Universiteit
Leiden
The Netherlands

Bachelor Computer Science

Improving AI-Based Repository Analysis
for Multi-Language Feedback

Hamzeh Akkad

First supervisor:
Prof. dr. ir. J.M.W. Visser

Second supervisor:
M.K.K. Kalavainen MSc

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

13/07/2026

Abstract

Background: AI-based repository feedback tools can support software engineering education by automatically generating feedback on project structure, source code, build configuration, and development practices. However, existing tools can be limited by language-specific assumptions and by processing repository contents that are not useful for feedback generation.

Aim: This thesis investigates whether improved repository content selection and multi-language support can improve the quality and generality of feedback generated by an AI-based repository analysis tool.

Method: An existing repository feedback tool, *AI repo feedback*, was analysed and extended. The improved implementation adds support for PHP and TypeScript, introduces language-specific analysis functions, and filters irrelevant files and directories before sending repository contents to the AI model. The evaluation compares the original and modified implementations through repository analysis and a user study with computer science students, focusing on language support, repository content selection, and the perceived usefulness of the generated feedback.

Results: The improved implementation supports more programming languages than the original version by adding PHP and TypeScript support. It also addresses a scalability problem observed on larger repositories such as Episciences, where the original approach could pass too much repository content to the AI model and fail during analysis. In addition, selecting the largest files without stronger filtering could result in generated or otherwise irrelevant files being chosen for analysis. The improved implementation reduces this problem by excluding dependency directories, generated output, and unusually large files before selecting source files for analysis. User evaluations indicate that these modifications improve the applicability of the tool, although the perceived quality of the generated feedback remains largely unchanged.

Conclusion: The results show that improved repository content selection and multi-language support make the repository feedback tool more general and scalable by enabling the analysis of a wider range of repositories while reducing unnecessary AI input. However, the perceived quality of the generated feedback remains largely determined by the underlying language model rather than by the repository selection strategy alone.

Contents

Chapter 1

Introduction

This chapter sets up the problem addressed in this thesis. It introduces repository feedback in software engineering, explains the limitations of an existing AI-based repository feedback tool, and defines the research question around multi-language support and repository content selection.

1.1 Context

Software repositories contain important information about a software project, such as its source code, directory structure, build configuration, documentation, and development history. In software engineering education, reviewing such repositories is a common way to assess student projects and provide feedback. However, manually reviewing repositories can be time-consuming, especially when many projects need to be evaluated.

Recent developments in AI-based code analysis and software repository mining have created opportunities to automate parts of this process. Repository mining focuses on extracting useful information from version control systems and project artifacts to support software quality and development practices [4]. In addition, machine learning and deep learning techniques have been applied to source code analysis tasks such as code similarity detection, automated review, and quality assessment [10, 6].

More recently, Large Language Models (LLMs) have been used for repository-level software analysis. These models can generate natural-language feedback based on repository contents, making them useful for educational settings where students benefit from explanations rather than only receiving rule-based warnings. However, LLM-based repository analysis also introduces challenges, such as token costs, limited context windows, and the risk of hallucinated feedback [3].

This thesis uses an existing AI-based repository analysis tool called *AI repo feedback*, developed by Diomidis Spinellis, as the starting point for the research [9]. The tool is analysed, modified, and evaluated in order to investigate how its language support and repository content selection can be improved.

1.2 Problem Statement

The AI repo feedback tool analyses a GitHub repository and generates feedback about repository contents, project architecture, source code quality, build configuration, and configuration management. While the existing tool demonstrates the potential of AI-assisted repository feedback, it has two important limitations.

First, the tool is purely designed around Java repositories. This limits its generality, because many software projects are written in other languages such as PHP or TypeScript. When a repository does not follow the assumptions of the Java ecosystem, the generated feedback can become incomplete or less relevant.

Second, the original tool processes repository contents in a relatively broad way. Large repositories often contain files that are not useful for feedback generation, such as third-party dependencies, generated files, build artifacts, and framework output directories. Including such files increases the amount of input sent

to the AI model without necessarily improving the quality of the feedback. In larger repositories, this can reduce scalability and may lead to quota or context-size limitations.

1.3 Research Question

This thesis investigates whether the repository feedback tool can be improved by extending its language support and by making better decisions about which repository contents should be analysed.

The following research question guides this research:

RQ To what extent can improved repository content selection and multi-language support improve the quality and generality of feedback generated by an AI-based repository analysis tool?

