Teamwork makes the dream work: LLMs-Based Agents for GitHub README.MD Summarization

Duc S. H. Nguyen
duc.nsh231061m@sis.hust.edu.vn
Hanoi University of Science and Technology
Hanoi, Vietnam

Bach G. Truong
bach.tg210087@sis.hust.edu.vn
Hanoi University of Science and Technology
Hanoi, Vietnam

Phuong T. Nguyen
phuong.nguyen@univaq.it
University of L'Aquila
67100 L'Aquila, Italy

Juri Di Rocco
juri.dirocco@univaq.it
University of L'Aquila
67100 L'Aquila, Italy

Davide Di Ruscio
davide.diruscio@univaq.it
University of L'Aquila
67100 L'Aquila, Italy

Abstract

The proliferation of Large Language Models (LLMs) in recent years has realized many applications in various domains. Being trained with a huge of amount of data coming from various sources, LLMs can be deployed to solve different tasks, including those in Software Engineering (SE). Though they have been widely adopted, the potential of using LLMs cooperatively has not been thoroughly investigated.

In this paper, we proposed `Metagente` as a novel approach to amplify the synergy of various LLMs. `Metagente` is a Multi-Agent framework based on a series of LLMs to self-optimize the system through evaluation, feedback, and cooperation among specialized agents. Such a framework creates an environment where multiple agents iteratively refine and optimize prompts from various perspectives. The results of these explorations are then reviewed and aggregated by a teacher agent. To study its performance, we evaluated `Metagente` with an SE task, i.e., summarization of README.MD files, and compared it with three well-established baselines, i.e., `GitSum`, `LLaMA-2`, and `GPT-4o`. The results show that our proposed approach works efficiently and effectively, consuming a small amount of data for fine-tuning but still getting a high accuracy, thus substantially outperforming the baselines. The performance gain compared to `GitSum`, the most relevant benchmark, ranges from 27.63% to 60.43%. More importantly, compared to using only one LLM, `Metagente` boots up the accuracy to multiple folds.

ACM Reference Format:
Duc S. H. Nguyen, Bach G. Truong, Phuong T. Nguyen, Juri Di Rocco, and Davide Di Ruscio. 2025. Teamwork makes the dream work: LLMs-Based Agents for GitHub README.MD Summarization. In *Companion Proceedings of the 33rd ACM Symposium on the Foundations of Software Engineering (FSE '25), June 23–27, 2025, Trondheim, Norway*. ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Large Language Models (LLMs) have transformed how we approach different tasks such as natural language understanding, creative writing, and software engineering [19]. However, despite the individual strengths of LLMs, no single model can fully address the vast range of human language and problem domains. For example, while LLMs like GPT have excelled in natural language understanding, their performance on tasks requiring domain-specific expertise or complex multi-step reasoning remains limited [11]. A potential solution involves prompt-tuning, where prompts are iteratively refined to improve the performance of the pre-trained language model without modifying its internal design. Although this approach has shown promise, it faces challenges when the LLM is accessible only via an API. Furthermore, manual prompt engineering techniques, such as Chain-of-Thought (CoT) reasoning, require significant human effort to refine prompts iteratively. This labor-intensive process is prone to subjective bias and scalability issues, making it difficult to generalize across diverse tasks [29].

To address the inherent limitations of single LLMs, researchers have proposed multi-agent systems that enable specialized LLMs to collaborate within a shared framework [26]. These systems capitalize on the unique strengths of different LLMs, where agents specialize in tasks such as code generation, debugging, or domain-specific problem-solving [11, 31]. For instance, frameworks like TransAgent have demonstrated how task-specific agents can integrate seamlessly to tackle complex engineering challenges [10]. However, these systems are not without limitations. Challenges such as effective coordination, efficient communication, and the overhead of integrating multiple agents persist. Additionally, designing robust frameworks for agent interaction and feedback loops requires considerable engineering and computational resources. Despite these constraints, the collaborative potential of multi-agent systems offers an interesting alternative to relying on single model's capabilities [27].

To illustrate a practical application of multi-agent LLM systems, we propose a novel framework targeting the problem of summarizing SE artifacts. This approach leverages the capabilities of specialized LLM agents to optimize task performance. The work supports a call for fundamentally new research directions with an initial evaluation on a real issue in SE, thus having the following contributions:

▹ **Solution**. We develop `Metagente`, an LLMs-based agents approach to perform the summarization of SE artifacts. A reciprocal teacher-student architecture is built with two components, i.e., the master module to perform the main task, and the optimization module to refine and enhance the master module. To the best of our knowledge, our work is the first one ever to study the applicability of LLMs-based agents in this domain.

