

A Dataset and Preliminary Study of Using GPT-5 for Code-change Impact Analysis

Katharina Stengg

Software Engineering Research Group
University of Klagenfurt
Klagenfurt, Austria
katharina.stengg@aau.at

Christian Macho

Software Engineering Research Group
University of Klagenfurt
Klagenfurt, Austria
christian.macho@aau.at

Martin Pinzger

Software Engineering Research Group
University of Klagenfurt
Klagenfurt, Austria
martin.pinzger@aau.at

Abstract—Understanding source code changes and their impact on other code entities is a crucial skill in software development. However, the analysis of code changes and their impact is often performed manually and therefore is time-consuming. Recent advancements in AI, and in particular large language models (LLMs) show promises to help developers in various code analysis tasks. However, the extent to which this potential can be utilized for understanding code changes and their impact is underexplored. To address this gap, we study the capabilities of GPT-5 and GPT-5-mini to predict the code entities impacted by given source code changes. We construct a dataset containing information about seed-changes, change pairs, and change types for each commit. Existing datasets lack crucial information about seed changes and impacted code entities. Our experiments evaluate the LLMs in two configurations: (1) seed-change information and the parent commit tree and (2) seed-change information, the parent commit tree, and the diff hunk of each seed change. We found that both LLMs perform poorly in the two experiments, whereas GPT-5 outperforms GPT-5-mini. Furthermore, the provision of the diff hunks helps both models to slightly improve their performance.

Index Terms—Code Change, Change Impact Prediction, LLM

I. INTRODUCTION

Modern software engineering is based on the frequent adaptation of source code to meet evolving requirements. To make these adaptations, software engineers need to understand the code, its changes, and the potential impact of those changes on other entities. Tao *et al.* [1] found that over 43% of their interviewed software engineers deal with code changes daily, and 36% even more often. Furthermore, they found that there is no adequate tool support to assess the quality, understanding, or decomposition of changes. A study by Bacchelli and Bird [2] focused on modern code review and reported similar findings, concluding that there is a need for tool improvement and that it is important to understand changes to provide good reviews. Recent studies explored the potential of using large language models (LLMs) for automating several code-related [3] and code-change related [4] tasks, such as vulnerability detection

and repair [5] and secure code generation [6]. However, to the best of our knowledge this potential has not been explored for analysing the impact of code changes. According to Angerer *et al.* [7], the change impact denotes the potential effects of a software change.

In this work, we explore the capabilities of two state-of-the-art LLMs, namely GPT-5 and its smaller version GPT-5-mini, for code change-impact prediction. Furthermore, we extend the dataset *ALEXANDRIA* from Yan *et al.* [8], as their dataset contains only changed method pairs and does not consider attribute or class changes. For our preliminary evaluation, we performed two experiments. The first experiment, named *Basic*, provides the LLM with the link to the parent commit tree and the fully qualified names of the seed changes. A seed change denotes the code changes that trigger changes in other code entities. The second experiment, named *Diff*, extends the first experiment by providing the diff hunks of the seed changes. With the two experiments, we aim to answer the following two research questions:

- RQ1: What is the precision, recall and F1-score of GPT-5 and GPT-5-mini in code-change impact prediction?
- RQ2: To what extent does the inclusion of the diff hunks of the seed change improve code-change impact prediction?

The results with 40 commits randomly selected from popular Java projects on GitHub show that GPT-5 outperforms GPT-5-mini in both experiments. LLMs benefit from knowing the commit that contains the changes. In addition, the provision of the diff hunks helps both models to slightly improve their performance. This paper makes the following contributions:

- 1) The *Alexend* dataset, which contains detailed information about code changes and their impact.
- 2) First results of the performance of GPT-5 and GPT-5-mini for code-change impact prediction.