In this thesis, repository content selection refers to the process of deciding which files and directories should be included in the analysis and which should be excluded because they are unlikely to contribute meaningful information.

1.4 Objectives

The main objective of this research is to improve the generality and scalability of an existing AI-based repository feedback tool.

This objective is divided into the following sub-goals:

- Analyse the original repository feedback tool and identify its main limitations.
- Extend the tool with support for additional programming languages.
- Improve repository content selection by filtering irrelevant files and directories.
- Reduce unnecessary input sent to the AI model.
- Evaluate whether the modified tool provides more useful and general feedback than the original implementation.

1.5 Overview of the Thesis

In Chapter 2, the background and related work are discussed. This includes repository analysis, AI-based code analysis, repository-level LLM systems, and the importance of selecting useful repository contents before sending them to an AI model.

Chapter 3 describes the research approach used in this thesis. The chapter explains how the original tool was examined, how the prototype was developed, and how the improved tool will be evaluated.

Chapter 4 presents the design and implementation of the improved tool. It describes the changes made to support multiple programming languages and to reduce unnecessary AI input through repository filtering and file selection.

Chapter 5 presents the evaluation of the prototype. This chapter compares the original and modified versions of the tool and discusses the results of testing the tool on different repositories.

Chapter 6 discusses the meaning of the findings, the limitations of the research, and the extent to which the research question can be answered.

Finally, Chapter 7 summarises the thesis and gives suggestions for future work.

Chapter 2

Background and Related Work

This chapter places the project in the context of existing work. It discusses repository mining, AI-based software analysis, and repository-level LLM analysis, then connects this work to the practical problem of selecting useful repository contents before sending them to an AI model.

2.1 Repository Feedback and Repository Mining

Software repositories contain more than only source code. They also include build files, documentation, tests, dependency definitions, configuration files, and version control history. Together, these artifacts give insight into the structure of a software project and the way it is developed. In software engineering education, this information can be used to assess student projects and provide feedback on aspects such as project organisation, maintainability, testing, and configuration management.

Mining software repositories is the research field concerned with extracting useful information from software repositories. Hassan describes this field as a way to support understanding of software development processes through the analysis of repository data [4]. This is relevant to this thesis because the repository feedback tool also extracts information from project artifacts before generating feedback.

Traditional repository analysis tools often rely on predefined rules or static analysis techniques. These tools can be useful for detecting specific problems, such as missing files, style violations, or certain code smells. However, they are usually limited to the checks that were explicitly implemented. As a result, they may be less suitable for producing broader explanatory feedback about project structure, architecture, and maintainability.

2.2 AI-Based Software Analysis

Machine learning and deep learning have increasingly been applied to software engineering tasks. For example, Tufano et al. show that deep learning models can learn similarities between code fragments using different representations of source code [10]. Machine learning has also been explored for code review and software quality assessment, showing that automated techniques can support parts of the review process [6].

More recently, Large Language Models (LLMs) have become important in software engineering. LLMs can generate code, explain existing code, suggest improvements, and produce natural-language feedback. Hou et al. describe how LLMs are increasingly used across different software engineering tasks [5]. This makes them relevant for repository feedback, especially in an educational setting where feedback should not only identify problems, but also explain them.

The *AI repo feedback* [9] follows this direction. Instead of only applying fixed rules, it collects information from a repository and sends it to an AI model. The model then generates feedback about the repository contents, architecture, source code, build configuration, and version control history.

2.3 Repository-Level LLM Analysis and Content Selection

Using LLMs for repository-level analysis is different from using them on a single source file or a small code snippet. A repository can contain many files, multiple programming languages, external dependencies, generated code, and framework-specific build artifacts. This creates a practical problem: not all repository contents are useful for feedback generation.

Recent work such as RepoAudit shows that LLMs can be used for repository-level code auditing [3]. At the same time, this type of analysis introduces challenges such as token costs, context-window limitations, and hallucinations. These challenges are relevant to this thesis because the feedback tool also depends on deciding what information should be sent to the model.

This makes repository content selection an important part of the system. Including directories such as `vendor`, `node_modules`, `dist`, `build`, `coverage`, and `.next` can increase the amount of input without improving the generated feedback. In some cases, this content can even distract the model from the actual project code. Liu et al. show that language models do not always make effective use of long context, which supports the idea that simply providing more input is not always better [7].

