An Early Variant Approach: The Extract, Transform and Execute (ETE) Agent For Research Software Reuse

Carlos Utrilla Guerrero[1,2]

[1]Universidad Politecnica de Madrid, Madrid, Spain

[2]Delft University of Technology, Delft, The Netherlands

Abstract

Automating the interpretation and execution of Research Software (RS) installation procedures is key to minimizing researcher workload while maximizing reusability. This paper presents ETE, a (work-in-progress) Large Language Model (LLM)-based agent designed to automatically support RS reuse activities from documentation. This early variant agent integrates multiple specialized tools and basic reasoning strategies—such as task decomposition and tool selection— to perform distinct tasks, ranging from extracting install-related instructions from README files to transforming them into a format that can be executed by machines. This work-in-progress introduces the conceptual approach, problem formulation, architectural design, and a minimal Python prototype exploring the potential suitability of using LLM-powered agents to facilitate RS reuse. While formal evaluation remains future work, this conceptual approach lays the foundation for a family of Artificial Intelligent (AI) agents that aim to bridge the gap between human-generated documentation and automated reuse, a significant step toward accelerating scientific discovery.

Keywords

Research Software, Reuse, Intelligent Assistant, Natural Language Processing, Artificial Intelligence

1. Introduction

It is widely recognized that documentation accompanying Research Software (RS) [1]—such as source code comments and/or README files—contains valuable human-generated information. This information (either explains how RS operates or how to install) can be exploited to build research services and infrastructures with the goal of facilitating software reusability, accelerating researcher productivity, and reducing associated costs [2].

README files commonly detail step-by-step install instructions as part of well-established methods such as Package Manager, Container, Source Code and/or Setup scripts that aim to facilitate the installation and execution of RS with minimal friction [3, 4]. However, encapsulating RS using portable environments-Docker containers or python package) might impedes the understandability, reuse and interpretation of the individual components contained therein, or are often outdated, or incomplete, requiring researchers to visually inspect instructions and follow human-generated procedures [5]. These unstructured narratives presents a major obstacle [6] for interpreting and executing instructions by both humans and machines, ultimately limiting the automation of reuse from documentation across RS [7].

SEPLN 2025: 41st International Conference of the Spanish Society for Natural Language Processing, Zaragoza, Spain, 23-26 September 2025.

*Corresponding author.

carlos.utrilla.guerrero@alumnos.upm.es (C. U. Guerrero)

https://carlosug.github.io/ (C. U. Guerrero)

0000-0002-9994-1462 (C. U. Guerrero)

To address the challenges, we present our Large Language Model (LLM)-based first variant agent designed to automatically assist researchers in reuse activities, which are to install a given RS documentation (e.g., GitHub Repository URL), as well as to execute it into a virtual environment. Here we tackle this challenge by exploring a multidimensional approach encompassing these following tasks: (1) **extracting** install-related instructions from README files and (2) **transforming** them into a format that can be (3) **executed** by machines at a minimum cost. In this paper, we introduce ETE agent, an early, minimal implementation of this conceptual framework, focused specifically on dealing with installation-relevant information. For demonstration purposes, we illustrate the framework's workflow as well as learning strategies using real-world GitHub-hosted repositories. Our exploration with ETE agent demonstrates the serious possibility of novel efficient learning strategies, powered by tool usage and reasoning capabilities for reuse-tasks. Through further exploration, we present how ETE agent deals each stage iteratively, some stages involving structured data transformation via tool-calling, and others decomposing tasks into subtasks to break complexity down into primitive actions for planning purposes. We also identify several challenges specific to plan tasks, such as gathering task-relevant information from README and alternative files. While the effectiveness and suitability of our approach still require rigorous empirical evaluation, it reveals key challenges and new opportunities in integrating LLMs to RS reuse problems.

