

Directing generative AI for Pharo Documentation

Pascal Zaragoza^{1,*}, Nicolas Hlad¹ and Mohamed Ilyes Amara¹

¹Berger-Levrault, 64 Rue Jean Rostand, 31670 Labège

Abstract

Inadequate code documentation can inhibit this activity and increases the time a developer spends on a task. However, for multiple reasons, good documentation is often lacking and rarely updated. This paper investigates how generative AI, specifically Large Language Models (LLMs), can enhance the quality of package-level documentation in the Pharo Smalltalk environment. Despite Pharo's support for embedded documentation and CRC (Class-Responsibility-Collaborator) methodology, proper package-level comments remain rare and often insufficient. To address this, we introduce three retrieval-augmented strategies for automatic package comment generation: (1) source-based extraction using class source code, (2) comment-based extraction leveraging existing class-level comments, and (3) a hybrid approach combining class comments with inter-class dependency data. We conduct an empirical evaluation across 21 packages of varying sizes using a structured Likert-scale questionnaire centered on CRC criteria, responsibility clarity, collaborator relevance, and comparison to existing comments. Results indicate that while no single strategy significantly outperforms others. However, all methods generate comments perceived as more useful, complete, and clear than the originals. Moreover, this study highlights the limitations in reliably identifying collaborators, especially in larger packages.

Keywords

generative AI, documentation, comment generation, Tool

1. Introduction

Writing and reading documentation are an integral part of the development cycle. In fact, $\sim 58\%$ of a developer's time is used in code comprehension-related activities [1]. Inadequate code documentation can inhibit this activity and increases the time a developer spends on a task. Therefore, good documentation is essential to facilitate this time-consuming task, so that developers can focus on contributing to the codebase. However, for multiple reasons, good documentation is often lacking and rarely updated.

Pharo¹ is a SmallTalk environment that allows several insertions of code documentation directly inside its environment at the package, class, and method level. Furthermore, Pharo recommends developers to describe classes and packages using *Class-Responsibility-collaborator* (CRC) cards. Introduced by [2], the CRC card method proposes to describe the class name, its responsibilities, and its collaborators (other classes with which it interacts). Despite this recommendation, less than 6% of packages in Pharo apply proper CRC - from the official distribution of Pharo 12.0 - 64bit image:

1. 128/768 (16,7%) packages have a comment,
2. 10 196/12 568 (81,1%) classes have a comment, and
3. 115 050/274 282 (41,9%) methods contain a comment.

Furthermore, package comments tend to be small with 76/128 (60,3%) comments are less than 100 characters (see fig. 1). Package comments of these sizes provide few information that can be used for code comprehension. An example package comment is given in fig. 2 (a).

In this work, we study how we can increase the quality of generated package comments using Large Language Models (LLMs). Particularly, we are interested in studying the impact of the source of information during the retrieval process of retrieval-augmented generation has on the generation of

IWST 2025: International Workshop on Smalltalk Technologies, July 1-4, 2025, Gdansk, Poland

*Corresponding author.

✉ pascal.zaragoza@berger-levrault.com (P. Zaragoza); nicolas.hlad@berger-levrault.com (N. Hlad); mohamedilyesamara@gmail.com (M.I. Amara)

ORCID 0000-0002-5033-6392 (P. Zaragoza); 0000-0003-4989-2508 (N. Hlad)



© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

¹<https://pharo.org/>

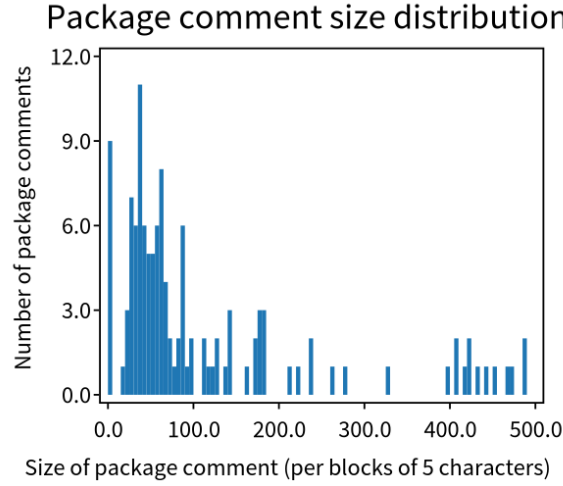


Figure 1: Distribution of package comments based on their size in terms of number of characters (outlier comments bigger than 500 were excluded for visualization of the distribution).

the package comment. This source of information can range from the source text of the classes to user descriptions and inter-package dependencies. With this in mind, we have implemented three different strategies to generate a package comment using its underlying classes, their comments (when available), and their dependencies to study the underlying effect on the source of information.