Chapter 3

Method

This chapter describes how the existing repository feedback tool was analysed, modified, and evaluated. It explains the baseline analysis of the original tool, the design choices behind the modified version, and the evaluation approach used to compare the original and modified implementations.

3.1 Research Approach

This research follows a prototype-based evaluation approach. The goal is to improve an existing AI-based repository feedback tool [9] and evaluate whether the changes address the limitations found in the original implementation. This fits the principles of Design Science Research, where an artifact is designed, implemented, and evaluated in response to an identified problem [8].

The research process consisted of four main phases. First, the original tool was studied to understand its workflow and limitations. Second, an improved implementation was designed. Third, the tool was implemented by extending the original tool. Finally, the modified tool was evaluated by comparing it with the original implementation.

3.2 Analysis of the Original Tool

The first phase focused on understanding the existing repository feedback tool. The implementation was inspected to determine how it collects repository information, how it selects files, how it constructs prompts, and how it sends these prompts to the AI model.

The original tool was also executed on a small Java project to confirm that it worked correctly in its intended setting. This helped establish a baseline before modifications were made. During this phase, several practical issues were identified and fixed to make the system stable enough for further development.

The analysis focused on two main limitations. The first limitation was language dependence. The original implementation was mainly structured around Java repositories. The second limitation was scalability. The tool could send large amounts of repository content to the AI model, especially when analysing larger repositories.

3.3 Prototype Design

After the baseline analysis, an improved implementation was designed. The design focused on two improvement areas.

The first improvement area was multi-language support. The prototype was designed so that files could be routed to language-specific analysis functions based on their file extension. This made it possible to support Java, PHP, and TypeScript in a single workflow.

The second improvement area was repository content selection. Instead of analysing repository contents broadly, the prototype filters files and directories before sending information to the AI model. This

design choice was made to reduce unnecessary input and to focus the analysis on files that are more likely to contain meaningful project logic.

3.4 Prototype Implementation

The prototype was implemented by modifying the original shell script of the AI repo feedback tool developed by Spinellis [9]. The resulting modified implementation is publicly available as a GitHub fork created for this research [2]. The implementation added language detection, TypeScript support, PHP-specific improvements, filtering rules, file-size limits, and language-specific source code analysis functions. The AI model used for both the original tool and the modified implementation was kept the same: `gpt-4.1-mini`

The implementation was developed iteratively. After each major change, the tool was tested on example repositories to verify whether the change worked as expected. When the tool failed because of excessive AI input, the file selection strategy was adjusted to reduce the amount of content sent to the model.

3.5 Evaluation Procedure

The evaluation compares the original and modified versions of the tool. The comparison focuses on three aspects.

- **Language generality:** whether the tool can analyse repositories written in Java, PHP, and TypeScript.
- **Repository content selection:** whether the modified version avoids unnecessary files and directories during analysis.
- **Feedback usefulness:** whether the generated feedback remains meaningful after filtering and language-specific routing are applied.

The technical evaluation was performed by executing both the original and the modified versions of the tool on a number of software repositories. The generated reports were inspected to verify that the modified implementation correctly analysed repositories written in different programming languages and that repository filtering reduced the inclusion of irrelevant files.

To evaluate the usefulness of the generated feedback, a small user study was conducted with three computer science students. Each participant received feedback reports generated by both the original and the modified versions of the tool for two Java repositories. Since the original implementation does not support TypeScript, participants also evaluated feedback generated by the modified tool for a TypeScript repository. The reports were presented without indicating which version of the tool had generated them whenever a direct comparison was possible.

After reviewing the reports, the participants completed a short questionnaire in which they assessed the usefulness, relevance, and specificity of the generated feedback and were invited to provide qualitative comments [1].

The qualitative feedback from the questionnaires was analysed to identify recurring strengths and weaknesses in the generated reports. Rather than measuring user satisfaction quantitatively, the purpose of the user study was to determine whether the modifications affected the perceived quality of the generated feedback and whether the additional language support produced useful feedback for repositories that could not be analysed by the original implementation

Chapter 4

Design and Implementation

This chapter describes the changes made to the repository feedback tool itself. It explains how the original Java-centred workflow was extended with language-specific analysis, and how repository filtering and file selection were added to reduce unnecessary input to the AI model.

