An LLM-Powered Tool for Enhancing Scientific Open-Source Repositories

Nikolay O. Nikitin [1] **Andrey Getmanov** [1] **Zakhar Popov** [1] **Ekaterina Ulyanova** [1] **Yaroslav Aksenkin** [1]
Ilya Sokolov [1] **Alexander Boukhanovsky** [1]

Abstract

We present OSA (Open Source Advisor), an open-source tool that uses large language models (LLMs) to improve scientific repositories. OSA uses an agent-based architecture to automate tasks such as generating README files, docstrings, and documentation, configuring CI/CD pipelines, creating tag annotations, and analyzing repositories to provide recommendations on best practice. Unlike code-generation tools, OSA focuses on enhancing existing repositories. We have validated OSA's effectiveness across diverse scientific repositories in multiple domains. OSA is available in https://github.com/aimclub/OSA.

1. Introduction

In recent years, the scientific community has witnessed unprecedented growth in the production and sharing of research outputs (Bornmann et al., 2021), facilitated by the growing acceptance of open-source principles (McKiernan et al., 2016). Scientific repositories have emerged as vital infrastructures that allow researchers to store, share, and collaborate on datasets, algorithms, and trained models (Benureau & Rougier, 2018). However, despite their importance, many scientific repositories face challenges related to usability and reproducibility (Trisovic et al., 2022; Färber, 2020).

The scientific code is difficult to read, there is no documentation, and sometimes not even trivial README files are available. Libraries and frameworks often lack basic CI/CD settings: linters, autotests, etc. That is why it turns out to be very hard to reproduce the formally open-sourced results (Ivie & Thain, 2018). There are many initiatives to increase the quality of scientific code (Thimbleby, 2024). However, it is hard to motivate scientists to do a large amount of routine software engineering work.

The involvement of LLM-based agents can be a useful solution to this problem. It is widely considered a valuable tool for the improvement of both scientific research (Nejjar et al., 2025) and software development (He et al., 2024). The current state-of-the-art LLM-driven tools can even generate complex scientific code from the text of the paper (Seo et al., 2025). However, the more common case in the scientific workflow is the need to improve the existing scientific code prior to publication.

To reduce the amount of routine work involved in preparing scientific repositories, we propose an open-source tool that leverages large language model (LLM) agents to enhance the preparation process. By integrating LLM agents into scientific workflows, we aim to achieve a more intuitive and user-friendly experience for reproducible research.

The proposed tool focuses on several key functionalities: (1) preparation of basic documentation for existing code (README, docstrings); (2) analysis of the compliance of existing repository with best practices; (3) improving the structure and configuration of CI/CD pipelines in repository;

This paper describes the architecture of the OSA ((Open Source Advisor) - open source tool, details the underlying LLM-based tools, and presents preliminary results for case studies that demonstrate its potential to improve the usability and effectiveness of scientific repositories. We believe that this approach not only enhances the experience of researchers but also contributes to the broader goal of making scientific knowledge more accessible and reproducible.

2. Related Works

2.1. Code generation and improvement

LLMs are already widely used for code generation in scientific applications (Nejjar et al., 2025). They are also applicable to the improvement of existing code quality (Trofimova et al., 2024).

Tools such as GitHubGPT demonstrate the potential of LLMs for analyzing and synthesizing code for GitHub repositories, while utilities like pipreqs automate dependency

[1]NSS Lab, AI Institute, ITMO University, Saint-Petersburg, Russia. Correspondence to: Nikolay O. Nikitin <nnikitin@itmo.ru>.

Proceedings of the ICML 2025 Workshop on Championing Open-source Development in Machine Learning (CODEML '25).

management by inferring requirements from Python imports. While these systems streamline code creation and dependency resolution, they primarily target generative tasks rather than systematic improvement of existing repositories.. This distinction highlights a critical gap in tools designed to audit and refine established codebases — which is the central focus of OSA.