In the following section, we present the related works. Next, we present the general approach towards applying retrieval-augmented strategies, as well as the three proposed strategies. Next, we present an experimentation with the goal of exploring the impact of retrieval-augmented generation around package comment generation. Finally, we conclude and offer insights on the results of the experimentation.

2. Related works

Generating documentation for developers is not a recent endeavour with [3] exploring comment generation. In this work the authors, propose a method to automatically generate comments for Java classes using heuristics and stereotypes. The generated summaries are indicative and abstract, aiming to provide a brief, informative description of the class content. The summaries are constructed based on heuristics, such as focusing on accessor methods for Data Provider classes while excluding other methods which follows a similar principle as the CRC methodology when it comes to the collaborators.

Another study [4] explores the use of large language models (LLMs) for the task of generating project-specific code summaries, using the few-shot learning technique. The principle behind this technique is to provide the LLM, via its prompt, with examples of input/output pairs (usually from other projects) that are similar to the input we want to process and the expected output.

More recently,[5] proposes a new method called Automatic Semantic Augmentation of Prompts (ASAP), which aims to build prompts for software engineering tasks. The hypothesis underlying this research is that an effective prompt is one that provides the LLM with everything a developer would take into consideration as semantic and syntactic facts when manually executing the task. This study deals with the case of code comment generation (code summaries). ASAP adds semantic facts extracted from an analysis of the source code to the prompt. In addition, it uses the few-shot technique. The ASAP approach improves the average performance of LLMs for several commonly used metrics, including BLEU (a semantic based-metric), by 1.68% to 18.69%.