4.1 Design Overview

This chapter describes the design and implementation of the modified repository feedback tool [2]. In this thesis, the implemented tool extends the original tool with improved repository content selection and support for PHP and TypeScript in addition to Java.

The original tool already generated feedback for several parts of a repository, including repository contents, project architecture, source code quality, build specification, and commit history. The improved implementation keeps this general workflow. The main change is that the tool now makes more explicit decisions about which files should be analysed and how these files should be routed to language-specific analysis functions.

The design therefore focuses on two main concerns. The first concern is language support. Instead of treating the repository as mainly Java-based, the modified tool detects the programming language of files and applies a suitable analysis function. The second concern is scalability. Instead of sending too much repository content to the AI model, the tool filters files and directories that are unlikely to contribute useful feedback.

4.2 Original Tool and Design Direction

The original AI repo feedback tool developed by Spinellis [9] was purely designed for Java repositories. The modified implementation developed during this research [2] extends this design with support for PHP and TypeScript.

Another limitation was the way source files were selected. The original implementation focused on the largest source files, but did not sufficiently exclude directories such as dependencies, generated output, or framework build artifacts. This became a practical problem during testing on a larger PHP repository, where the tool generated too much input for the AI model.

For these reasons, the design direction changed from simply adding more file analysis to making the analysis more selective. The goal became to analyse files that are more likely to represent the actual project logic, while avoiding files that mostly add noise or unnecessary input.

4.3 Multi-Language Analysis

The modified implementation extends the language detection mechanism to support Java, PHP, and TypeScript. TypeScript support includes both `.ts` and `.tsx` files. The repository's main language is detected by counting source files for the supported languages:

```
detect_main_language()
{
local repo="$1"

local java_count=$(find "$repo" -name ".java" | wc -l)
local php_count=$(find "$repo" -name ".php" | wc -l)
local ts_count=$(find "$repo" ( -name ".ts" -o -name ".tsx" ) | wc -l)

if (( php_count > java_count && php_count > ts_count )); then
echo "php"
elif (( ts_count > java_count && ts_count > php_count )); then
echo "typescript"
else
echo "java"
fi
}
```

In addition to detecting the main language of a repository, the tool also detects the language of each selected source file. This is done using file extensions:

```
get_file_language()
{
local file="$1"

case "$file" in
*.java) echo "java" ;;
*.php) echo "php" ;;
.ts|.tsx) echo "typescript" ;;
*) echo "unknown" ;;
esac
}
```

This structure is important because repositories can contain more than one language. The tool should therefore not assume that every selected file belongs to the main language of the repository. By routing files individually, the modified implementation can apply language-specific prompts more accurately:

```
if [ "$lang" = "java" ]; then
report_source_code_java "$repo" "$path" "4.$counter"
elif [ "$lang" = "php" ]; then
report_source_code_php "$repo" "$path" "4.$counter"
elif [ "$lang" = "typescript" ]; then
report_source_code_typescript "$repo" "$path" "4.$counter"
fi
```

PHP and TypeScript support were implemented as separate analysis functions. For PHP, the tool extracts classes, interfaces, traits, and related declarations. For TypeScript, the tool extracts classes, interfaces, enumerations, type aliases, exported functions, and arrow functions. These extracted elements are then used to provide the AI model with a compact view of the project architecture.

4.4 Repository Content Selection

A major part of the implementation concerns repository content selection. During testing, it became clear that sending more repository content to the AI model does not automatically produce better feedback. Large repositories often contain files that are not part of the developers' own implementation, such as dependencies, generated files, build output, and framework artifacts.

The modified tool therefore excludes several common directories from source code analysis. These include `vendor`, `node_modules`, `.git`, `dist`, `build`, `out`, `coverage`, and `.next`. These directories were selected because they commonly contain dependency files, generated files, or output that is not useful for assessing the project's own code.

The file-selection step also applies a file-size limit and then selects the largest remaining source files:

```
find . -type f
( -name ".java" -o -name ".php" -o -name ".ts" -o -name ".tsx" )
-not -path "/vendor/"
-not -path "/node_modules/"
-not -path "/.git/"
-not -path "/dist/"
-not -path "/build/"
-not -path "/out/"
-not -path "/coverage/"
-not -path "/.next/"
-size -80000c
-printf "%s %p\n" |
sort -nr |
head -5
```

Files larger than 80,000 bytes are excluded from detailed source code analysis. This prevents unusually large files from dominating the AI prompt and reduces the chance of reaching model or quota limits.