II. RELATED WORK

Hanam *et al.* [9] proposed a tool called *SEMCIA*, which improves the code reviewing process for developers in terms of time and performance. Furthermore, they defined semantic change relations and performed a user study comparing traditional change impact analysis techniques with their proposed

This research was funded in whole or in part by the Austrian Science Fund (FWF) 10.55776/P36698. For open access purposes, the author has applied a CC BY public copyright license to any author accepted manuscript version arising from this submission. The research was supported by the Austrian ministries BMIMI, BMWET and the State of Upper Austria in the frame of the SCCH COMET competence center INTEGRATE (FFG 892418).

approach. In 2023, Lin *et al.* [10] proposed a model based on the Text-To-Text-Transfer Transformer (T5) model. Their model is pre-trained on five tasks. For evaluation, they fine-tuned it for each of the five code-change related tasks. These tasks are commit message generation, just-in-time comment update, and defect prediction as well as code-change quality and code review generation. They proposed and used a large-scale dataset called CodeChangeNet. This dataset consists of code-change and commit message pairs of six programming languages. In their work [8], Yan *et al.* propose their impact analysis approach *ATHENA*, which is based on GraphCodeBERT and structural dependence graphs. They also provide a novel code-change impact analysis dataset called *ALEXANDRIA*. This dataset contains method-level Java production code changes and meta-information. Compared to our dataset, *ALEXANDRIA* indicates no clear seed method, therefore they perform their impact analysis experiments treating all method-level changes of a commit as query and impact. The paper also states that this may not be a realistic scenario. This is a gap that we address in this work.

In their work [4], Fan *et al.* compare various LLMs and small-pretrained models (*e.g.*, CCT5 [10]) for code review generation, commit message generation and just-in-time comment update. They investigated whether in-context learning (ICL) or parameter-efficient fine-tuning (PEFT) improves performance, and compared two input formats, the code itself and the code diff, aiming to find the best combinations per code-change task. The authors state that LLMs offer potential for code-change related tasks, but did not evaluate this potential for code-change impact analysis.

III. FINE-GRAINED CHANGE IMPACT DATASET

Our dataset *Alextend* is an extension of the *ALEXANDRIA* dataset proposed by Yan *et al.* [8]. The original dataset contains method-level changes and corresponding information, such as the repository name, commit hash, parent commit hash, file path, etc. per changed method.

From *ALEXANDRIA*, we randomly selected 40 commits from all Java projects that satisfied the following criteria:

- A minimum of one and a maximum of five changed Java source code files
- A minimum of two changes, that are not comment, import, or systematic changes (identical modifications across multiple code entities)

Table I lists the number of selected commits (*#Commit*) and the number of manually identified change pairs (*#Change pair*) per project. In total, our dataset contains 40 commits with 192 code-change pairs that spread among 14 projects.

For each commit, our dataset stores the attributes *repo*, *commit_hash*, and *parent_commit_hash*, that list the repository name and the hash of the commit and its parent commit. The *github_link* attribute provides the URL of the repository at the current commit on GitHub. Furthermore, the dataset contains the field *java_class_count*, which represents the amount of changed Java source code files per commit.

TABLE I: Descriptive statistics of the 40 commits.

Project	#Commit	#Change pair
apache/ant-ivy	12	66
apache/commons-codec	1	4
apache/commons-compress	3	4
apache/commons-configuration	4	23
apache/commons-io	1	2
apache/commons-lang	1	2
apache/commons-math	3	12
apache/commons-net	3	15
apache/commons-scxml	1	1
apache/commons-vfs	4	10
apache/giraph	2	6
apache/gora	1	8
apache/opennlp	3	6
apache/parquet-mr	1	33
Total	40	192

Compared to *ALEXANDRIA*, we include not only method, but also class and attribute changes. We first extracted all Java source code changes that do not alter comments or imports per commit. We use the fully qualified names¹ to make our code entities uniquely identifiable. For example, the changed method *foo(int)* is identified as *org.test.Class1.foo(int)*. We do not use fully qualified names for method parameters for improved readability and reduced token count. If a nested class declaration or body was changed, the fully qualified name includes both the enclosing and the nested class, *e.g.* *org.test.Class1\$nestedClass.method1()*.