▹ **Evaluation**. Even though our work explores a new direction, being still in early stages of research, we supported by initial evidence with a concrete SE task. In particular, we used the pipeline to perform the summarization of GitHub README.MD files. This task has been specifically chosen due to the following reasons: *(i)* Many GitHub repositories do not have an About description [7], posing difficulties to users when getting acquainted with their content; and *(ii)* The diversity of text, mark down content, and source code in README.MD files makes it difficult to produce a good summary.

FSE '25, June 23–27, 2025, Trondheim, Norway

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

▹ **Comparison**. An empirical evaluation has been conducted to compare `Metagente` with `GitSum` [7] and `LLaMA-2`, two state-of-the-art baselines.

▹ **Open Science**. A replication package including the dataset and source code of `Metagente` has been published to foster future research [2].

2 Related Work

2.1 LLMs-based Multi-Agent Systems

Integrating multi-agent systems and LLMs has facilitated the development of adaptive systems that enhance collaboration and productivity in SE. Several frameworks have emerged to address specific challenges in this domain, focusing on collaboration, modularity, and optimization.

OPRO [32] employs different LLMs as optimizers, leveraging natural language prompts to generate and refine solutions iteratively. Similarly, APE [35] generates candidate instructions and iteratively refines them using semantic similarity and evaluation metrics. Other frameworks, such as Camel [15] and MetaGPT [12], emphasize modularity and structured collaboration among agents. Camel introduces a role-playing framework that guides chat agents using inception prompting, aligning their actions with human intentions. This enables instruction-following cooperation and generates conversational data that advances the understanding of multi-agent behaviors. MetaGPT, on the other hand, incorporates human workflows as a meta-programming approach to address challenges like hallucination.

Frameworks like AutoGen [30] and LangChain[1] focus on leveraging LLMs for modular and conversation-driven application development. AutoGen facilitates the creation of LLM-based applications through conversable agents that can autonomously engage in multi-turn conversations, incorporate human feedback, and combine modular capabilities. Its conversation programming paradigm simplifies the definition of agent roles and interaction behaviors across a variety of domains, including coding and operations research. LangChain complements this effort by providing components and customizable pipelines for integrating external data sources and interacting with other applications. Its modular abstractions and chains streamline the development of applications [1, 24, 25, 27].

2.2 LLMs for summarization tasks

The SE community has increasingly explored the application of summarization techniques across a variety of tasks [17]. Integrating LLMs and pre-trained model into text summarization has garnered significant attention due to their potential to streamline various domain-specific tasks. A recent study [21] compared five pre-trained models across three different natural language processing (NLP) tasks: text classification, summarization, and generation. The results of the text summarization task revealed that GPT, BART, and LLaMA achieved the highest accuracy among others.

Prompt engineering has been explored in various domains as well. Bajaj and Borhan [3] compared zero-shot, few-shot, and role-based prompting techniques for summarizing diverse articles. Their findings emphasized the versatility of simple zero-shot prompts, which consistently outperformed other methods. Similarly, Oliveira and Lins [18] evaluated both abstractive and extractive summarization methods, highlighting the superior performance of Pegasus for news summarization tasks.

Doan et al. [7] introduced `GitSum`, a novel approach to summarizing README.MD content. `GitSum` is built upon BART and T5 to recommend descriptions for repositories that lack such metadata. Similarly, FILLER [14] is a solution for generating titles for Stack Overflow posts by leveraging a fine-tuned language model equipped with self-improvement mechanisms and post-ranking capabilities. FILLER adopts a multi-task learning strategy, fine-tuning the CodeT5 model on a dataset of Stack Overflow posts while simultaneously training across multiple programming languages [14].

[1] https://www.langchain.com/

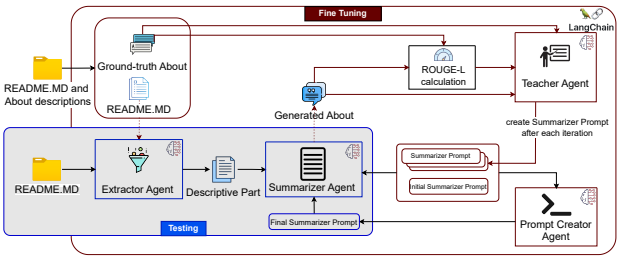


Figure 1: The proposed `Metagente` pipeline.

3 Proposed Approach

Fig. 1 depicts the proposed framework to create an environment where multiple LLMs-based agents iteratively refine and optimize the outcome. It is a teacher-student architecture with 2 components: the main module, which generates "About" content from README.MD text, and the optimization module, which participates only in the training phase to refine and enhance the main module.