After filtering, the remaining files are sorted by size and the five largest relevant files are selected for detailed source code analysis. This keeps the original idea of analysing representative files, but makes the selection more practical by first removing files that are unlikely to be useful.

This heuristic is simple and understandable, but it is not perfect. A large file is not always the most important file in a project, and an important file can sometimes be small. Later improvements could select files using more signals, such as directory location, commit activity, number of functions, or whether the file contains central project classes.

4.5 Practical Implementation Notes

The implementation was developed iteratively. After each major change, the tool was tested on example repositories to check whether the correct files were selected and whether the correct language-specific analysis function was used.

Additional debug output was added to support this process. The debug output shows which files are skipped because of the file-size limit and which language is detected for each selected file. This made it easier to understand the behaviour of the file-selection process during development.

One practical issue that remains is runtime. The tool waits between AI calls to avoid exceeding API or quota limits. This makes the tool safer to run, but it also means that a full analysis can take a long time.

Chapter 5

Evaluation

This chapter describes the evaluation of the modified tool. It reports the evaluation setup, the repositories tested, the observations about language support and repository filtering, and the user-based evaluation.

5.1 Experimental Setup

The purpose of the evaluation is to determine whether the improved implementation addresses the research question: to what extent improved repository content selection and multi-language support can improve the quality and generality of feedback generated by an AI-based repository analysis tool.

This evaluation focuses on three aspects.

- First, language support was evaluated to determine whether the modified implementation can analyse repositories written in more programming languages than the original version.
- Second, repository content selection was evaluated to determine whether the filtering strategy reduces unnecessary AI input by excluding irrelevant files such as dependencies and generated artifacts.
- Third, the generated feedback was evaluated through a small user study in which computer science students assessed the usefulness, relevance, and specificity of the generated reports.

In this evaluation, the modified tool developed during this research [2] was tested on selected repositories to validate the implementation. The original tool was first tested on a small Java repository to confirm that the baseline version worked in its intended setting. The modified tool was then tested on repositories containing PHP and TypeScript code. A larger PHP repository (Episciences) was especially useful because it exposed scalability issues in the original approach, where too much repository content could be selected for analysis and exceed the available AI quota.

Repository	**Main language**	**Version tested**	**Purpose**
Online Book Store	Java	Original and modified	Baseline test
Episciences	PHP	Modified	Larger PHP test case
Immich	TypeScript	Modified	TypeScript support test

Table 5.1: Repositories used in the evaluation.

5.2 Technical Evaluation

The first result is that the modified tool is more general than the original implementation. The original tool was purely designed around Java repositories, while the modified implementation supports Java, PHP, and TypeScript. The support is not fully language-independent, because each language still requires specific extraction rules and prompts. However, the structure is now easier to extend than before.

The second finding concerns repository content selection. During testing on a larger PHP repository, the tool generated too much input for the AI model and exceeded the available quota. This showed that

simply adding language support was not enough. If too much irrelevant repository content is included, the tool can still become inefficient or fail on larger repositories.

The filtering strategy reduced this problem by excluding common dependency and build directories such as `vendor`, `node_modules`, `dist`, `build`, `out`, `coverage`, and `.next`. A file-size limit was also added to avoid sending unusually large files to the AI model. These changes made the tool more selective and more practical to run during testing.

The third finding concerns runtime. Both the original and the modified implementations wait between AI calls in order to avoid quota or rate-limit problems. This makes the tool more stable, but also increases the total time needed to generate a complete report. Reducing the runtime of the tool is therefore left for future work.

5.3 User-Based Evaluation

To complement the technical evaluation, a small user study was conducted with three computer science students. The participants reviewed feedback generated by both the original and the modified versions of the tool for two Java repositories. In addition, they evaluated feedback generated by the modified tool for a TypeScript repository, since this language is not supported by the original implementation. After reviewing the reports, the participants completed a short questionnaire describing how useful, relevant, and specific they considered the generated feedback.

Overall, the participants reported mixed experiences with the generated feedback. Several observations were consistently identified as useful. For example, participants appreciated concrete security-related findings, such as the warning about a publicly accessible `check.php` file that could expose sensitive information. Suggestions regarding repository configuration, such as providing a `.env.example` file, were also considered useful when applicable. Furthermore, observations about project-specific naming conventions were generally recognised as accurate descriptions of the analysed repositories. Because only three participants took part in the study, the results are intended to provide qualitative insights rather than statistically significant evidence.