Furthermore and compared to *ALEXANDRIA*, for each commit, our dataset contains the type of each change, the change pairs, the seed changes, and the diff hunk for the seed changes. Regarding the change types, change pairs, and seed changes, two co-authors independently and manually annotated the code changes of each commit. Thereafter, they met and discussed the results until they reached agreement on all annotations. In the initial phase, seed changes were often misinterpreted, therefore we defined them as the primary logical changes that trigger the remaining changes.

For the annotation of the *change types*, the two co-authors used the change type taxonomy introduced by Fluri and Gall [11]. Their taxonomy distinguishes between declaration and body changes. The main categories for classes are *Class Declaration Changes* and *Class Body Changes*. Class body changes can be split up into *Method Declaration Changes*, *Method Body Changes*, and *Attribute Declaration Changes*. We skipped the types *Access Modifier Changes* and *Final Modifier Changes*. A method body change is also a class body change, but we recorded it solely as method body change as we classified changes only by their most specific type. The results are stored in the column *change_categories* of our dataset.

For the annotation of the *seed changes* and *change pairs*, the two co-authors analysed each code change for its relationship to the other changes. Listing 1 shows an example of a change in Java code. The class attribute *loadCounter* has been changed from *int* to *AtomicInteger*. Because an *AtomicInteger*

¹<https://docs.oracle.com/javase/specs/jls/se17/html/jls-6.html#jls-6.7>

must be initialised before it can be used, the constructor creates a new instance. *loadCounter* and *PropertiesConfigurationLayout(PropertiesConfigurationLayout)* form a change pair, whereas *loadCounter* is a seed-change, *i.e.*, the change that triggers the change in the constructor. This dependency is of semantic nature rather than of syntactic nature, making it hard for existing approaches to capture. Our dataset stores the change pairs of each commit as a semicolon-separated list in *change_pairs* and the seed changes in *seed_changes*.

Listing 1: Code-change impact example

```
@@ -133,7 +134,7 @@ public class
    ↪ PropertiesConfigurationLayout
...
-     private int loadCounter;
+     private final AtomicInteger loadCounter;
...
public PropertiesConfigurationLayout (
    ↪ PropertiesConfigurationLayout c)
{
+     loadCounter = new AtomicInteger();
```

Our dataset also contains the minimal *diff hunk* of each seed change of a commit. The minimal diff hunk is exported from GitHub and consists of:

- An identifier of the changed entity;
- Lines starting with + that show added code;
- Lines starting with - that show removed code;

We removed any further diff information to save input tokens when using the diff hunk in our prompts. The git diff hunk is stored in the column *seed_changes_diff*. Finally, we also stored the commit message of each commit in the column *commit_message*.

IV. PROMPT DEFINITION

Listing 2 shows the basic prompt created for our experiments. We followed guidelines and best practices from other research papers and OpenAI [12]. In their work [13], Salewski *et al.* found that telling an LLM to act as a domain-expert can lead to better results. Therefore, all our prompts start with the impersonation of the LLM as an intelligent assistant that helps with the analysis of Java source code changes and their impact on other code entities. The prompt then states instructions and rules, as LLMs benefit from precise information. The specified input variables *github_link_parent* and *seed_changes* are dynamically updated during the experiments. Finally, we define the JSON output format and provide two examples, one with an impacted method and attribute and one with no impacted entities.

Wang *et al.* [3] tested prompt engineering techniques for code-related tasks on various models. They provided guidelines for the choice of LLM and prompt engineering technique and highlighted the importance of adapting the prompt to the strengths and weaknesses of the chosen LLMs. To optimize our prompt for the use with GPT-5 and GPT-5-mini, the prompt optimizer from OpenAI was used².