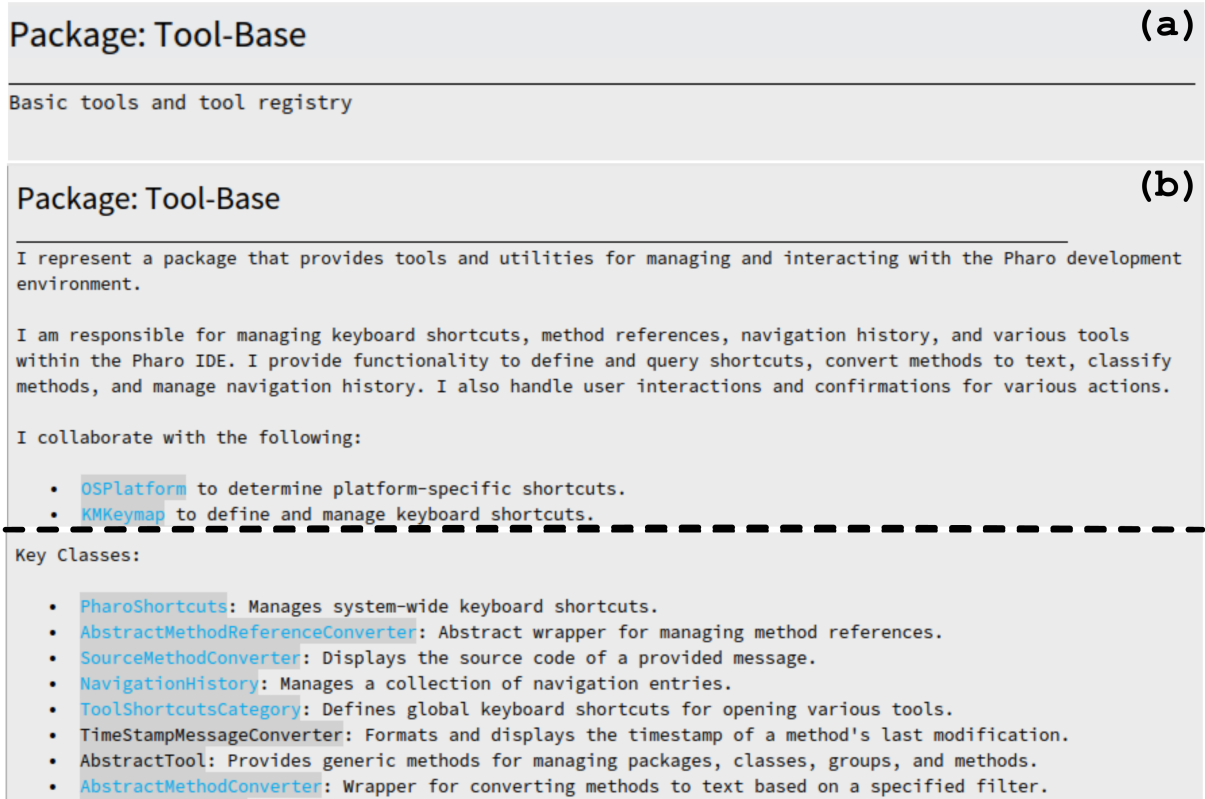


Figure 2: Difference between a (a) package comment reference and (b) a generated package comment (we have abbreviated the generated comment for spacing reasons).

3. Our comment generation solution

To generate a package comment, we have implemented three different retrieval-augmented strategies towards generating packages comments. Each approach relies on a general three-step plan: (1) extract a model representation of the package (2) extract (or retrieve) the relevant data (3) generate a package comment. Depending on the strategy, we extract different types of data from the model. In fig. 3, we illustrate 3 different extraction strategies that rely on the model representation.

Strategy 1: Naive extraction: For the first strategy, we apply a naive strategy where for each class within a package, we extract the source text (or .st file). From this extraction, we generate a summary with instructions to describe the class' responsibilities, collaborators and key implementations. Then, we instruct an LLM to create a package comment following the CRC card methodology based on the sum of the class summaries. This strategy relies on the LLM to understand the Smalltalk-based languages to extract quality class summaries. However, there are risks that including all the source text will increase the context and therefore increase the likelihood of hallucinations [6]. However, it is a more compute-expensive strategy as there is a multi-step generation process. An example generation is given in fig. 2, and compared with its reference package comment. This example highlights the potential strength of package comment generation. In this case, we are able to retrieve a succinct description of the responsibility of the package, its collaborators, and key classes within the package and their responsibilities. Indeed, with this strategy, the LLM is able to recognize that `OSPlatform` and that it is used by `PharoShortcuts` to manage platform specific shortcuts. However, this strategy tends to create hallucinations which can be seen with the mention of non-existent classes within the package such as `TimeStampMessageConverter`, which should be `TimeStampMethodConverter`.

Strategy 2: Comment-based extraction: For the second strategy, we apply a strategy that relies on the existing class comments. In this context, we extract a list of existing class comment from the package entity. Then, we instruct an LLM to create a package comment following the CRC card methodology

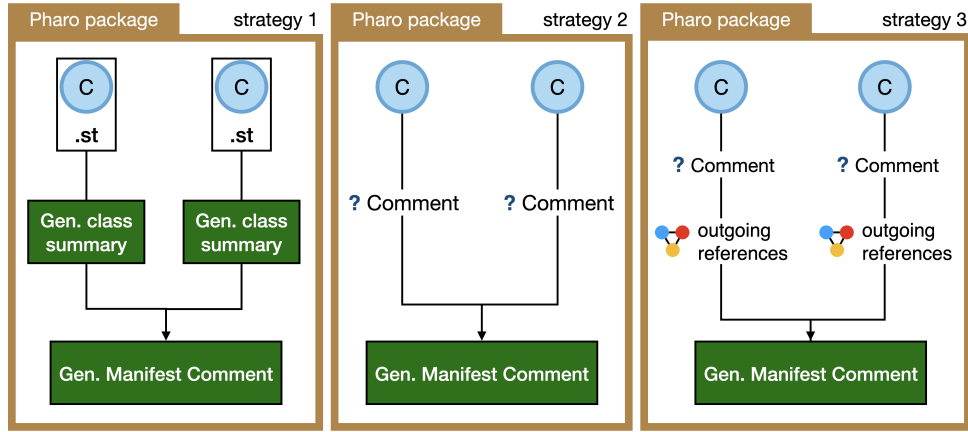


Figure 3: An illustration of the three different approaches we propose to generate package comments.