2.2. Documentation generation

Documentation generation has seen notable innovation, with frameworks like docstring-gen leveraging Codex to produce structured docstrings, and readme-ai employing repository analysis to generate README templates. There is a lot of researches on further improvement of README generation quality (Koreeda et al., 2023).

Specialized tools such as ts-readme-generator and mermaidjs-github-svg-generator further automate API documentation and UML diagram integration. However, these efforts often operate in isolation, addressing single components (e.g., docstrings or READMEs) without unified workflows. OSA advances this paradigm by integrating multi-level documentation synthesis—from inline docstrings to repository-wide guides—through a cohesive LLM-driven pipeline.

2.3. Repository maintenance and analysis

Infrastructure tools like SonarQube and CodeClimate enable continuous code quality inspection, while gitinspector and repo-analyzer provide statistical insights into contributor activity and repository health. Emerging solutions such as repo-map employ LLMs to generate architectural summaries, yet lack actionable recommendations for improvement. OSA extends these capabilities by combining static analysis with context-aware best practices, automating CI/CD configuration, tag annotation, and repository-specific optimizations. This agentic approach distinguishes OSA from passive analysis tools, enabling proactive, end-to-end repository enhancement. The other similar tool is RepoAgent `https://github.com/OpenBMB/RepoAgent`, which is an LLM-powered repository agent designed to generate documentation and analyze repositories. Also, tools like (Phan et al., 2024) and (Yang et al., 2024) allow achieve better interaction with the repository.

2.4. Benchmarks

There are many benchmarks for code generation and its accompanying documentation. For example, ML-Bench (Tang et al., 2023) evaluates code generation quality at the repository level. As a benchmark for documentation, *generate-readme-eval* (`https://huggingface.co/datasets/patched-codes/generate-readme-eval`) can be considered. EnvBench (Eliseeva et al., 2025) focuses on the task of automated environment setup. However, there are currently no benchmarks available that directly evaluate the quality of scientific repository improvement across various aspects.

3. OSA: approach for improvement of scientific repositories

3.1. Motivation

Scientific code often operates within specialized domains such as chemistry, physics, or climate modeling, requiring a deep understanding of both the underlying scientific principles and the computational methods used to model them. This complexity results in code that is not only intricate but also heavily dependent on specific libraries, frameworks, and data formats that may be unfamiliar to many users. As a result, scientific code can present significant barriers to entry for researchers from other disciplines or those new to the field.

Various tasks related to improving open source repositories can be addressed using LLM, including: retrieval-augmented generation (RAG) for documentation, code review, automated bug fixes, updating examples, analysis of test results, and Q&A about code and documentation. We aim to automate the solution of these tasks in OSA as much as possible.

3.2. Workflow, agents and tools

The structure of the OSA workflow is presented in Figure 1. It consists of several tools and planner agent that automates the selection of appropriate tool.

There are several self-implemented tools implemented in OSA: (1) tool for README generation; (2) tool for docstrings generation; (3) tool for generation of reports; (4) tool for CI/CD scripts generation.

The integrated automatic scheduler in OSA operates in three distinct modes: basic, automatic, and advanced, selected at runtime. In the basic mode, the tool performs a minimal set of predefined improvements without additional user interaction. The advanced mode provides full manual control, allowing the user to manage all operations directly. In automatic mode, the scheduler employs an LLM to analyze the repository structure and the existing README file. Based on this analysis, it generates a set of proposed enhancements that the user may accept or decline.