The remainder of this paper is organized as follows: Section 2 introduces prior research literature, providing a foundational context for our work. Section 3.1 describes the practical problem we are interested in addressing and potential solution based on state-of-art. In Section 3.2 we present our workflow for ETE agent, describing its design details and all the stages involved, including extraction, transformation and execution. Section 3.3 focuses on the technical implementation of our workflow, Section 4 illustrates the practical example to be carry out by the agent in the demo, presenting its limitations and our future lines of work in Section 5. Finally, Section 6 concludes the paper.

2. Related Work

Existing related work can be grouped under two major topics: RS reuse and LLMs as Scientific Agents:

- **RS Reuse:** the benefits of reusing software (e.g., reducing duplication of effort) —are broadly acknowledged since early days in the software engineering field [8]. In spite of its promise, RS reuse has not become standard practice in RS development yet [9]. In light of this persistent challenge, the RS community has renewed its interest in understanding the barriers to effective RS reuse and developing strategies to overcome them. Among others, research initiatives such as Codemeta, SoFAIR, and EVERSE have emerged recently to enhance research software reuse by standardizing metadata, automating lifecycle management, and promoting software quality.
- **LLMs as Scientific Agents:** there is a widespread desire in the scientific community to address the reusability challenge with a fully automatic approach; Recent work has initiated to explore the power of LLM-based IAs in software engineering tasks [10] and scientific discovery [11]. Notable among these are typically repository-level tasks [12] that aim to exploit the vast amount of code openly available in repositories. Several research projects explore automated solutions to generate unit test [13], bugs [14] and issue [15] reports with summarization techniques for README files [16] as well as checking conflicts and libraries vulnerabilities [17]. However, a key research challenge remain insufficiently understood, which is how suitable are LLM-powered agents to assist RS reuse from documentation.

3. Automating RS Reuse via Documentation

This section presents our conceptual approach for the automatic reuse of RS based on available documentation. We begin by defining the specific problem our work aims to address, along with a potential solution (Section 3.1). Next, we provide an overview of the proposed conceptual approach 3.2, followed by a detailed description of the core components and implementation of the framework (Sections 3 and Section 3.3, respectively). The overall process is depicted in Figure 1.

3.1. Problem Statement

We define the machine problem of automatic reuse of Research Software (RS) as follows: given access to openly available RS documentation (e.g., a URL to a public Git repository), what actions should a system—such as an artificial agent—perform to automatically convert human-generated installation instructions into actionable, machine-executable commands, and execute them within a virtual environment?

Table 1
Problems that machines need to address to accelerate scientific discovery through reuse of RS from documentation (e.g., README files): Brief Explanations and representative Prior Work.

id	Problem	Explanation	Prior Work
P1	Unstructured documentation	Machines lack a formal representation or efficient algorithm to extract install, configure, and execute commands reliably.	[18]
P2	Idiosyncrasies in install steps	If one install-step in a procedure is not machinery-consumable, then these exceptions must be handled, and automation is harder or impossible.	[4]
P3	Complex documentation	This complexity requires substantial additional capabilities for machines (managing methods, dependencies, configurations consistency across systems).	[19]
P4	Lack instruct-to-action mapping	No structured representation linking natural-language instructions to concise machine operations, preventing robust reuse workflow.	[20]

Unlike prior work that addresses isolated challenges or narrow tasks, our objective is to explore how to develop a unified solution covering the full range of problems summarized in Table 1. A robust solution to enable RS reuse at scale would need to proceed as follows: to overcome the lack of standardization in RS documentation, including inconsistent machinery install-instructions (P1 and P2), an agent first would need to extract all software metadata and alternative installation methods described in the README files (and other files), then transforms each method's sequence of steps into a structured JSON format. To tackle P3 (e.g., automation in managing environments and configurations) and P4 (e.g., automated solutions for execution of RS), the community utilises continuous and development solutions; however these are relying on permission and might not be fully automated. Therefore, an agent would need to provide detailed thinking process (e.g., adopting the approach introduced in our previous work [3]: the *PlanStep* in which agent first breaks down a complex install methods found in a README file such "from source" into several subtasks, and then plan for each subtask in a fixed, sequential order) before generating the two targeted outputs: i) an isolated environment (e.g., via Docker or virtual environments) and another to configure and install the RS within that environment.