from the sum of the existing class comments. This strategy relies on the high rate of class comments to work. In the case where a package has a low rate of class comments, this strategy can generate poor quality package comments. Furthermore, it removes the ability of the LLM to extract information from the classes which would not be included in a class comment such as dependencies.

Strategy 3: Comment & dependency-based extraction For the third strategy, we apply a strategy that relies on the entity relationship between the classes. Just like with strategy 2, we extract the existing comments. Furthermore, we extract the outgoing references from the class methods to identify the potential collaborators. Then, we instruct an LLM to create a package comment following the CRC card methodology from the sum of the existing class comments along with the outgoing references in its context. This strategy has the upside of potentially exploiting user-written class comments, while also having an understanding of the internal and external dependencies of the package. Similarly to Strategy 2, in the case where a package has a low rate of class comments, this strategy can generate poor quality package comments.

The implementation of these strategies are made in Pharo and can be found on Github².

4. Experimentation

We propose a set of package comment generation strategies which use different sources of information to highlight the strength and weaknesses of each source. By exploring different sources, we wish to highlight their advantages and inconveniences. To highlight this, we establish a set of research questions:

- **RQ 1:** Does the strategy have an impact on the overall CRC structure?
- **RQ 2:** Does the strategy have an impact on the description of the responsibility of the package?
- **RQ 3:** Does the strategy have an impact on the description of the collaborators of the package?
- **RQ 4:** Does the strategy have an impact on the overall quality across the CRC structure, the responsibility and collaborator description of the generated comment when it is compared to the source comment?

Furthermore, the longer the context provided to an LLM becomes, the more likely it is that the LLM will improperly retrieve relevant information for a task [6]. This raises another question:

- **RQ 5:** Does the size of a package impact the quality of the generated comment?

²<https://github.com/pzaragoza93/AutoCodeDocumentator>

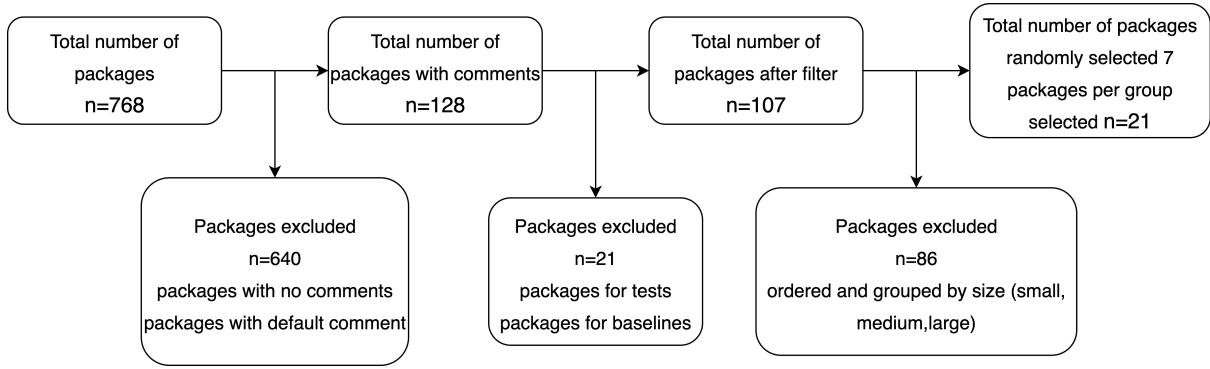


Figure 4: Dataset selection process for the Pharo packages.