Listing 2: Basic prompt optimized for GPT-5 models

```
You are an intelligent assistant designed to help
    ↪ developers and code reviewers analyze Java
    ↪ source code changes and assess their impact
    ↪ on related code entities.

## Instructions
You will be provided with:
- A link to a parent commit in a GitHub repository.
- A list of seed changes made to the code.
Your task is to:
- Identify every Java code entity affected by these
    ↪ seed changes; specifically, those entities
    ↪ that must be modified as a direct result of
    ↪ the seed changes.

Follow these rules strictly:
1. For each impacted code entity, list its fully
    ↪ qualified name (FQN). For nested or inner
    ↪ classes, use the fully qualified name of the
    ↪ outer class, followed by a dollar sign ('$'),
    ↪ then the inner class name, and then the
    ↪ changed entity (e.g., 'com.example.Outer$
    ↪ Inner.changedEntity').
2. Do not use fully qualified names for method
    ↪ parameters; only for Java classes, interfaces
    ↪ , methods, attributes, and variables.
3. Do not include the seed changes themselves in the
    ↪ output; any entity listed as a seed change
    ↪ must be excluded.
4. Exclude any changes to test files, '.md' files,
    ↪ or '.xml' files; your analysis should focus
    ↪ exclusively on Java source code files.
5. You must only view and analyze the parent commit
    ↪ provided; reviewing or utilizing information
    ↪ from any follow-up or subsequent commits is
    ↪ strictly forbidden.
6. If a code entity is renamed due to the seed
    ↪ changes, list only the new fully qualified
    ↪ name in your output.
7. Impacted entities must be those that must change
    ↪ *because of* the seed change, not the seed
    ↪ change itself.

If ambiguity or uncertainty occurs, do not ask for
    ↪ clarification; choose and report the entities
    ↪ most likely to be impacted.

## Inputs
- **Commit link:** {{github_link_parent}}
- **Seed changes:** {{seed_changes}}

## Output Format
Return a JSON object with a single property named "
    ↪ impacted_entities", whose value is an array
    ↪ of strings. Each string should be the fully
    ↪ qualified name of an impacted code entity. If
    ↪ there are no impacted code entities, return
    ↪ an empty array for "impacted_entities".

**Example:**
'''json
{"impacted_entities": ["com.example.Foo.someMethod(
    ↪ String, int)", "com.example.bar.attribute1"]}
'''

**Example with no impacted entities:**
'''json
{"impacted_entities": []}
'''
```

²<https://platform.openai.com/chat/edit?models=gpt-5&optimize=true>

V. PRELIMINARY EVALUATION

This section describes the preliminary evaluation of our proposed approach. We used our dataset *Alextend* and the models GPT-5 and GPT-5-mini to evaluate their performance in predicting the impact of given code changes. The precise versions of the models are **gpt-5-2025-08-07** (GPT-5) and **gpt-5-mini-2025-08-07** (GPT-5-mini). GPT-5 was trained on information up until Sep 30, 2024 and GPT-5-mini up until May 31, 2024. Both models have a context window of 400,000 tokens. As it is practically not feasible to equip our prompts with the whole codebase of a Java project, we decided to enable the tool web search³, such that the models can look up the provided commit tree on GitHub.

The models’ responses are compared with our labelled ground truth impact set computed from the change pairs in our dataset. More specifically, the ground truth impact set contains all direct and indirect dependent code entities of the seed changes in a commit. For each response, we calculate precision, recall, and F1-score per commit based on exact matches of the fully qualified names of impacted code entities. We implemented our experiments in a Python script using the `openai`⁴ library.

A. Using the Basic Prompt

For answering RQ1, our Python script iterated over the 40 commits in our dataset and for each commit composed and sent the *Basic* prompt shown in Listing 2 to GPT-5 and to GPT-5-mini. This prompt contains the link to the parent commit tree and the fully qualified names of the seed changes. The responses were stored in a Huggingface dataset. In addition, logs of responses were stored to files for subsequent analysis.

Figure 1 visualises the precision, recall, and F1-score of experiment *Basic* for the 40 commits of the dataset.