The main tool implemented in OSA automates the generation of README files, offering two formats: a standard README style and an article style, selectable via a hyper-

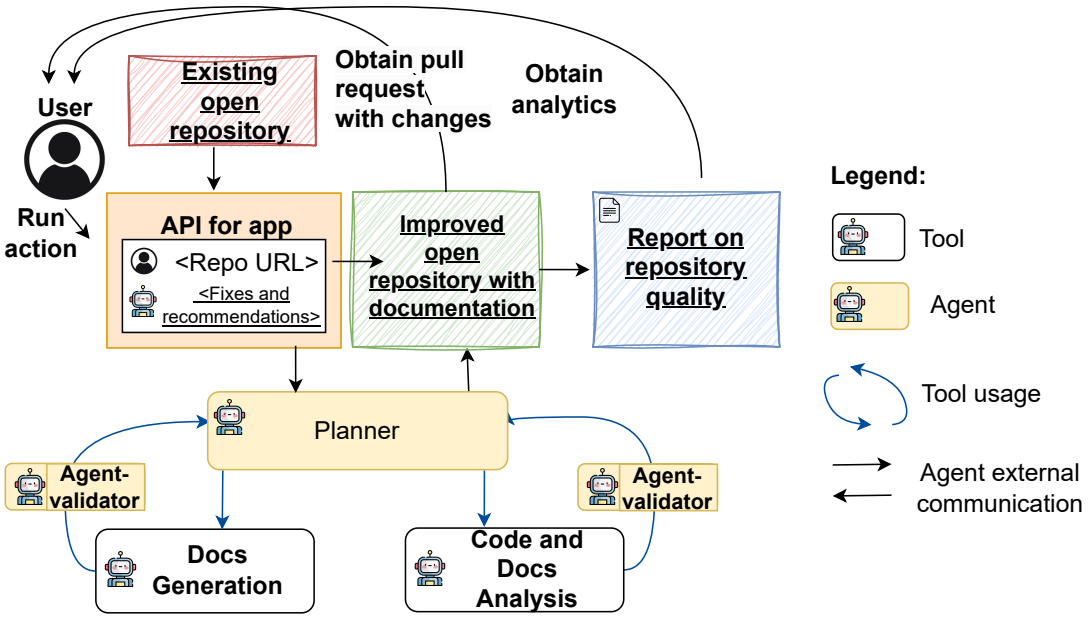

Figure 1. Architecture of OSA with description of main agents and tools.

parameter. The process begins by identifying key repository files, prioritizing those containing core business logic and project descriptions, and incorporating the repository file structure and any existing README when available.

For standard style, the tool extracts core features from the existing README, source files, and metadata, which form the basis for a comprehensive overview. It also generates a *Getting Started* section when example scripts or demonstration files are noted, providing practical guidance for users.

In article mode, the tool produces a summary of the accompanying scientific paper and extracts summaries from key files. These are combined to create an *Overview* outlining the project objectives, a *Content* section describing the main components and their interactions, and an *Algorithms* section explaining the role of implemented methods within the research context. This ensures that the documentation remains scientifically grounded and accessible to a broad audience.

All prompts used for agents and LLM-based tools are provided in the Appendix A. We intentionally avoided using complicated third-party tools (e.g. Cursor) to make the solution more straightforward and interpretable.

3.3. Use cases

OSA is a package that runs through the CLI. It can also be deployed locally in Docker so that by specifying the API key to any LLM, you can interact with the tool via the console. Also, it is available as a Streamlit-based web application. There are several use cases for OSA:

Author of the paper processes a repository with a draft program code for experiments and scientific articles describing this library or algorithm. The article is uploaded as a separate file and used by OSA to generate a better README. OSA creates a fork of the repository and offers a pull request with all the changes. It remains for the developer to look at the suggestions and fix what seems wrong.

Scientific developer uses OSA For more complicated libraries and frameworks, OSA generates CI/CD scripts and docstrings, and improves project structure. It also generates supplementary files such as a contributing guide, pull requests, and issue templates to simplify the involvement of external collaborators.

In addition, OSA provides reports on possible improvements. It also indicates the presence of the main components of the repository: README, license, documentation, usage examples, tests, and a brief description.