To our knowledge, this is the first integrated Extraction–Transformation–Execution (ETE) conceptual framework for automated RS reuse using LLM-based agents.

3.2. Proposed ETE Agent: An Overview

In this section, we introduce the ETE agent, a high-level, LLM-powered agent designed to automate the reuse of Research Software (RS) from documentation, specifically the installation and execution instructions from open-source RS repositories. We briefly describe our proposed approach, covering all the stages involved in our workflow. We also describe the agent's potential reasoning capabilities and how ETE interacts autonomously with a suite of discrete, function-based tools[1] to accomplish each phase of the workflow. These stages are briefly summarized in the following subsections and are depicted in Figure 1.

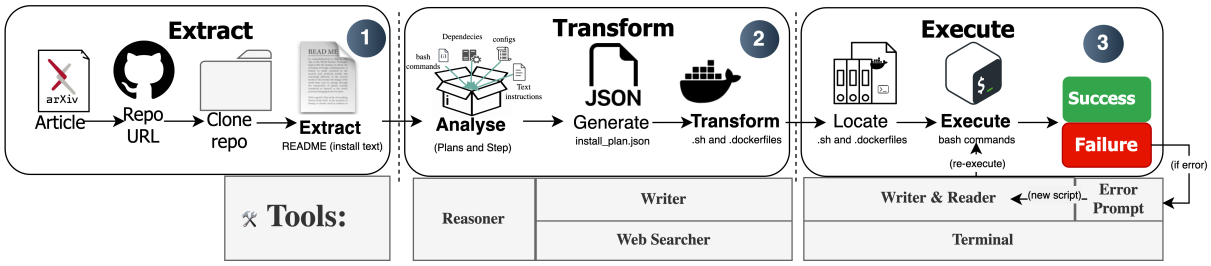

Figure 1: A high-level overview of ETE Agent Workflow - An illustration of how ETE agent make progress from initial extraction stage (1) through transformation of these outputs into machine executable files (2), and to subsequently the execution of these files into a local environment (3). Python-style (in grey boxes) functions exposed as tools for the agent to use.

3.2.1. Stage 1: Extract

Provided that a RS path (URL of a git repository) is given, the ETE agent first retrieves all install-specific information from the README file—gather fields covering the set of statements that determine install methods, procedural steps, instructions, and other key characteristics required for configuring, setting and executing a RS. Additionally, the agent clones the repository and extracts other project-specific metadata relevant to our context (e.g., Name, Programming Language, Executable and Usage examples). With the environment all set, ETE agent sends a query to the a LLM, which generates a JSON output (see Figure 3) following the CodeMeta schema as shown in Figure 2. The corresponding LLM-generated output is then validated and constrained into a predefined JSON schema. We chose a widely used standards such as CodeMeta properties to represent the install-specific categories from README files using Pydantic Library as its the most popular Python library for performing data validations, ensuring strict adherence to the expected format.

3.2.2. Stage 2: Transform

Once Extract stage (Stage 1) is completed, the initial ETE-generated output (e.g., Installation Instructions JSON file) is fed into our Stage 2, with the goal of transforming it into machine-executable files ready to be executed by a machine. Specifically, the agent must:

1. Compare and analyse comprehensively the installation-relevant information from Installation Instructions JSON file and any other supplementary with information about dependencies and environmental requirements, among others: .yaml, .config and pyproject.toml.
2. Generate an actionable plan, step-by-step installation plan that maps each step to its prerequisites, orderly installation steps with instructions, dependency information and compatible Operating System (see Figure 5).

[1]Mimicking the steps that a human tasked at the same activity would need to perform such as exploring documentation, setting up the environment, following instructions based on operating systems, and executing commands in a terminal.