Overall, the box plots show that both models perform poorly in this setting, with median values of 0.0 for precision, recall, and F1-score. Comparing the plots for GPT-5 and GPT-5-mini, we see that GPT-5 outperforms GPT-5-mini in all three metrics. The third-quartile scores (25% of the tested samples) of GPT-5 show 0.27 for precision, 0.18 for recall, and 0.22 for F1-score compared to 0.0 for all three metrics for GPT-5-mini

B. Adding Diff Hunks

The second experiment aims to answer RQ2 and extends the *Basic* prompt with the minimal diff hunk of the seed changes in a commit, as shown in Listing 3.

Listing 3: Adding the diff hunks of seed changes to the prompt for experiment *Diff*.

```
- The code diff from git (just the minimal
↳ hunk: identifier of changed entity and
↳ changed lines (add is indicated via +; delete
↳ via -)).
...
- **Code diff:** {{seed_changes_diff}}
```

Figure 2 shows the box plots of the F1-scores of GPT-5 and GPT5-mini and compares them with the F1-scores from experiment *Basic*. For experiment *Diff*, both models again achieved a median F1-score of 0.00. But compared to the *Basic*, the values for the third-quartile scores increased for both models. GPT-5 achieved a Q3 F1-score of 0.4, while GPT-5-mini achieved 0.04. In comparison, in experiment *Basic*, GPT-5 achieved a F1-score of 0.22 and GPT-5-mini of 0.00. These results indicate that code-change impact prediction improves when the LLM is not only provided with the fully qualified name of the changed code entity but also with the information about the code changes in the form of diff hunks.

C. Log Analysis

For each experiment, we analysed the logs, focusing on the web search calls of the two models to gain further insight into web information retrieval. Overall, the logs show that for each prompt, both models performed various web search tool calls to different websites. We also found that neither model queried the current commit or the code changes in it. They only accessed the source code denoted by the parent tree, *i.e.*, they followed the instructions provided in the prompt.

For experiment *Basic*, GPT-5 always queried at least once a folder at the parent tree and GPT-5-mini did so for 75% of the tested commits. GPT-5 wrongly queried the parent commit, which contains the code changes of the previous commit, in two samples. For experiment *Diff*, GPT-5 always queried at least once a folder at the parent tree except for one sample, where it neither queried the parent tree nor any folder. GPT-5-mini queried no parent tree for four commits, and did not query any folder of the parent tree for 22 of the 40 tested commits. This partially explains the poorer performance of GPT-5-mini.

D. Costs

The costs per 1M tokens for GPT-5 are \$1.25 for input, \$0.125 for cached input, and \$10.00 for output. In comparison, 1M input tokens using GPT-5-mini cost \$0.25, cached input tokens \$0.025, and output tokens \$2.00. 1k web search tool calls costs \$10. Table II lists the costs of the two experiments presented in this paper in US dollars (\$).

We observe that the web search tool is the main cost factor, with a total cost of \$20.91 for 2,091 calls. Another observation is that using GPT-5 costs almost twice the price of GPT-5-mini. The two experiments resulted in a total cost of \$29.75.