4.1. Evaluation Dataset

To study the impact of package comment generation, we applied the following package selection process to create a dataset of packages with existing comments to serve as a reference (see fig. 4). Initially, we extract all Pharo 12 packages (768) and filter them to only include packages with comments which can serve as a reference (128). Then, we exclude test and baseline packages since we deem them less interesting for comment generation (107).

To study the impact of the different strategies, as well as the context size on the quality of package comment generation, we order the remaining packaged based on their size (nb of classes). We then, create three distinctive groups of packages by splitting the ordered list into three groups (small, medium, and large). Finally, from each group we randomly select 7 packages for a total of 21 packages. From this dataset of 21 packages, we apply the three different strategies to generate 63 package comments. To generate the set of package comments, we used 'mistral-small-2503', an open-source LLM model available on huggingface³.

4.2. Evaluation Metrics

To evaluate the quality of the generated package comments, we propose a manual evaluation of the 63 generated comments based 4 categories : (1) the overall CRC methodology structure, (2) the description of the package's responsibility (3) description of the package's collaborators, and (4) overall comparison with the original comment as a reference.

With this in mind, we propose a set of 12 statements divided into 4 different categories corresponding to the categories mentioned above (see table 1). For each statement, the user is asked to evaluate the veracity of a statement on a Likert scale of 7 points. The more they agree with a statement the higher the score. Inversely, the more they disagree with a statement the lower the score. For each package, we then ask these statements for each comment generated by the three comment-generation strategies. This means 36 statements are asked per package. Finally, we divide the 21 packages into three groups, so that each group of participants only evaluates 7 packages (and their 3 generated comments) for a total of 21 generated comments, or 252 statements.

For this experiment, we separated 6 Pharo users into three different groups and evaluate the 21 generated comments over 7 packages. To better illustrate the overall evaluation, we propose fig. 5.

5. Results

The results of the manual evaluation, the scripts for the analysis, and the evaluation dataset is available in the github project⁴. First, we present the average score for each statement and strategy across all

³<https://huggingface.co/mistralai/Mistral-Small-3.1-24B-Instruct-2503>

⁴<https://github.com/pzaragoza93/label-studio-pharo-evaluation>

Category	Statement ID	Statement
CRC Methodology	crc_methodology accuracy q1	The CRC format (Class name, Responsibility, Collaborators) is clearly respected.
CRC Methodology	crc_methodology accuracy q2	The comment clearly explains both what the package does and who it interacts.
CRC Methodology	crc_methodology accuracy q3	The generated comment contains a structured title, description of purpose, and list of external dependencies.
Responsibility	responsibility accuracy q1	The generated responsibility description correctly describes the core responsibilities of the package.
Responsibility	responsibility accuracy q2	The generated description of the package's responsibility is clear.
Responsibility	responsibility accuracy q3	The generated description of the package's responsibility is succinct.
Collaborators	collaborator accuracy q1	The package's main collaborators are mentioned and described accurately.
Collaborators	collaborator accuracy q2	The generated comment DOES NOT omit important external dependencies.
Collaborators	collaborator accuracy q3	The reason for the interactions between its collaborators is clearly explained.
Comparison	comparison accuracy q1	The generated comment is more complete than the original comment.
Comparison	comparison accuracy q2	The generated comment is more clear than the original comment.
Comparison	comparison accuracy q3	The generated comment is more useful than the original comment.

Table 1

List of statements, their category and statement ID used in the questionnaire.

packages and groups in table 2.

Statement	Strategy 1	Strategy 2	Strategy 3	ANOVA (p-value)
crc_methodology_accuracy_q1	6.438	6.188	6.188	0.618
crc_methodology_accuracy_q2	6.062	6.125	5.750	0.574
crc_methodology_accuracy_q3	6.188	6.125	6.062	0.962
responsibility_accuracy_q1	6.125	6.188	6.000	0.850
responsibility_accuracy_q2	6.188	6.188	5.625	0.180
responsibility_accuracy_q3	6.625	6.312	6.375	0.431
collaborator_accuracy_q1	4.500	5.062	4.438	0.614
collaborator_accuracy_q2	3.625	4.062	2.875	0.406
collaborator_accuracy_q3	4.625	5.000	4.000	0.395
comparison_accuracy_q1	6.188	6.625	6.188	0.463
comparison_accuracy_q2	6.375	6.562	6.188	0.414
comparison_accuracy_q3	6.125	6.562	6.250	0.492

Table 2

Average Likert score for each statement across all 3 strategies.