JSON Schema for Installation Instructions (Summarization Data validation)

```python
class ReadmeAnalysisContent(BaseModel):
    """Model for parsed README."""
methods: List[str] = Field( default_factory=list ,
description="List of installation methods"
installation_instructions_per_method: List[InstallStep] =
Field( default_factory=list ,
description="List of installation and usage commands with metadata")
important_links: List[str] = Field(
 default_factory=list , description="List of relevant documentation links")
```

Figure 2: Overview of the JSON schema JSON response from ETE agent in **Stage (1) - Extract**. ETE agent prompts a LLM model to retrieve and organise install-specific categories: (i) installation methods, (ii) installation steps , (iii) requirementson operating systems, dependencies and configuration, (iv) invocation commands for each step, and (v) external links available in the README files.

3. Transform the plan into executable commands and output them as a Dockerfile and shell script (install.sh) suitable for automated execution (depicted in Figure 6 and Figure 7, respectively).

Installation Instructions (Summarization response in JSON)


```json
{ "repo": "Darwin Godel Machine (DFM)",
  "methods": ["Docker", "Package Manager"],
  "prerequisites": ["Docker", "Python", "pip"],
  "operating_system": "Linux",
  "installation_instructions_per_method": [
    { "method": "Docker", "order": 1,
      "instruction": "Verify Docker configuration",
      "commands": ["docker run hello-world"]
    },(...)
    {  "method": "Package Manager", "order": 1,
    (...)},
  "environment": ["OPENAI_API_KEY",
  "ANTHROPIC_API_KEY"],
  "important_links": [
    "https://github.com/jennyzzt/dgm/blob/main/LICENSE",
    "https://sakana.ai/dgm/"]}
```


Figure 3: Example of an ETE-generated response install-related extraction from DFM repository in **Stage (1) - Extract**. This JSON file contains README install-specific information organised in: two distinct plans, precise sequence of install steps with commands, list of dependences (and pre-requirements (setup API keys) as well as important links.

Installation Plan (Summarization reasoning response in JSON)

```
"reasoning": { "analysis": "The provided installation requirements
include multiple methods,
including installation from GitHub, pip, and Docker.",
"method_choice": "Based on the provided requirements,
the preferred installation method is 'source'
as it allows for the most flexibility and customization.",
"next_steps": "After installation,
the user can run the software using
the provided invocation commands
and follow the usage examples."}
```

Figure 4: Example of an ETE-generated Reasoning response, as part of Installation Plan for **Stage (2) - Transform**. This JSON reasoning section explains how the installation plan was derived. The analysis field notes that the software can be installed via GitHub, pip, or Docker. The method_choice field justifies selecting "source" as the preferred method due to its flexibility. Finally, next_steps suggests running the software with the provided commands.

Installation Plan (Summarization Installation Plan response in JSON)

```
"steps": [
    {"order": 1,
      "method": "source",
      "description": "Clone the GitHub repository",
      "command": [
        "git clone https://github.com/KnowledgeCaptureAndDiscovery/somef.git"
      ]}, (...)  (...)
      "requirements": [
        "A valid GitHub authentication token (if used)"
      ]}],
  "operating_systems": [
    "Linux",
    "macOS",
    "Windows"]}
```

Figure 5: Example of an ETE-generated Installation Plan response for **Stage (2) - Transform**. This example outlines the steps to setup the RS (e.g.,SOMEF RS). Each step includes an order, a method such as "source" indicating the most suitable option, a description "Clone the GitHub repository", and command lines. Note that total fields gathered by the agent are omitted.

A key feature of the ETE agent is exploring its reasoning process. This intermediate "thought trace" aims to aid in the generation of Installation Plan, enabling the ETE agent to reason why particular install approaches were chosen over alternatives (see Figure 4).