³<https://platform.openai.com/docs/guides/tools-web-search?api-mode=responses>

⁴<https://github.com/openai/openai-python>

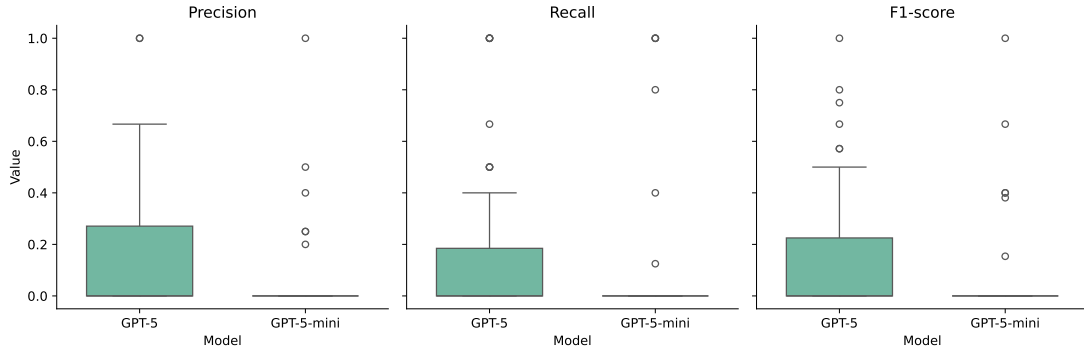


Fig. 1: Precision, recall, and F1-score of all 40 commits per model for the experiment *Basic*

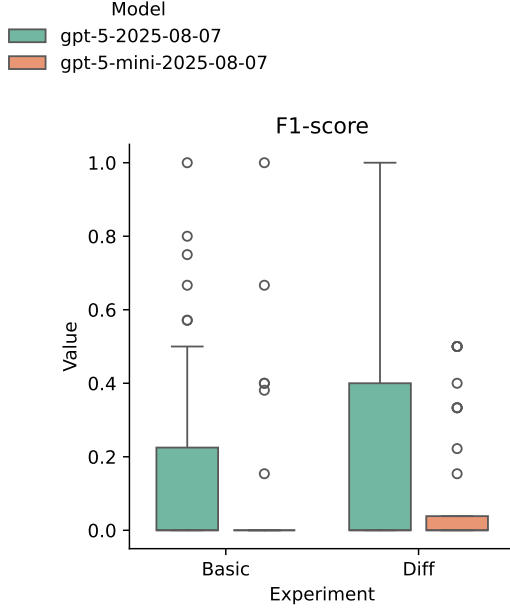


Fig. 2: F1-score of all 40 commits per model for the experiments *Basic* and *Diff*

TABLE II: Total costs of the experiments per model in US dollars (\$)

Model	Price (\$)				
	Input	Cached Input	Output	Web Search	Total
GPT-5	4.41	0.04	3.30	11.79	19.54
GPT-5-mini	0.68	0.01	0.40	9.12	10.21
Total	5.09	0.05	3.70	20.91	29.75

E. Threats to Validity

There is a potential bias in the experiments due to the uncertainty of the training data of the LLMs. We can not be sure whether they already used the selected repositories in their training. Similarly, our sample of 40 commits might not be representative. We plan to address both threats in our future work. Another threat of this approach is the use of the OpenAI web search feature. We mitigated this threat by analyzing the

logs with a focus on the links queried by web search. Non-determinism of LLMs is another potential threat to the validity of our results. To mitigate the risk, we plan on repeating our experiments ten times for evaluation. The dataset was labelled by two co-authors, which could introduce bias. We mitigated this bias by discussing all differences in the labeling until consensus was reached. We used commits with a maximum of five changed Java source code files. This can affect our dataset, as we leave out certain commits. But on the other hand, this reduced the bias introduced by very large changes that span multiple change requests. Another potential threat concerns the prompt itself, as slightly different instructions, for example the change of one word in the prompt, can lead to unforeseen results.

VI. CONCLUSIONS AND FUTURE WORK

We presented a preliminary study of the capabilities of GPT-5 and GPT-5-mini for code-change impact prediction. For that, we also curated the novel dataset *Alextend*, that is built upon 40 commits from the *ALEXANDRIA* dataset [8]. Compared to *ALEXANDRIA*, our dataset provides code changes, their type, change pairs, and information about the commit’s seed-change as well as the corresponding commit message. Using our dataset, we experimented with two prompts, whereas the *Basic* prompt contains the fully qualified name of the seed change and the *Diff* prompt adds the diff hunks. In both settings, the results show that GPT-5 and GPT-5-mini performed poorly, whereas GPT-5 outperformed GPT-5-mini. Furthermore, the provision of the diff hunks showed to help both models to slightly improve their performance.