4. Experimental studies

The experimental part is focused on the generation of README for repositories. We prepared a small benchmark to compare the quality of OSA (the list is provided in Appendix A.1. It includes the subsets of repositories referred to as *Common* (common-purpose projects) and *Scientific*

Table 1. Metrics for OSA against LLM-based baseline and ReadmeAI. The *General* subset represents general-purpose projects and the *Common* subset represents scientific and educational projects. *All* represents the metrics for the full benchmark. The quality metric is evaluated using the GPT4 model. The best results are highlighted in **bold**.

Repository type	LLM type						
	GPT 4.1			Claude 4 Sonnet		Gemma3 27b	
	LLM baseline	Readme AI	OSA	LLM baseline	OSA	LLM baseline	OSA
Common	0.31	0.72	**0.76**	0.17	**0.79**	0.35	**0.75**
Scientific	0.40	0.77	**0.80**	0.33	**0.82**	0.25	**0.78**
All	0.36	0.75	**0.78**	0.25	**0.81**	0.30	**0.77**

(scientific and educational projects).

We are focused on the quality of README generation since it can evaluated in quite a straightforward way. To measure the quality of the README generation, we used *geval* library with a custom prompt to evaluate the reliability, correctness, and validity of generation results using GPT4 model. The exact prompt is provided in Appendix A.2. There are no repositories from the benchmark that we used for fine-tuning or few-shot results improvement.

As an alternative solution, we consider a simple baseline (direct generation of README via LLM) and open-source ReadmeAI tool (https://github.com/eli64s/readme-ai). Each was initialized via GPT-4.1, Claude 4 Sonnet, and Gemma3 27b model. The comparison of metrics is presented in Table 1. Performance metrics for DeepSeek-V3 and Gemini 2.5 Flash is not included because it is comparable to Gemma3 27B

It can be seen that OSA generates README files with better quality than pure-LLM solutions and widely-used ReadmeAI tool. Also, the quality of OSA generation is less affected by the selection of LLM than ReadmeAI. While results are relatively comparable for Claude 4, the quality of ReadmeAI decreases dramatically when switching to Gamme 3 27b. So, we can use relatively lightweight LLM in OSA without loss in quality. Also, the improvement of metrics for scientific and educational repositories is higher than for general-purpose ones, which confirms the applicability of OSA for scientific code.

5. Conclusions and future works

In the paper, we propose the OSA tool that allows to simplify the development of scientific open-source tools. It combines the advantages of LLMs and existing best practices for open source software development.

In future we would like to add a RAG system to OSA based on our repository with the best practices of open source design. It is assumed that by comparing with the sample, OSA will determine what exactly is missing from the repository being checked. Let's say if he already has a great README, then it won't have to be generated.

We are currently working only on GitHub, because there are more projects there and it is much easier to work with it. However, scientific developments are often uploaded to local versions of GitLab or other GitHub analogues, so it is likely that we will try to be present there as well. Initially, we focused on Python, but we plan to expand the list of supported languages. Also, the more extensive and complex validation outside README generation quality is planned.

Acknowledgments

The Ministry of Economic Development of the Russian Federation (IGK 000000C313925P4C0002), agreement No139-15-2025-010.

Software and Data

OSA is available in repository https://github.com/aimclub/OSA.

Impact Statement

This paper presents work whose goal is to advance the field of scientific open-source and machine learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Benureau, F. C. and Rougier, N. P. Re-run, repeat, reproduce, reuse, replicate: transforming code into scientific contributions. *Frontiers in neuroinformatics*, 11:69, 2018.

Bornmann, L., Haunschild, R., and Mutz, R. Growth rates of modern science: a latent piecewise growth curve approach to model publication numbers from established and new literature databases. *Humanities and Social Sciences Communications*, 8(1):1–15, 2021.

Eliseeva, A., Kovrigin, A., Kholkin, I., Bogomolov, E., and Zharov, Y. Envbench: A benchmark for automated

environment setup. *arXiv preprint arXiv:2503.14443*, 2025.

Färber, M. Analyzing the github repositories of research papers. In *Proceedings of the ACM/IEEE joint conference on digital libraries in 2020*, pp. 491–492, 2020.