3.2.3. Stage 3: Execute

Building on the outputs generated in Stage 2, the third stage of the ETE agent is responsible for executing the resulting files. Thus, this stage focuses on two primary objectives:

- **Locating the files:** Verifying the presence of all required outputs, including the Dockerfile, install.sh script, and associated JSON files.
- **Executing the files:** Running the commands specified in the executable artifacts to automate the environment setup and RS installation process.

During this stage, the agent interacts with an LLM by sending structured prompts, processing responses, and executing tool calls dynamically. If all objectives are successfully met, the ETE agent returns a completion report indicating a successful installation. Otherwise, the agent invokes available

install_and_run.sh

```
#!/bin/bash
set -e
echo "Creating and activating virtual environment..."
python3 -m venv venv
source venv/bin/activate
echo "Installing main dependencies..."
pip install -r requirements.txt
echo "Installing system dependencies..."
sudo apt-get install graphviz graphviz-dev
(...)
echo "Checking out specific commit..."
cd SWE-bench
git checkout dc4c087c2b9e4cefebf2e3d201d27e36
echo "Installing SWE-bench in development mode..."
pip install -e .
echo "Preparing Polyglot dataset..."
python polyglot/prepare_polyglot_dataset.py
```

Figure 6: Illustrative example ETE-generated shell script from DGM Github repository.

Dockerfile

```
FROM ubuntu:latest
# Set working directory
WORKDIR /app
# Copy installation script
COPY install.sh .
# Make script executable and run it
RUN chmod +x install.sh && ./install.sh
# Default command
CMD ["bash"]
```

Figure 7: Illustrative example ETE-generated Dockerfile from DGM Github repository.

tools (e.g., writer tool, web searcher or terminal tool), formulates a hypothesis to guide error resolution, generates a revised prompt for repairing the issue, and writes and re-executes a new installation script (see output generated by ETE agent in Figure 8).

3.3. Implementation Details

This section describes the formulation of prompting approaches as well as the models and tool usage taken into account for our ETE agent exploration.

3.3.1. Prompting Approaches

Our ETE agent employs a widely-used guidelines and standards to compliance with protocols of the GROQ reasoning services for prompt comprising: agent role (e.g., defines primary task of the agent), tasks (e.g., outlines specific objectives), examples (if applicable, simple examples in the prompt to enact the expected format) and available tools (if applicable, the functions that are to be used by the agent). Additionally, we explore different settings, depending on the stages (see Figure 9):

1. **Extraction**: we implement few-shot prompting with concrete examples.
2. **Transform**: we employ zero-shot instructions with constraints for the reasoning approach.

Step 3 – Installation Error Diagnosed

<THINK>

The installation process failed due to a missing requirements.txt. According to the protocol, when an installation error occurs, the next action is to generate a search prompt based on the error details (STEP 4). If Error Occurs, the next logical step is to invoke the tool formulate_error_search_prompt. **Tool Call:**

```
{"name": "execute_tool_in_terminal",
  "arguments": {
    "name": "formulate_error_search_prompt",
    "arguments": {
"error_file": "execution_agent_workspace/outputs/installation_errors.log"}}}
```

<ACTION>

```
-> Executing: run_install_script
Debug: Executing tool run_install_script with arguments: {'script_path': '
    execution_agent_workspace/outputs/install.sh'}
Installation failed with exit code 1
ERROR: Could not open requirements file: No such file or directory: 'requirements.txt'
```

Figure 8: Illustrative example ETE generated response - **Stage (3) - Execute** & Tool Usage (Web Searcher and Terminal.)

System prompt (a)
Role: You are an AI installation assistant.
Task: Generate an installation script that will run INSIDE a container for user_content. The response MUST contain:
Rule 1: A <think> section with your REASONING
Rule 2: A <script> section with ONLY valid EXECUTABLE commands

System prompt (b)
Role: You are an AI installation assistant:
Task: Execute the script:
Rule 1: always THINK before you ACT:
THINK: Once script is found (stage 2), run the script (stage 3)
ACT: Run the installation script
TOOL_AVAILABLE:"tool_name": "run_install_script","arguments"

Figure 9: (a) Prompt to generate a script (install.sh) with reasoning, and (b) Prompt to execute a script (install.sh) with reasoning (tools).