3.1 LLMs-based Agents

The pipeline in Fig. 1 consists of 4 agents, i.e., `Extractor Agent`, `Summarizer Agent`, `Teacher Agent`, and `Prompt Creator Agent`. Throughout the pipeline, prompts are used to instruct the agents, and a scoring mechanism based on the ROUGE metrics is utilized to guide the LLMs in identifying the most suitable actions. *Due to the strict space limit, we cannot show the prompts here, but upload them to the online appendix for the sake of references* [2].

▹ `Extractor Agent`. Being steered by prompts, this agent removes irrelevant and noisy details from input README.MD files, focusing only on content that introduces or describes the repository. A README.MD might contain many sections such as introduction, description, installation, contributing, license, and `Extractor Agent` needs to filter out all sections not relevant to the description of the repository to yield a short and concise descriptive text. During optimization, only this descriptive text is used instead of the entire README.MD file, so as to significantly reduce time and computational resources.

▹ `Summarizer Agent`. It begins with a predefined initial prompt and uses it to summarize the extracted text from a README.MD file. Through an iterative process, during each iteration the agent takes as input an updated summarizer prompt generated by the `Teacher Agent`. The goal is to produce a concise About description that captures the core concept or purpose of the repository, focusing on key terms, features, and specific context without including explanations or extraneous details. The final output should be a short and precise phrase that enhances clarity and relevance while reflecting the repository's essential idea.

▹ `Teacher Agent`. By reviewing the following four inputs: *(1)* current `Summarizer Agent` prompt, *(2)* generated About, *(3)* ground-truth About, and *(4)* ROUGE-L scores, this agent optimizes `Summarizer Agent`'s prompt for each training sample. A complex prompt with various steps guides `Summarizer Agent` in analyzing the input features, and improving the current prompt of `Summarizer Agent`.

▹ `Prompt Creator Agent`. Accepting as input a set of prompts, the agent analyzes and identifies common parts to produce the final prompt. It extracts specific details or conditional key points from the input prompts to be included in the final prompt. Being derived from these seed data instances, the final prompt serves as an overall guide for `Summarizer Agent`'s inference task. It provides high-level instructions while offering specific guidance on key details tailored to the nuances of the training data.

For `Extractor Agent` and `Summarizer Agent`, we opted for OpenAI's `GPT-4o-mini` as the LLM engine. For `Teacher Agent` and `Prompt Creator`

Agent, we leveraged the more advanced GPT-4o model. This hierarchical deployment ensures that the larger and more advanced LLMs guide the smaller models, optimizing their performance without incurring excessive computational costs during inference. This helps the final system maintain a balance of high efficiency, cost-effectiveness, and optimal performance.

3.2 Orchestration of Agents

Metagente is a cooperative pipeline where LLMs-based agents work together towards a shared objective by means of three main phases, i.e., *Communications*, *Self Improvement*, and *Prompt Generation* as follows.

▹ **Communications**. An agent works by interacting with the environment and other agents. In our pipeline, besides LangChain, that has been adopted as the communication backbone, we also employed a structured output mechanism to guide the LLMs to produce information that is easily digestible as input for the subsequent agents. This is to ensure consistency and seamlessness in exchanging messages among the agents.

▹ **Self Improvement**. At the beginning of each iteration, the extracted text is fed as input to Summarizer Agent. Based on the current prompt for that iteration, Summarizer Agent generates a short description as output. To evaluate the generated About, we use the ROUGE metrics, which have been widely adopted for text summarization evaluation [5, 7, 9]. Specifically, we focus on the ROUGE-L score to facilitate the comparison of results across iterations. The current Summarizer Agent prompt, generated About, ground-truth About, and ROUGE-L scores are used as input for Teacher Agent. Being guided with dedicated prompts, the agent compares the generated About with the ground-truth About to find out the key differences, and propose the necessary improvements to Summarizer Agent 's prompt.

Teacher Agent generates a new prompt, which is used by Summarizer Agent in the next iteration. During the optimization process, we impose a limit on the number of iterations. The termination is met when: *(i)* The ROUGE-L score of the current generated About reaches a predefined threshold; *(ii)* or the maximum number of iterations is exceeded. After experimenting with multiple configurations, we set the following hyperparameters: *(i)* ROUGE-L threshold = 0.7; and maximum number of iterations = 15.

At the end of the optimization, we obtain a final prompt for each training data instance. Instances failing to meet the ROUGE-L threshold within the allowed iterations are discarded. The remaining successful data instances are referred to as seed training data, as they best represent and generalize the dataset in terms of converting README.MD text into an About description.