He, J., Treude, C., and Lo, D. Llm-based multi-agent systems for software engineering: Literature review, vision and the road ahead. *ACM Transactions on Software Engineering and Methodology*, 2024.

Ivie, P. and Thain, D. Reproducibility in scientific computing. *ACM Computing Surveys (CSUR)*, 51(3):1–36, 2018.

Koreeda, Y., Morishita, T., Imaichi, O., and Sogawa, Y. Larch: Large language model-based automatic readme creation with heuristics. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pp. 5066–5070, 2023.

McKiernan, E. C., Bourne, P. E., Brown, C. T., Buck, S., Kenall, A., Lin, J., McDougall, D., Nosek, B. A., Ram, K., Soderberg, C. K., et al. How open science helps researchers succeed. *elife*, 5:e16800, 2016.

Nejjar, M., Zacharias, L., Stiehle, F., and Weber, I. Llms for science: Usage for code generation and data analysis. *Journal of Software: Evolution and Process*, 37(1):e2723, 2025.

Phan, H. N., Nguyen, T. N., Nguyen, P. X., and Bui, N. D. Hyperagent: Generalist software engineering agents to solve coding tasks at scale. *arXiv preprint arXiv:2409.16299*, 2024.

Seo, M., Baek, J., Lee, S., and Hwang, S. J. Paper2code: Automating code generation from scientific papers in machine learning. *arXiv preprint arXiv:2504.17192*, 2025.

Tang, X., Liu, Y., Cai, Z., Shao, Y., Lu, J., Zhang, Y., Deng, Z., Hu, H., An, K., Huang, R., et al. Ml-bench: Evaluating large language models and agents for machine learning tasks on repository-level code. *arXiv preprint arXiv:2311.09835*, 2023.

Thimbleby, H. Improving science that uses code. *The Computer Journal*, 67(4):1381–1404, 2024.

Trisovic, A., Lau, M. K., Pasquier, T., and Crosas, M. A large-scale study on research code quality and execution. *Scientific Data*, 9(1):60, 2022.

Trofimova, E., Sataev, E., and Jowhari, A. S. Coderefine: A pipeline for enhancing llm-generated code implementations of research papers. *arXiv preprint arXiv:2408.13366*, 2024.

Yang, J., Jimenez, C. E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K. R., and Press, O. Swe-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

A. Technical appendix

A.1. Repositories for validation

The following repositories was used in experimental studies:

https://github.com/wkentaro/labelme, https://github.com/AntonOsika/gpt-engineer, https://github.com/THUDM/ChatGLM-6B, https://github.com/OpenEthan/SMSBoom, https://github.com/lra/mackup, https://github.com/chenfei-wu/TaskMatrix, https://github.com/hacksider/Deep-Live-Cam, https://github.com/google/python-fire, https://github.com/stitionai/devika, https://github.com/Pythagora-io/gpt-pilot, https://github.com/CorentinJ/Real-Time-Voice-Cloning, https://github.com/encode/httpx, https://github.com/lss233/kirara-ai, https://github.com/assafelovic/gpt-researcher, https://github.com/mkdocs/mkdocs, https://github.com/ageitgey/facerecognition, https://github.com/donnemartin/system-design-primer, https://github.com/chatanywhere/GPTAPIfree, https://github.com/Asabeneh/30-Days-Of-Python, https://github.com/kaixindelele/ChatPaper, https://github.com/twintproject/twint, https://github.com/pallets/flask, https://github.com/charlax/professional-programming, https://github.com/ethereum/EIPs, https://github.com/xai-org/grok-1, https://github.com/eriklindernoren/PyTorch-GAN, https://github.com/public-apis/public-apis, https://github.com/Z4nzu/hackingtool, https://github.com/gto76/python-cheatsheet, https://github.com/danielgatis/rembg, https://github.com/bregman-arie/devops-exercises, https://github.com/Vision-CAIR/MiniGPT-4, https://github.com/Jack-Cherish/python-spider, https://github.com/openai/chatgpt-retrieval-plugin, https://github.com/black-forest-labs/flux, https://github.com/sb-ai-lab/EmotiEffLib, https://github.com/sb-ai-lab/Ride, https://github.com/hse-cs/delPezzo, https://github.com/deeppavlov/chatsky, https://github.com/deeppavlov/dialog2graph, https://github.com/corl-team/verl-loras, https://github.com/sb-ai-lab/Eco2AI, https://github.com/AIRI-Institute/GENALM, https://github.com/AIRI-Institute/eco4cast, https://github.com/Vishnu-tppr/Camouflage-AI, https://github.com/tbhvishal/Python-Weather-Info-App, https://github.com/readytensor/rt-repo-assessment, https://github.com/DrpidamenteNanjesha/TagGenerator, https://github.com/stephenombuya/Code-Contribution-Analyzer.