3. **Execute**: Each step in Stage 3 follows a *ReAct*-oriented approach as proposed by Yao et al., 2022, where LLMs generate both reasoning traces and actions.

3.3.2. Models

We chose the DeepSeek-R1-Distill-Llama-70B model[2] via the GROQ API endpoints with a temperature of 0.3 to balance creativity and accuracy. DeepSeek-R1 family models tend to work well when reasoning capabilities and agent tool approaches are required in the agents [20].

3.3.3. Tool Usage

To support ETE agent undertaken its tasks, we equip with a suite of tools implemented as executable Python functions. These tools aims to enhance the ETE agent's capabilities across the ETE workflow entirely. At the time of writing this paper, however, the following set of tools is only supported in stage 2 and 3:

- **Terminal**: It executes shell commands within a Linux environment.
- **Reasoner**: It reads initial installation plans, selects an optimal path with ordered steps, and outputs reasoning traces in JSON format.

[2]https://console.groq.com/docs/model/deepseek-r1-distill-llama-70b

- **Reader and Writer**: Handles file operations in bash. Given a file path and content, the tools either read from or write to the target file.
- **Web Searcher**: Issues web queries to collect best practices related to container configuration and integrates relevant results into installation recommendations during validation.

4. ETE Agent Demonstration

For our demonstration, we will present ETE agent from a conceptual perspective and demonstrate its potential suitability for assisting in reuse-oriented tasks, as previously discussed. The proposed ETE agent is publicly available at https://github.com/carlosug/agent.rse.

5. Limitations and Future Work

In this paper we introduce the first variant of the ETE agent as a conceptual framework, still in its early stage of exploration and development. We acknowledge that our work has limitations summarised as follows:

- **RS Documentation Complexity** Our limited solution performs well on well-documented, widely used research software (RS), particularly when relying exclusively on README files. The suitability of our approach to non-standardised RS documentation beyond README files remains insufficiently studied. To address this, we plan to mine RS repositories to analyze the diversity of installation methods. This will characterise how RS complexity affects agent performance at scale.
- **Need for Systematic Evaluation** The ETE agent leverages emerging LLM capabilities such as tool calling, though its robustness in long-term planning is still unproven. A key limitation is the lack of benchmarks for RS reuse-tasks. Future work will focus on building a more sophisticated prototype and developing methodologies for accurately benchmarking RS reuse in realistic scenarios.

6. Conclusion

In this work, we presented the ETE agent, our first variant agent designed to automate the reuse of research software (RS) by interpreting installation and execution instructions from open-source RS repositories. The agent leverages LLMs, alongside specialized tools and reasoning capabilities, to extract install-related instructions from README files, and convert them into machine-executable formats with minimal manual intervention. This work marks a first step toward a family of ETE-derived agents capable of autonomously supporting RS reuse—a key challenge in accelerating scientific discovery.

Acknowledgments

This work is supported by the Ontology Engineering Group (OEG) under the PhD in Artificial Intelligence Program with Universidad Politécnica de Madrid, and through the support of the research team supervisor Dr. Daniel Garijo. The authors would also like to acknowledge TU Delft University.

Declaration on Generative AI

During the preparation of this work, the author(s) used ChatGPT and in order to: Grammar and spelling check.

References

[1] M. Barker, N. P. Chue Hong, D. Katz, et al., Introducing the FAIR Principles for research software, Scientific Data 9 (2022). doi:10.1038/s41597-022-01710-x.

[2] P. Abate, R. Di Cosmo, L. Gesbert, F. Le Fessant, R. Treinen, S. Zacchiroli, Mining component repositories for installability issues, in: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, IEEE, 2015, pp. 24–33.