▹ **Prompt Generation**. The optimization flow on a selected dataset results in a set of distinct prompts for Summarizer Agent, each of them is optimized to deliver strong performance on a specific data instance. This set of prompts is then passed as the sole input for Prompt Creator Agent, which extracts common instructions shared across all candidate prompts, while identifying and integrating key conditional points. These points guide Summarizer Agent in adapting its structure and writing style to suit the varying contexts and characteristics of different README.MD, ensuring consistent and high-quality About content generation under diverse scenarios.

4 Proof of Concept

4.1 Research questions

We perform various experiments to answer the following research questions.

▹ **RQ$_1$**: *Does the use of multi LLMs-based agents result in more relevant About descriptions?* This research question compares Metagente with a system that has only one GPT-4o summarization engine. We study if the combination of a series of LLMs is really needed, considering the fact that a single LLM agent might possibly be sufficient to get a good recommendation performance. This is important in practice as a simple yet effective approach is preferable in the era of Generative AI.

▹ **RQ$_2$**: *How does* Metagente *perform compared to* GitSum *and* LLaMA-2*?* We compare Metagente with GitSum [7]–a state-of-the-art tool in summarizing README.MD files, and LLaMA-2–a large language model that has been widely applied in various SE tasks [13], and summarization [8, 20].

4.2 Dataset and Metrics

We adopted an existing dataset for README.MD-related tasks [7], and extended it by incorporating a diverse range of data sources to enhance its comprehensiveness and applicability. First, we augmented the initial dataset with GitHub repositories categorized under awesome-lists and documentation-related topics[2] that align with the document repositories category [33]. Then, we enriched the dataset by including curated repositories containing popular Python projects [22], Jupyter notebooks for data analysis [4, 23]. This approach is motivated by the observation that popular repositories often feature well-maintained and detailed README.MD files [28]. As a result, we compiled 6,933 unique repositories containing at least a README.MD file. Through a manual inspection, we noticed that by several repositories, the About descriptions do not match with what was written by the README.MD files. This happens because developers changed the README.MD files, but then forgot to update the corresponding About. Thus, we filtered out those and eventually obtained 925 samples, from which 2 training sets were randomly selected with 10 and 50 samples to yield TS_{10} and TS_{50}, and a testing set of 865 samples named as *ES*.

To evaluate the recommendations, we use the ROUGE metrics, i.e., ROUGE-1, ROUGE-2, ROUGE-L, which have been widely used in text summarization [6, 34]. Due to space limit, we cannot recall them here, interested readers are kindly referred to the work of Lin [16] for greater details.

4.3 Settings

During the experiments, we observed that Extractor Agent adapted effectively to various README.MD texts without encountering significant challenges. Thus, the prompt for Extractor Agent was pre-optimized and remained fixed throughout the entire optimization. We compared Metagente with three baselines: GitSum (based on BART), fine-tuned LLaMA-2, and GPT-4o. These methods were identified as the most promising ones in a recent study [21]. For the first experiment, we used TS_{50} with 50 README.MD files for training Metagente and the selected baselines. In the second experiment, we decreased the number of samples to 10, i.e., the TS_{10} dataset. This aims to study whether the tools are capable of generating About descriptions when an extremely small amount of data is available for training. This is useful in practice, as curating a suitable dataset for training is both time consuming and prone to error, and a model that can produce relevant About descriptions given a limited amount of training data is preferable.

5 Results and Discussion

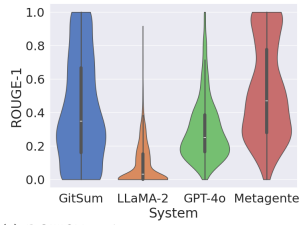
Fig. 2 and Fig. 3 report the ROUGE scores using violin boxplots. By each subfigure, we compute the average scores of GitSum, LLaMA-2, GPT-4o, and Metagente. In brackets we depict the increase in percent (a green up arrow ↑) of the average score of Metagente compared to those of the baselines.

5.1 Result Analysis

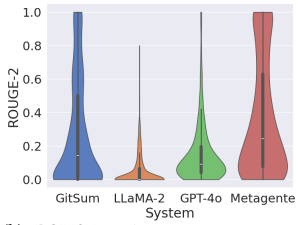
▹ **RQ$_1$: *Does the use of multi LLMs-based agents result in more relevant About descriptions?***

For the comparison between a single GPT-4o and Metagente, we consider the third and fourth boxplots in Fig. 2 and Fig. 3, corresponding to the use of TS_{50} and TS_{10} for fine tuning (training), respectively and *ES* for testing. Overall, it is evident that Metagente obtains a better accuracy compared to that of GPT-4o in terms of ROUGE-1, ROUGE-2, and ROUGE-L. As shown in Fig. 2(a), by most of the testing instances, GPT-4o gets a