The ”common” or ”scientific” labels were assigned due to the type of the repository.

A.2. Metric evaluation prompt

```
Determine whether the AI-generated Readme file (ACUTAL_OUTPUT)
is better than the original one (EXPECTED_OUTPUT).
ACTUAL_OUTPUT contains two fields: 'readme', which contains generated README itself,
and 'repo_structure' which is json with repository's structure.
Generated README's content must be consistent with provided repository structure.
The ACTUAL_OUTPUT is not neccessary has to be the same as EXPECTED_OUTPUT,
Your goal is to determine which text is better, using provided Evaluations steps.
Readme structure does not matter much as long as it passes evaluation steps.

"Step 1: Does provided structure of the repository addresses README content?",
"Step 2: Does the README provide a clear and accurate overview
    of the repository's purpose?",
"Step 3: Are installation and setup instructions included and easy to follow?",
"Step 4: Are usage examples provided and do they clearly demonstrate functionality?",
"Step 5: Are dependencies or requirements listed appropriately?",
"Step 6: Is the README easy to read, well-structured, and free of confusing language?",
```

A.3. README style prompts

A.3.1. PRE-ANALYSIS PROMPT

```
TASK:
Based on the provided data about the files and the README content,
your task is to identify and return the paths to 3-5 key files that
contain the main business logic or project description.
RULES:
- Return one path per line.
- Choose the most important files that define the project's core logic.
- Exclude files related to tests, configuration,
or assets unless they are central to the business logic.
- Exclude README files.
```

A.3.2. CORE FEATURES PROMPT

```
TASK:
Based on the provided information about the project,
generate a list of core features for the project.
Each core feature should be represented by JSON format following this structure:
{{
    "feature_name": "text",
    "feature_description": "text",
    "is_critical": boolean
}}
RULES:
- The output should be a JSON array.
- Each element in the array should represent one core feature.
- Use double quotes for JSON formatting.
- Be sure to generate multiple core features,
each describing a different aspect of the project.
- The 'feature_name' should describe a key aspect.
- The 'feature_description' should be a detailed
but short explanation of the feature.
```

A.3.3. OVERVIEW PROMPT

```
TASK:
Generate a concise overview of the project by analyzing the provided data.
Your response should be a short paragraph that encapsulates the core use-case,
value proposition.
RULES:
- Focus on the project's core purpose and its value
proposition without mentioning specific technical aspects.
- Avoid technical jargon, code snippets, or any implementation details.
- The overview should be no more than 60 words.
```

A.3.4. GETTING STARTED PROMPT

```
TASK:
You are generating the "Getting Started" section
for the README file of the project above.
Your goal is to help a new user understand how to start
using the project by analyzing the provided example files.
If you find a clear entrypoint or demonstration of how to run or integrate
```

```
the project - use that information to generate a concise, helpful section.
If the example files are empty, contain only boilerplate code
(e.g., import statements or data definitions), or do not provide any meaningful
insight on how to use the project | return `null` in JSON instead of a section.
RULES:
- Be concise and beginner-friendly.
- Use markdown formatting with code blocks if needed.
- Prefer actual code found in the examples over assumptions.
- DO NOT make up usage instructions | rely only on provided content.
- If unsure or examples are not helpful, return "getting_started": null
- Don't add ## Getting Started at the beginning
```