At the same time, participants identified several recurring weaknesses. A considerable part of the generated feedback consisted of generic positive remarks that were not supported by concrete examples from the repository. Some improvement suggestions were considered unhelpful or even inappropriate for the specific project. In addition, participants noted that recommendations such as improving documentation or adding CI/CD descriptions could apply to almost any repository and therefore contributed little repository-specific value.

Another recurring observation was that the generated feedback rarely referred to concrete functions, source files, or line numbers. As a result, participants found it difficult to determine why certain recommendations had been made or how strongly they were supported by the analysed source code.

Finally, participants reported that they observed little noticeable difference in the overall quality of the feedback generated by the original and modified versions of the tool. While the modified implementation successfully extended language support and improved repository content selection, these technical improvements did not lead to a clear improvement in the perceived usefulness of the generated feedback.

These findings suggest that the modifications successfully improve the generality of the repository feedback tool by extending its applicability to additional programming languages while maintaining comparable feedback quality. Although repository filtering and multi-language support make the tool applicable to a wider range of repositories, participants indicated that the overall style and specificity of the generated feedback remain mostly determined by the underlying language model. Future improvements should therefore focus on generating feedback that is more repository-specific and better supported by concrete evidence from the analysed source code.

Chapter 6

Discussion

This chapter discusses what the current findings mean for the research question. It reflects on the shift from analysing more repository content to selecting more relevant content, and discusses the limitations and remaining work of the project.

6.1 The Interpretation of the Findings

The research question of this thesis asks to what extent improved repository content selection and multi-language support can improve the quality and generality of feedback generated by an AI-based repository analysis tool.

The clearest improvement is in generality. The original tool was mainly designed around Java repositories, while the modified implementation supports Java, PHP, and TypeScript. Both the technical evaluation and the user study showed that the tool can now analyse repositories written in multiple programming languages without changing the overall analysis workflow. This makes the tool applicable to a wider range of repositories.

The interpretation of deeper file-level analysis also changed during the project. Initially, it seemed that deeper analysis would mean reading more files or analysing source code in more detail. During implementation, however, it became clear that analysing more content is not always useful. Large repositories often contain dependencies, generated files, and build artifacts that increase input size without improving feedback.

For this reason, deeper file-level analysis is interpreted in this thesis as better repository content selection. The modified tool tries to decide which files are worth sending to the AI model and which files should be ignored.

The user study provided additional insight into the quality of the generated feedback. Participants considered several observations to be useful, particularly when the tool identified concrete security-related issues or made project-specific observations that could easily be verified. However, they also noted that a considerable part of the generated feedback consisted of generic recommendations that could apply to many repositories. In addition, the reports rarely referred to specific files, functions, or code fragments to support the generated advice.

Overall, these findings indicate that the modifications made in this thesis mainly improve the applicability of the repository feedback tool rather than the quality of the generated feedback itself. The improvements in language support and repository content selection enable the tool to analyse a wider range of repositories more efficiently, but the overall quality and specificity of the generated feedback remain largely dependent on the underlying language model.

6.2 Limitations

First, the filtering strategy is based on simple heuristics. Excluding directories such as `vendor` and `node_modules` is usually reasonable, but there can be exceptions. Similarly, the file-size limit helps

reduce unnecessary input, but it can also exclude a file that contains important project logic.

Second, the quality of the generated feedback remains limited by the underlying language model. Although the modified implementation improves language support and repository content selection, the user study showed that the generated reports can still contain generic recommendations and are not always supported by concrete references to the analysed source code. As a result, improving the repository analysis pipeline alone is not sufficient to substantially improve the perceived quality of the feedback.

6.3 Implications and Next Steps

The current results suggest that repository content selection is an important part of AI-based repository feedback. Instead of sending as much repository content as possible to the AI model, the tool should try to select useful content and avoid files that add little value.

The user study suggests that future improvements should focus less on extending the repository analysis pipeline and more on improving the interaction with the language model. Although the modified implementation successfully analyses more programming languages and selects repository contents more effectively, participants indicated that the generated feedback could become more repository-specific and better supported by concrete evidence from the analysed source code.

Another future improvement is to make the tool faster. The current waiting period between AI calls protects against quota and rate-limit problems, but it also makes the tool slow. A better approach could use adaptive waiting, batching, or tracking of previous request times. The goal would be to reduce runtime without upsetting the AI limits.