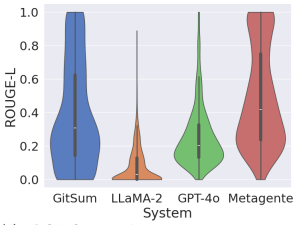
[2] https://github.com/topics



(a) ROUGE-1: Average 0.409, 0.162, 0.282, **0.522** (↑27.63%, ↑222.22%, ↑85.11%)

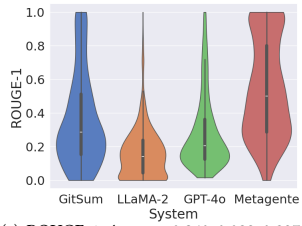


(b) ROUGE-2: Average 0.272, 0.100, 0.152, **0.363** (↑33.46%, ↑263.00%, ↑138.82%)

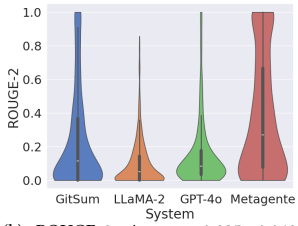


(c) ROUGE-L: Average 0.387, 0.146, 0.250, **0.486** (↑25.58%, ↑232.88%, ↑94.40%)

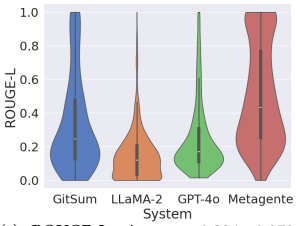
Figure 2: Comparison of `GitSum`, `LLaMA-2`, `GPT-4o`, and `Metagente`: TS_{50} is used for training and fine tuning.



(a) ROUGE-1: Average 0.360, 0.088, 0.297, **0.536** (↑48.89%, ↑509.09%, ↑80.47%)



(b) ROUGE-2: Average 0.235, 0.049, 0.151, **0.377** (↑60.43%, ↑669.39%, ↑149.67%)



(c) ROUGE-L: Average 0.334, 0.079, 0.256, **0.503** (↑50.60%, ↑536.71%, ↑96.48%)

Figure 3: Comparison of `GitSum`, `LLaMA-2`, `GPT-4o`, and `Metagente`: TS_{10} is used for training and fine tuning.

ROUGE-1 score lower than 0.3. Meanwhile, `Metagente` performs better as the density of the violin boxplots is concentrated on the 0.5 to 1.0 range, corresponding to superior ROUGE-1 scores by the majority of the testing samples. Looking at the average score, we see that `GPT-4o` and `Metagente` get 0.282 and 0.522, respectively, resulting in an increase of 85.11% by `Metagente` compared to `GPT-4o`. Such a gain is greater by the ROUGE-2 scores in Fig. 2(b), which shows that `Metagente` gets 0.363 as ROUGE-2 score, being greater than 0.152, the corresponding score obtained by `GPT-4o`, yielding an improvement of 138.82%. The same trend is seen with the ROUGE-L scores (Fig. 2(c)), i.e., `Metagente` performs better than `GPT-4o`.

When fewer samples are used for training, i.e., TS_{10} includes only 10 README.MD files and About descriptions, as shown in Fig. 3, the gain by `Metagente` compared to `GPT-4o` is much greater. In particular, the increase is 80.47%, 149.67%, and 96.48% by ROUGE-1, ROUGE-2, and ROUGE-L, respectively. Such a difference implies that `Metagente` is effective, even when there is a limited amount of data for fine tuning.

We ran Wilcoxon rank tests on every pair of ROUGE scores of `Metagente` and `GPT-4o` for the whole testing set with 865 samples, and got the following p-values: p=8.62e-89 (ROUGE-1), p=3.40e-72 (ROUGE-2), p=4.91e-91 (ROUGE-L). The rank tests confirm that *the performance difference obtained by* `Metagente` *in relation to* `GPT-4o` *is statistically significant.*

> ***Answer to RQ$_1$:*** Compared to a single LLM agent, the combination of multi LLMs-based agents is clearly advantageous, as it brings a lot more precise About descriptions with respect to all the ROUGE scores.

