of large language models," *arXiv preprint arXiv:2307.06435,* 2023.

[4] A. Forward and T. C. Lethbridge, "The relevance of software documentation, tools and technologies: a survey," 2002, pp. 26-33.

[5] N. J. Kipyegen and W. P. K. Korir, "Importance of software documentation," *International Journal of Computer Science Issues (IJCSI),* vol. 10, no. 5, p. 223, 2013.

[6] C. J. Stettina and W. Heijstek, "Necessary and neglected? An empirical study of internal documentation in agile software development teams," 2011, pp. 159-166.

[7] E. Aghajani *et al.*, "Software documentation issues unveiled," 2019: IEEE, pp. 1199-1210.

[8] T. C. Lethbridge, J. Singer, and A. Forward, "How software engineers use documentation: The state of the practice," *IEEE software,* vol. 20, no. 6, pp. 35-39, 2003.

[9] E. Aghajani *et al.*, "Software documentation: the practitioners' perspective," 2020, pp. 590-601. [10] D. Team, "What is Software Documentation? Its types and Best practices," 2020-11-10 2020.

[11] G. Garousi, V. Garousi-Yusifoğlu, G. Ruhe, J. Zhi, M. Moussavi, and B. Smith, "Usage and usefulness of technical software documentation: An industrial case study," *Information and software technology,* vol. 57, pp. 664-682, 2015.

[12] G. Garousi, V. Garousi, M. Moussavi, G. Ruhe, and B. Smith, "Evaluating usage and quality of technical software documentation: an empirical study," 2013, pp. 24-35.

[13] A. Bandi, P. V. S. R. Adapa, and Y. E. V. P. K. Kuchi, "The power of generative ai: A review of requirements, models, input–output formats, evaluation metrics, and challenges," *Future Internet,* vol. 15, no. 8, p. 260, 2023.

[14] K. Han, A. Xiao, E. Wu, J. Guo, C. Xu, and Y. Wang, "Transformer in transformer," *Advances in neural information processing systems,* vol. 34, pp. 15908-15919, 2021.

[15] W. X. Zhao *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223,* 2023.

[16] G. Marvin, N. Hellen, D. Jjingo, and J. Nakatumba-Nabende, "Prompt Engineering in Large Language Models," 2023: Springer, pp. 387402.

[17] OpenAI. "*ChatGPT* (Mar 14 version) [Large language model]." https://chat.openai.com/chat (accessed 2024).

[18] G. Marcus, E. Davis, and S. Aaronson, "A very preliminary analysis of DALL-E 2," *arXiv preprint arXiv:2204.13807,* 2022.

[19] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, and A. Chadha, "A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications," *arXiv preprint arXiv:2402.07927,* 2024.

[20] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *Advances in neural information processing systems,* vol. 35, pp. 22199-22213, 2022.

[21] R. Wang *et al.*, "Role Prompting Guided Domain Adaptation with General Capability Preserve for Large Language Models," *arXiv preprint arXiv:2403.02756,* 2024.

[22] G. Yong, K. Jeon, D. Gil, and G. Lee, "Prompt engineering for zero‐shot and few‐shot defect detection and classification using a visuallanguage pretrained model," *Computer‐Aided Civil and Infrastructure Engineering,* vol. 38, no. 11, pp. 1536-1554, 2023.

[23] K. Vasisht, B. Ganesan, V. Kumar, and V. Bhatnagar, "Infusing Knowledge into Large Language Models with Contextual Prompts," *arXiv preprint arXiv:2403.01481,* 2024.

[24] Z. Xu, K. Peng, L. Ding, D. Tao, and X. Lu, "Take Care of Your Prompt Bias! Investigating and Mitigating Prompt Bias in Factual Knowledge Extraction," *arXiv preprint arXiv:2403.09963,* 2024.

[25] J. Shin, C. Tang, T. Mohati, M. Nayebi, S. Wang, and H. Hemmati, "Prompt Engineering or Fine Tuning: An Empirical Assessment of Large Language Models in Automated Software Engineering Tasks," *arXiv preprint arXiv:2310.10508,* 2023.

[26] J. Y. Khan and G. Uddin, "Automatic code documentation generation using gpt-3," 2022, pp. 1-6.

[27] M. Micheli, I. Hupont, B. Delipetrev, and J. Soler-Garrido, "The landscape of data and AI documentation approaches in the European policy context," *Ethics and Information Technology,* vol. 25, no. 4, p. 56, 2023/10/28 2023, doi: 10.1007/s10676-023-09725-7.

[28] M. Willis and M. H. Jarrahi, "Automating documentation: a critical perspective into the role of artificial intelligence in clinical documentation," 2019: Springer, pp. 200-209.

[29] G. AI. "Gemini Large Language Model." https://gemini.google.com/ (accessed 2024).

[30] V. Paradigm, *Visual Paradigm User's Guide*. 2023.

[31] R. Soley, "Model driven architecture," *OMG white paper,* vol. 308, no. 308, p. 5, 2000.

[32] C. Wang, H. Li, Z. Gao, M. Yao, and Y. Yang, "An automatic documentation generator based on model-driven techniques," 2010, vol. 4: IEEE, pp. V4-175.

[33] L. Naimi, H. Abdelmalek, and A. Jakimi, "A DSL-based Approach for Code Generation and Navigation Process Management in a Single Page Application," *Procedia Computer Science,* vol. 231, pp. 299-304, 2024.