Chapter 7

Conclusions

This chapter summarizes the conclusions of the thesis. It answers the research question based on the work completed so far and lists the main contributions.

7.1 Answers to the Research Question

This thesis investigated whether improved repository content selection and multi-language support can improve the quality and generality of feedback generated by an AI-based repository analysis tool.

The evaluation shows a clear improvement in the generality of the tool. While the original implementation primarily targeted Java repositories, the modified implementation also supports PHP and TypeScript. As a result, the tool can analyse a wider range of software repositories without requiring major changes to the overall workflow.

The second contribution concerns repository content selection. During development, it became clear that analysing more repository content does not necessarily improve the generated feedback. By filtering dependency directories, generated output, and unusually large files before analysis, the modified implementation is able to analyse larger repositories more reliably while reducing unnecessary input to the AI model.

The user study showed that the generated feedback contains both useful repository-specific observations and generic AI-generated recommendations. Participants considered concrete findings, such as security-related observations and project-specific comments, to be valuable. At the same time, they noted that some recommendations were too general and were not always supported by concrete examples from the analysed source code. Furthermore, participants observed little difference in the overall quality of the feedback generated by the original and modified versions.

Overall, the results indicate that the modifications primarily improve the generality and scalability of the repository feedback tool. Although they do not substantially change the perceived quality of the generated feedback, they allow the tool to analyse a broader range of repositories while making more effective use of the available AI context. These findings indicate that the remaining limitations of the tool are primarily related to the underlying language model rather than the repository analysis pipeline.

7.2 Contributions

The main contribution of this thesis is an improved implementation of the repository feedback tool, which has been released as an open-source GitHub fork [2]. The prototype adds multi-language support, language-specific analysis functions, repository filtering, and file-size limits.

A second contribution is the practical insight that AI-based repository feedback depends strongly on selecting useful input. The project showed that filtering irrelevant repository content is necessary for making the tool more scalable and practical.

7.3 Future Work

Future work could also improve the runtime of the tool. The current implementation waits between AI calls to avoid quota or rate-limit problems, but this makes the analysis slow. A better waiting strategy could make the tool faster while still respecting AI limits.

Another direction for future work is improving the quality of the generated feedback itself. The user study showed that while the modified implementation successfully improves language support and repository content selection, the generated feedback can still contain generic recommendations that are not always specific to the analysed repository. This suggests that the main limitation is no longer the repository analysis pipeline, but the interaction with the underlying language model. Future work could therefore investigate improved prompting strategies, retrieval of more repository-specific context, or mechanisms that require the model to support its recommendations with concrete evidence from the analysed source code.

Bibliography

[1] Hamzeh Akkad. Questionnaire form. https://forms.gle/xBj3BAHKgun9f4Pb7.

[2] Hamzeh Akkad. Ai repo feedback (modified version). `https://github.com/BaguetteSama/ai-repo-feedback-multi-language`, 2026. Fork of the AI repo feedback tool by Diomidis Spinellis, extended with PHP and TypeScript support and improved repository content selection.

[3] Jinyao Guo, Chengpeng Wang, Xiangzhe Xu, Zian Su, and Xiangyu Zhang. Repoaudit: An autonomous llm-agent for repository-level code auditing, 2025. URL `https://arxiv.org/abs/2501.18160`.

[4] Ahmed E. Hassan. The road ahead for mining software repositories. pages 48–57, 11 2008. doi: 10.1109/FOSM.2008.4659248.

[5] Xinyi Hou et al. Large language models for software engineering: A systematic literature review. *ACM Computing Surveys*, 2024.

[6] Harsh Lal and Gaurav Pahwa. Code review analysis of software system using machine learning techniques. In *2017 11th International Conference on Intelligent Systems and Control (ISCO)*, pages 8–13, 2017. doi: 10.1109/ISCO.2017.7855962.

[7] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024.

[8] Ken Peffers, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *Journal of management information systems*, 24(3):45–77, 2007.

[9] Diomidis Spinellis. Ai repo feedback. `https://github.com/dspinellis/ai-repo-feedback`, 2024.

[10] Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, and Denys Poshyvanyk. Deep learning similarities from different representations of source code. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18, page 542–553, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450357166. doi: 10.1145/3196398.3196431. URL `https://doi.org/10.1145/3196398.3196431`.