When we apply an analysis of variance (ANOVA) test to evaluate whether the difference between the three strategies across all 12 statements, we obtain a p-value above 0.05 across the board. Therefore, we cannot conclude that the difference is significant. With regards to RQ1, RQ2, RQ3, and RQ4, we cannot conclude that there is a strategy that is particularly better at increasing the quality of the comment based on the CRC methodology structure, the description of the responsibility, the description of the collaborators, nor quality when compared to the reference.

However, when we look at statements regarding the quality when compared to the reference (comparison_accuracy), we highlight the overall positive score of 6+ out of 7 across all three strategies. This

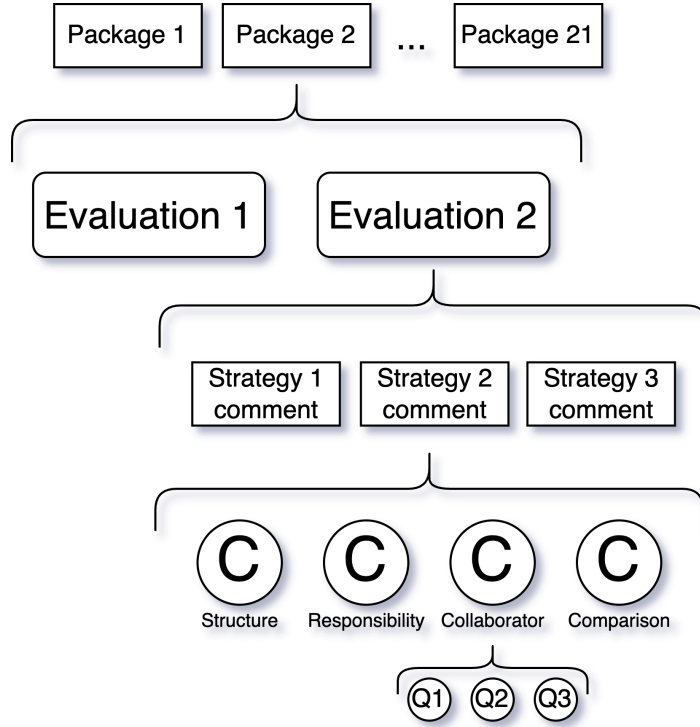


Figure 5: Illustration of the distribution of the package evaluation, in which 2 different experts evaluate the same

denotes that while a strategy is not particularly better than any other, applying any of these strategy does generate comments that are preferred over the existing comment in terms of completeness (q1), clarity (q2), and usefulness (q3).

Furthermore, statements related towards the description of the collaborators of a package contains the lowest average. This seems to indicate that LLMs have difficulty highlighting key collaborators within a package. Indeed, when asked for feedback, several evaluators reported that the listing of collaborators had a tendency to go on and on, to hallucinate certain relationships, and to not provide key details.

Statement	small	medium	large	ANOVA (p-value)
crc_methodology_accuracy_q1	6.583	6.500	6.000	0.107
crc_methodology_accuracy_q2	6.583	5.833	5.625	0.040
crc_methodology_accuracy_q3	6.583	6.333	5.750	0.167
responsibility_accuracy_q1	6.250	5.833	6.042	0.671
responsibility_accuracy_q2	6.333	5.667	5.875	0.334
responsibility_accuracy_q3	6.583	6.333	6.375	0.696
collaborator_accuracy_q1	5.833	4.500	4.333	0.061
collaborator_accuracy_q2	4.750	5.667	2.458	0.001
collaborator_accuracy_q3	6.083	5.333	3.958	0.004
comparison_accuracy_q1	6.667	6.833	6.042	0.166
comparison_accuracy_q2	6.583	6.833	6.167	0.086
comparison_accuracy_q3	6.750	6.833	6.000	0.048

Table 3

Average Likert score for each statement based on the package size.