▹ **RQ$_2$: *How does* `Metagente` *perform compared to* `GitSum` *and* `LLaMA-2`*?***

We refer to Fig. 2 and Fig. 3 again for the comparison between `Metagente` and the two baselines, i.e., `GitSum` and `LLaMA-2` on the TS_{50} and TS_{10} datasets. Concerning ROUGE-1, as shown in Fig. 2(a), `GitSum` is much better with respect to `LLaMA-2` as most of its scores are scattered between 0.2 and 0.5; it also obtains ROUGE-1 score of 1.0 by different testing samples, while the accuracy of `LLaMA-2` is much lower, with no score being seen at the 1.0 level. `Metagente` outperforms both baselines since it has more scores on the range from 0.6 to 1.0. The density of the 1.0 level by `Metagente` is also larger than that of `GitSum` and `GPT-4o`. On the average score, `Metagente` gets 0.522, which is better than 0.409 and 0.162–the corresponding values achieved by `GitSum` and `LLaMA-2`, resulting in a gain of 27.63% and 222.22%, respectively. The difference in performance of `Metagente` compared to the baselines is more evident with ROUGE-2 in Fig. 2(b), i.e., 33.46% and 263.00%; and ROUGE-L in Fig. 2(c), i.e., 25.58% and 232.88%.

Similarly, the results for the TS_{10} dataset in Fig. 3 show that `Metagente` outweighs the baselines by all the three metrics. Especially, with the average ROUGE-2 score, `Metagente` yields an increase of 669.39% compared to `LLaMA-2`. Wilcoxon rank tests reveal that the performance gain is always statistically significant as all the p-values are much smaller than 5e-2.[3]

> ***Answer to RQ$_2$:*** `Metagente` outperforms both `GitSum` and `LLaMA-2` by all the three metrics, i.e., ROUGE-1, ROUGE-2, and ROUGE-L. Our tool is robust even when there is a small amount of data for fine tuning.

▹ **Timing performance**. An important factor is the timing efficiency of `Metagente` for fine tuning and testing. While the pipeline was implemented by us to orchestra the agents, and run on a normal laptop, all the computations are performed on OpenAI's servers, since `GPT-4o-mini` and `GPT-4o` are utilized as the LLM engines. We recorded the time and got the following information: With the TS_{50} dataset, `Metagente` takes 3 minutes for fine tuning and 8 minutes for testing on *ES* (865 samples).

5.2 Discussion

▹ **Applicability.** The evaluation demonstrates that the use of a single `GPT-4o` agent does not suffice to produce precise About descriptions. Thus, our proposed pipeline is meaningful since it combines the strength of different LLMs-based agents to perform the task. The agents reciprocally enforce each other by means of prompts, being able to self-optimize through evaluation and feedback. In other words, teamwork allows LLMs to augment the

[3]Due to space limit, we publish the statistical test results in the replication package [2].

synergies among them, thus achieving a superior performance. `Metagente` needs a few samples for fine tuning, and this is practical in real-world usages, as curating a proper dataset is time consuming and error prone.

▹ **Limitations.** While `Metagente` obtains a promising performance for the dedicated task, there are different aspects to be considered. We suppose that the prompts used to guide the agents could be further optimized to improve the overall performance. In this paper, 4 agents were used for the summarization, depending on the tasks, we may extend the architecture to have more tailored LLMs-based agents.

▹ **Threats to Validity.** *(i) Internal validity*: We compared `Metagente` with `GitSum` using the original implementation of `GitSum`. For the comparison with the baselines, we used the same set of data for fine tuning (training), and testing; *(ii) External validity*: The findings are valid for the dataset used in this paper. Repositories collected from GitHub are heterogeneous, and thus requiring additional prompts to preprocess and clean the data.

6 Conclusion and Future work

In this paper, we conceptualized `Metagente` as a practical approach to tackle the issue of summarization for GitHub README.MD files, leveraging LLMs-based agents. An empirical evaluation conducted on `Metagente` using a dataset collected from GitHub showed that our proposed approach augments the strengths of different agents, and thus outperforming a single agent as well as two state-of-the-art baselines. For future work, we plan to study the applicability of the framework in other domains in Software Engineering, such as code completion, or code review.

Acknowledgments

This paper was partially supported by the MOSAICO project (Management, Orchestration and Supervision of AI-agent COmmunities for reliable AI in software engineering) that has received funding from the European Union under the Horizon Research and Innovation Action (Grant Agreement No. 101189664). The work has been partially supported by the EMELIOT national research project, which has been funded by the MUR under the PRIN 2020 program (Contract 2020W3A5FY). It has been also partially supported by the European Union–NextGenerationEU through the Italian Ministry of University and Research, Projects PRIN 2022 PNRR *"FRINGE: context-aware FaiRness engineerING in complex software systEms"* grant n. P2022553SL. We acknowledge the Italian "PRIN 2022" project TRex-SE: *"Trustworthy Recommenders for Software Engineers,"* grant n. 2022LKJWHC.