In ongoing and future work, we first plan to perform our experiments with recent coding agents, such as Claude Code⁵ or Codex⁶. Furthermore, we plan to integrate sophisticated code analysis tools to provide the models with control and data flow information. Finally, we plan to extend our dataset to contain the information from more commits.

⁵<https://www.claude.com/product/claude-code>

⁶<https://openai.com/codex/>

REFERENCES

- [1] Y. Tao, Y. Dang, T. Xie, D. Zhang, and S. Kim, 'How do software engineers understand code changes? An exploratory study in industry', in *Proceedings of the ACM SIGSOFT 20th International symposium on the foundations of software engineering*, 2012, pp. 1–11.
- [2] A. Bacchelli and C. Bird, 'Expectations, outcomes, and challenges of modern code review', in *2013 35th International Conference on Software Engineering (ICSE)*, IEEE, 2013, pp. 712–721.
- [3] G. Wang et al., 'Do advanced language models eliminate the need for prompt engineering in software engineering?', *ACM Transactions on Software Engineering and Methodology*, 2024.
- [4] L. Fan, J. Liu, Z. Liu, D. Lo, X. Xia, and S. Li, 'Exploring the Capabilities of LLMs for code change-Related Tasks', *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 6, pp. 1–36, 2025.
- [5] X. Zhou, S. Cao, X. Sun, and D. Lo, 'Large language model for vulnerability detection and repair: Literature review and the road ahead', *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 5, pp. 1–31, 2025.
- [6] M. Bruni, F. Gabrielli, M. Ghafari, and M. Kropp, 'Benchmarking prompt engineering techniques for secure code generation with gpt models', in *2025 IEEE/ACM Second International Conference on AI Foundation Models and Software Engineering (Forge)*, IEEE, 2025, pp. 93–103.
- [7] F. Angerer, A. Grimmer, H. Prähofer, and P. Grünbacher, 'Change impact analysis for maintenance and evolution of variable software systems', *Automated Software Engineering*, vol. 26, no. 2, pp. 417–461, 2019.
- [8] Y. Yan, N. Cooper, K. Moran, G. Bavota, D. Poshyanyk, and S. Rich, 'Enhancing code understanding for impact analysis by combining transformers and program dependence graphs', *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 972–995, 2024.
- [9] Q. Hanam, A. Mesbah, and R. Holmes, 'Aiding code-change understanding with semantic change impact analysis', in *2019 IEEE international conference on software maintenance and evolution (ICSME)*, IEEE, 2019, pp. 202–212.
- [10] B. Lin, S. Wang, Z. Liu, Y. Liu, X. Xia, and X. Mao, 'Cct5: A code change-oriented pre-trained model', in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 1509–1521.
- [11] B. Fluri and H. C. Gall, 'Classifying change types for qualifying change couplings', in *14th IEEE International Conference on Program Comprehension (ICPC'06)*, IEEE, 2006, pp. 35–45.
- [12] OpenAI, "Best practices for prompt engineering with the OpenAI API", <https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-the-openai-api> (accessed Nov. 10, 2025)
- [13] L. Salewski, S. Alaniz, I. Rio-Torto, E. Schulz, and Z. Akata, 'In-context impersonation reveals large language models' strengths and biases', *Advances in neural information processing systems*, vol. 36, pp. 72044–72057, 2023.
- [14] S. Ouyang, J. M. Zhang, M. Harman, and M. Wang, 'An empirical study of the non-determinism of chatgpt in code generation', *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 2, pp. 1–28, 2025.
- [15] R. Tufano, L. Pascarella, M. Tufano, D. Poshyanyk, and G. Bavota, 'Towards automating code review activities', in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, IEEE, 2021, pp. 163–174.
- [16] C. Marsavina, D. Romano, and A. Zaidman, 'Studying fine-grained co-evolution patterns of production and test code', in *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*, IEEE, 2014, pp. 195–204.