Regarding **RQ5**, we calculate the average score for each statement based on the size of the packages. Furthermore, we apply an ANOVA test to evaluate whether the difference between the three strategies across all 12 statements are significant. This is the case for 4 state-

ments: `crc_methodology_accuracy_q2`, `collaborator_accuracy_q2`, `collaborator_accuracy_q3`, `comparison_accuracy_q3`. In general, there is a tendency for a better score when the package size is smaller. In the case of statements `collaborator_accuracy_q2` and `comparison_accuracy_q3`, generated comments for medium packages performed better followed by those generated for small packages. Overall, comments generated for large packages received the lowest score, which indicates that LLMs have a tougher time generating accurate comments the larger a package gets.

6. Conclusion

We proposed three distinct strategies for generating package comments to investigate how the choice of context may influence the quality of CRC-based package comments. While no statistically significant differences were found between the three strategies, all strategies-generated comments that were preferred over existing comments in terms of clarity, completeness, and usefulness. These results are encouraging in terms of LLMs use for better code comprehension and documentation quality.

However, several limitations must be acknowledged. First, the inter-agreement on manual evaluations remains to be addressed. While we attempted to reduce bias by introducing multiple ratings for each package, we were unable to reach a sufficient Kappa coefficient to confirm an inter-judge agreement. This may be caused by having an insufficient number of evaluators per package or a questionnaire that is not explained well enough.

In future works, we wish to address this by proposing a CI-integrated comment generation pipeline where the generated comments can be judged by active developers within the community through a submitted pull request. Furthermore, we propose enhancing the system's precision by reinforcing the comment generation with static analysis-based heuristics. For instance, studies in hallucinations reduction in models suggest graph-oriented data can help with reducing this type of hallucinations [7]. By providing a list of key collaborators through these heuristics we can reduce the hallucinations and anchor the generation to verified class interactions. In this hybrid approach, LLMs would focus on generating the responsibility descriptions of the package, while collaborator information is algorithmically selected to ensure correctness and consistency.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

- [1] X. Xia, L. Bao, D. Lo, Z. Xing, A. E. Hassan, S. Li, Measuring program comprehension: A large-scale field study with professionals, *IEEE Transactions on Software Engineering* 44 (2018) 951–976. doi:10.1109/TSE.2017.2734091.
- [2] K. Beck, W. Cunningham, A laboratory for teaching object oriented thinking, in: *Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications, OOPSLA '89*, Association for Computing Machinery, New York, NY, USA, 1989, p. 1–6. doi:10.1145/74877.74879.
- [3] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. Pollock, K. Vijay-Shanker, Automatic generation of natural language summaries for java classes, in: *2013 21st International Conference on Program Comprehension (ICPC)*, 2013, pp. 23–32. doi:10.1109/ICPC.2013.6613830.
- [4] T. Ahmed, P. Devanbu, Few-shot training llms for project-specific code-summarization, in: *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, ASE '22*, Association for Computing Machinery, New York, NY, USA, 2023. URL: <https://doi.org/10.1145/3551349.3559555>. doi:10.1145/3551349.3559555.

- [5] T. Ahmed, K. S. Pai, P. Devanbu, E. T. Barr, Automatic semantic augmentation of language model prompts (for code summarization), 2024. URL: <https://arxiv.org/abs/2304.06815>. arXiv:2304.06815.
- [6] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, P. Liang, Lost in the middle: How language models use long contexts, *Transactions of the Association for Computational Linguistics* 12 (2024) 157–173. doi:10.1162/tac1_a_00638.
- [7] M. Barry, G. Caillaut, P. Halftermeyer, R. Qader, M. Mouayad, F. Le Deit, D. Cariolaro, J. Gesnouin, GraphRAG: Leveraging graph-based efficiency to minimize hallucinations in LLM-driven RAG for finance data, in: G. A. Gesese, H. Sack, H. Paulheim, A. Merono-Penuela, L. Chen (Eds.), *Proceedings of the Workshop on Generative AI and Knowledge Graphs (GenAIK)*, International Committee on Computational Linguistics, Abu Dhabi, UAE, 2025, pp. 54–65.