[3] C. Utrilla Guerrero, O. Corcho, D. Garijo, Automated Extraction of Research Software Installation Instructions from README Files: An Initial Analysis, Lecture Notes in Computer Science 14770 LNAI (2024) 114–133. doi:10.1007/978-3-031-65794-8_8.

[4] H. Gao, C. Treude, M. Zahedi, Adapting installation instructions in rapidly evolving software ecosystems, IEEE Transactions on Software Engineering (2025).

[5] S. Hermann, J. Fehr, Documenting research software in engineering science, Scientific Reports 12 (2022). doi:10.1038/s41598-022-10376-9.

[6] L. Salerno, C. Treude, P. Thongtatunam, Open source software development tool installation: Challenges and strategies for novice developers, arXiv preprint arXiv:2404.14637 (2024).

[7] S. Yuan, K. Song, J. Chen, X. Tan, Y. Shen, R. Kan, D. Li, D. Yang, Easytool: Enhancing llm-based agents with concise tool instruction, arXiv preprint arXiv:2401.06201 (2024).

[8] P. Naur, B. Randell, Software engineering: Report on a conference by the nato science commitee, NATO Scientific Affairs Division, Brüssel, (1968).

[9] C. Goodwin, S. Woolley, Barriers to device longevity and reuse: A vintage device empirical study, Journal of Systems and Software 211 (2024) 111991.

[10] M. Pezzè, S. Abrahão, B. Penzenstadler, D. Poshyvanyk, A. Roychoudhury, T. Yue, A 2030 roadmap for software engineering, ACM Transactions on Software Engineering and Methodology (2025).

[11] A. Ghafarollahi, M. J. Buehler, Sciagents: Automating scientific discovery through multi-agent intelligent graph reasoning, arXiv preprint arXiv:2409.05556 (2024).

[12] R. Bairi, A. Sonwane, A. Kanade, V. D. C, A. Iyer, S. Parthasarathy, S. Rajamani, B. Ashok, S. Shet, CodePlan: Repository-level Coding using LLMs and Planning, arXiv preprint arXiv:2309.12499 (2023).

[13] B. Li, C. Vendome, M. Linares-Vasquez, D. Poshyvanyk, N. A. Kraft, Automatically Documenting Unit Test Cases, in: Proceedings of IEEE International Conference on Software Testing, Verification and Validation, ICST 2016 (2016) 341–352. doi:10.1109/ICST.2016.30.

[14] S. Rastkar, G. C. Murphy, G. Murray, Automatic summarization of bug reports, IEEE Transactions on Software Engineering 40 (2014) 366–380. doi:10.1109/TSE.2013.2297712.

[15] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, K. Vijay-Shanker, Towards automatically generating summary comments for Java methods, in: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (2010) 43–52. doi:10.1145/1858996.1859006.

[16] H. Gao, C. Treude, M. Zahedi, Evaluating Transfer Learning for Simplifying GitHub READMEs, in: Proceedings of the 31st ACM Joint European Software Engineering Conference, 2023. doi:10.1145/3611643.3616291.

[17] H. O. Delicheh, A. Decan, T. Mens, Quantifying Security Issues in Reusable JavaScript Actions in GitHub Workflows, in: Proceedings of IEEE/ACM 21st International Conference on Mining Software Repositories, MSR 2024 (2024) 692–703. doi:10.1145/3643991.3644899.

[18] A. Mao, D. Garijo, S. Fakhraei, Somef: A framework for capturing scientific software metadata from its documentation, in: 2019 IEEE International Conference on Big Data (Big Data), 2019, pp. 3032–3037. doi:10.1109/BigData47090.2019.9006447.

[19] I. Bouzenia, P. Devanbu, M. Pradel, Repairagent: An autonomous, llm-based agent for program repair, arXiv preprint arXiv:2403.17134 (2024).

[20] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, et al., Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, arXiv preprint arXiv:2501.12948 (2025).