References

[1] Rakha Asyrofi, Mutia Rahmi Dewi, Muhammad Irfan Lutfhi, and Prasetyo Wibowo. 2023. Systematic Literature Review Langchain Proposed. In *2023 International Electronics Symposium (IES)*. 533–537. https://doi.org/10.1109/IES59143.2023.10242497

[2] Anonymous Authors. 2025. Replication package for "Teamwork makes the dream work: LLMs-Based Agents for GitHub README Summarization". https://anonymous.4open.science/r/Metagente-43E5

[3] Akhilesh Bajaj and Iffat Borhan. 2024. The Effect of Prompt Types on Text Summarization Performance With Large Language Models. *J. Database Manage.* 35, 1 (Nov. 2024), 1–23. https://doi.org/10.4018/JDM.358475 Read_Status: New Read_Status_Date: 2025-01-10T09:37:02.712Z tex.ids= bajajEffectPromptTypes2024a.

[4] Hudson Borges, Andre Hora, and Marco Tulio Valente. 2016. Understanding the Factors That Impact the Popularity of GitHub Repositories. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, Raleigh, NC, USA, 334–344. https://doi.org/10.1109/ICSME.2016.31

[5] Songqiang Chen, Xiaoyuan Xie, Bangguo Yin, Yuanxiang Ji, Lin Chen, and Baowen Xu. 2021. Stay professional and efficient: automatically generate titles for your bug reports. In *Proceedings of the 35th ASE (ASE '20)*. ACM, New York, NY, USA, 385–397. https://doi.org/10.1145/3324884.3416538

[6] Songqiang Chen, Xiaoyuan Xie, Bangguo Yin, Yuanxiang Ji, Lin Chen, and Baowen Xu. 2021. Stay Professional and Efficient: Automatically Generate Titles for Your Bug Reports. In *Proceedings of the 35th ASE* (Virtual Event, Australia) *(ASE '20)*. Association for Computing Machinery, New York, NY, USA, 385–397. https://doi.org/10.1145/3324884.3416538

[7] Thu T. H. Doan, Phuong T. Nguyen, Juri Di Rocco, and Davide Di Ruscio. 2023. Too long; didn't read: Automatic summarization of GitHub README.MD with Transformers. In *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering* (Oulu, Finland) *(EASE '23)*. Association for Computing Machinery, New York, NY, USA, 267–272. https://doi.org/10.1145/3593434.3593448

[8] Wei Feng, Huan Zhao, Min Zhang, Hao Yang, and Wei Tang. 2024. A Novel Summarization Framework based on Reference-Free Evaluation of Multiple Large Language Models. In *2024 IEEE International Conference on Metaverse Computing, Networking, and Applications (MetaCom)*. 247–252. https://doi.org/10.1109/MetaCom62920.2024.00047

[9] Zhipeng Gao, Xin Xia, John Grundy, David Lo, and Yuan-Fang Li. 2020. Generating Question Titles for Stack Overflow from Mined Code Snippets. *ACM Trans. Softw. Eng. Methodol.* 29, 4, Article 26 (Sept. 2020), 37 pages. https://doi.org/10.1145/3401026

[10] Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large Language Model based Multi-Agents: A Survey of Progress and Challenges. arXiv:2402.01680 (April 2024). https://doi.org/10.48550/arXiv.2402.01680

[11] Junda He, Christoph Treude, and David Lo. 2025. LLM-Based Multi-Agent Systems for Software Engineering: Literature Review, Vision and the Road Ahead. *ACM Trans. Softw. Eng. Methodol.* (Jan. 2025). https://doi.org/10.1145/3712003 Just Accepted.

[12] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=VtmBAGCN7o

[13] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. *ACM Trans. Softw. Eng. Methodol.* 33, 8, Article 220 (Dec. 2024), 79 pages. https://doi.org/10.1145/3695988

[14] Duc Anh Le, Bui Thi-Mai-Anh, Phuong T. Nguyen, and Davide Di Ruscio. 2024. Good things come in three: Generating SO Post Titles with Pre-Trained Models, Self Improvement and Post Ranking. In *Proceedings of the 18th ESEM* (Barcelona, Spain) *(ESEM '24)*. Association for Computing Machinery, New York, NY, USA, 212–222. https://doi.org/10.1145/3674805.3686682

[15] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2024. CAMEL: communicative agents for "mind" exploration of large language model society. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 2264, 18 pages.