A.4. Article style prompts

A.4.1. FILES SUMMARY PROMPT

```
TASK:
Analyze the provided code repository. Your task is to summarize the following:
Purpose: The overall goal and intended function of the codebase.
Architecture: The structure of the codebase and its key components.
Functionality: What the code achieves and the main algorithms or approaches
it implements. Connection to the context: How the code reflects or supports
the methodology, key ideas, or results described in the article. Ensure
your summary is clear, concise, and omits technical implementation details.
Focus on high-level insights that help understand the codebase.
RULES:
- Avoid phrases like "This file", "The file", or "This code".
- Begin with a verb or noun.
- Do not include quotes, code snippets, bullets, or lists.
- Limit the response to 200-250 words.
```

A.4.2. ARTICLE SUMMARY PROMPT

```
TASK:
Analyze the given text, which may be a technical report or article.
Your task is to extract the following,
basing your response solely on the context provided:

Main topic: Identify the central subject of the document.
Key ideas: Highlight the primary concepts or arguments presented.
Methodology: Describe the methods or approaches used.
Working steps: Outline the significant actions or processes involved.
Results: Summarize the outcomes or findings.
Ensure your response is precise, uses concise language, and avoids
any additional information not present in the text.
RULES:
- Start with a strong, attention-grabbing statement.
- Avoid phrases like "This PDF" or "The document".
- Exclude quotes, code snippets, bullets, or lists.
- Limit the response to 200-250 words.
```

A.4.3. OVERVIEW PROMPT

```
TASK:
Outline the repository's purpose and objectives based on the following context.
Emphasize main functionalities and goals without technical jargon.
```

```
RULES:
- Begin with a clear statement capturing the project's essence.
- Use 150-200 words.
- Avoid phrases like "This project" or "The repository".
```

A.4.4. CONTENT PROMPT

```
TASK:
Describe the repository's components|including databases, models,
and other relevant parts|and explain how they interrelate to support
the project's functionality.
RULES:
- Emphasize each component's role in the overall project.
- Focus on high-level concepts, avoiding technical details.
- Don't put quotes around or enclose any code.
```

A.4.5. ALGORITHMS PROMPT

```
TASK:
Detail the algorithms used in the codebase, explaining their functions.
RULES:
- Describe each algorithm's role without technical implementation details.
- Use clear, accessible language.
```

A.5. Scheduler prompt

```
TASK:
Analyze the repository structure and README content to determine
the appropriate settings for the following options.
Generate a JSON report following this exact structure:
{{
  "generate_report": boolean,
  "translate_dirs": boolean,
  "generate_docstring": boolean,
  "ensure_license": str or None,
  "community_docs": boolean,
  "generate_readme": boolean,
  "organize": boolean,
}}

RULES:

- Return only a valid JSON object following exactly the structure above.
- "generate_report": true if an additional user report would be helpful,
false otherwise.
- "translate_dirs": true if directory and file names
need to be translated to English.
- "generate_docstring": true if significant Python code lacks docstrings.
- "ensure_license": one of "bsd-3", "mit", "ap2" if a license
file should be generated, or None if license is alreade presented.
- "community_docs": true if community files like CODE_OF_CONDUCT.md or
PULL_REQUEST_TEMPLATE.md should be created.
- "generate_readme": true if README is missing or of low quality,
false if README is clear and sufficient.
- "organize": true if 'tests' or 'examples' directories are missing and should be added.
```

```
- Do not add any explanations, comments or extra text.
- Use lowercase true/false and null exactly as JSON booleans/null.
- Your response must be valid JSON parsable by standard parsers.
```