[16] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*. Association for Computational Linguistics, Barcelona, Spain, 74–81. https://aclanthology.org/W04-1013

[17] Antonio Mastropaolo, Matteo Ciniselli, Massimiliano Di Penta, and Gabriele Bavota. 2024. Evaluating Code Summarization Techniques: A New Metric and an Empirical Characterization. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 218:1–218:13. https://doi.org/10.1145/3597503.3639174

[18] Hilário Oliveira and Rafael Dueire Lins. 2024. Assessing Abstractive and Extractive Methods for Automatic News Summarization. In *Proceedings of the ACM Symposium on Document Engineering 2024 (DocEng '24)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/3685650.3685664 Read_Status: New Read_Status_Date: 2025-01-10T09:31:41.724Z tex.ids= oliveiraAssessingAbstractiveExtractive2024a.

[19] Ipek Ozkaya. 2023. Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications. *IEEE Softw.* 40, 3 (2023), 4–8. https://doi.org/10.1109/MS.2023.3248401

[20] Prakrit Pathak and Prashant Singh Rana. 2024. Comparative Analysis of Pretrained Models for Text Classification, Generation and Summarization: A Detailed Analysis. In *Pattern Recognition - 27th International Conference, ICPR 2024 (Lecture Notes in Computer Science)*, Apostolos Antonacopoulos, Subhasis Chaudhuri, Rama Chellappa, Cheng-Lin Liu, Saumik Bhattacharya, and Umapada Pal (Eds.), Vol. 15301. Springer, 151–166. https://doi.org/10.1007/978-3-031-78107-0_10

[21] Prakrit Pathak and Prashant Singh Rana. 2025. Comparative Analysis of Pretrained Models for Text Classification, Generation and Summarization: A Detailed Analysis. In *Pattern Recognition*, Apostolos Antonacopoulos, Subhasis Chaudhuri, Rama Chellappa, Cheng-Lin Liu, Saumik Bhattacharya, and Umapada Pal (Eds.). Springer Nature Switzerland, Cham, 151–166.

[22] 2021-03-23 popular-3k python. 2023. Dataset — Software Heritage documentation. https://docs.softwareheritage.org/devel/swh-dataset/graph/dataset.html

[23] Adam Rule, Aurélien Tabard, and James D. Hollan. 2018. Exploration and Explanation in Computational Notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3173574.3173606

[24] Fatih Soygazi and Damla Oguz. 2023. An Analysis of Large Language Models and LangChain in Mathematics Education. In *Proceedings of the 2023 7th International Conference on Advances in Artificial Intelligence*. 92–97.

[25] Oguzhan Topsakal and Tahir Cetin Akinci. 2023. 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. 1050–1056.

[26] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. 2024. A survey on large language model based autonomous agents. *Frontiers Comput. Sci.* 18, 6 (2024), 186345. https://doi.org/10.1007/S11704-024-40231-1

[27] Luqiao Wang, Yangtao Zhou, Huiying Zhuang, Qingshan Li, Di Cui, Yutong Zhao, and Lu Wang. 2024. Unity Is Strength: Collaborative LLM-Based Agents for Code Reviewer Recommendation. In *Proceedings of the 39th IEEE/ACM ASE (ASE '24)*. ACM, New York, NY, USA, 2235–2239. https://doi.org/10.1145/3691620.3695291

[28] Tianlei Wang, Shaowei Wang, and Tse-Hsun (Peter) Chen. 2022. Study the Correlation between the Readme File of Github Projects and Their Popularity. https://doi.org/10.2139/ssrn.4281782

[29] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C. Schmidt. 2023. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. arXiv:cs.SE/2302.11382 https://arxiv.org/abs/2302.11382

[30] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2024. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversations. In *First Conference on Language Modeling*. https://openreview.net/forum?id=BAakY1hNKS

[31] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. 2024. Agentless: Demystifying LLM-based Software Engineering Agents. *CoRR* abs/2407.01489 (2024). https://doi.org/10.48550/ARXIV.2407.01489 arXiv:2407.01489

[32] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. 2024. Large Language Models as Optimizers. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. https://openreview.net/forum?id=Bb4VGOWELI

[33] Francisco Zanartu, Christoph Treude, Bruno Cartaxo, Hudson Silva Borges, Pedro Moura, Markus Wagner, and Gustavo Pinto. 2022. Automatically Categorising GitHub Repositories by Application Domain. https://doi.org/10.48550/arXiv.2208.00269 arXiv:2208.00269 [cs].

[34] Ting Zhang, Ivana Clairine Irsan, Ferdian Thung, DongGyun Han, David Lo, and Lingxiao Jiang. 2022. iTiger: An Automatic Issue Title Generation Tool *(ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 1637–1641. https://doi.org/10.1145/3540250.3558934

[35] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023. Large Language Models are Human-Level Prompt Engineers. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. https://openreview.net/forum?id=92gvk82DE-